MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
"IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE"

Liubov Oleshchenko

BIG DATA AND ANALYTICS

Textbook

Approved by the Academic Council of Igor Sikorsky Kyiv Polytechnic Institute
as a textbook for students studying in the specialty F2 «Software Engineering»

Electronic online educational publication

KYIV
PROSVITA LTD
2026

UDC 004.65+004.89](075.8)
O43

Approved by the Academic Council of Igor Sikorsky Kyiv Polytechnic Institute
(protocol No.11 dated 10.11.2025)

Reviewers: ***Volodymyr Kyselov***, Doctor of Technical Sciences, Professor, Honored Worker of Education of Ukraine, Director of the Educational and Scientific Institute of Municipal Management and Urban Economy, Volodymyr Vernadskyi Taurida National University

Yaroslav Sokolovskyi, Doctor of Technical Sciences, Professor, Excellent Education Worker of Ukraine, Professor of the Department of Computer Design Systems, Lviv Polytechnic National University

Managing editor: ***Viktor Legeza***, Doctor of Technical Sciences, Professor, Professor of the Department of Computer Systems Software of Igor Sikorsky Kyiv Polytechnic Institute

Editor -in-chief: ***Serhii Syrota***, Candidate of Technical Sciences, Associate Professor, Department of Applied Mathematics of Igor Sikorsky Kyiv Polytechnic Institute

Oleshchenko L. M.

O43 Big data and analytics [Electronic resource]: textbook for higher education students studying in the specialty F2 «Software Engineering» / L.M. Oleshchenko; Igor Sikorsky Kyiv Polytechnic Institute. – Electronic text data (1 file 9,64 MB). – Kyiv: Prosvita Ltd, 2026. – 317 p.

ISBN 978-617-7010-42-4

The basics of big data analytics and its role in decision-making in business, science, and manufacturing are considered. Software methods for storing, processing, and analyzing big data are described, including Hadoop, Spark, Kafka, and Flink technologies. Examples of using the Python programming language for preprocessing and analytics of big data are given. The basic methods of visualization and statistical analysis of big data, machine learning algorithms, and artificial intelligence for big data analytics systems are considered. For students studying in specialty F2 "Software Engineering", the textbook provides theoretical knowledge and practical skills for solving professional and scientific problems that require processing and analytics of big data.

UDC 004.65+004.89](075.8)

ISBN 978-617-7010-42-4

CONTENTS

PREFACE

Big data analytics plays a critical role in defining strategies and making decisions in a variety of industries, from business and technology to science and medicine.

Big data software analytics allows modern companies and organizations to make decisions based on objective data and facts, instead of intuition or hypotheses, to identify trends and relationships in large amounts of data.

Big data analytics allows companies to optimize business processes, reduce costs, and increase productivity by creating personalized offers, products, and services for customers.

Big data analytics helps identify new opportunities and create innovative products, services, and solutions for customers.

Big data analytics can help predict future events, such as computer system failures, risks, and other situations, and take measures to prevent them. Big data is used in scientific research, market analysis, and sociological studies to understand various trends.

The textbook is intended for students studying in specialty F2 "Software Engineering".

The purpose of this textbook is to provide theoretical knowledge and practical skills in using basic technologies and software methods storage and analysis of big data.

The textbook examines the capabilities of the Python programming language for preprocessing and analytics of big data.

As part of the training for students, the discipline "Big Data and Analytics" is taught, the subject of which is software methods and technologies of big data analytics for solving professional and scientific problems.

After mastering the discipline "Big Data and Analytics", the learning outcomes are:

– **knowledge**:

- big data storage methods and big data distributed computing technologies;
- features of the functioning of the Hadoop software platform and the Hadoop distributed file system (HDFS);

- Hadoop stack components, including the YARN resource and task management system, HDFS, and the MapReduce programming model;
- parallel programming methods using built-in Spark libraries;
- Spark Streaming and Sliding Window Analytics, Spark GraphX & Graph Analytics;
- Kafka and Apache Flink technologies for streaming data analytics;
- software methods and tools for big data visualization;
- Big Data cloud provider technologies;
- software methods of statistical analysis and machine learning algorithms for big data analytics.

– **skills**:

- use big data analytics technologies using the capabilities of the Python programming language;
- use distributed computing technologies for big data analytics;
- apply the acquired knowledge to conduct scientific research, further scientific and professional activities.

Studying the discipline "Big Data and Analytics" contributes to the formation of competencies in higher education students necessary for solving practical problems of professional activity related to big data analytics:

- the ability for abstract thinking, analysis and synthesis;
- ability to conduct research at an appropriate level;
- the ability to develop and implement scientific or applied software engineering projects.

Big Data and Analytics Textbook consists of 2 chapters and 12 topics, which provide theoretical information, a list of theoretical questions for self-testing, a list of tasks for independent study, and a list of recommended literature.

CHAPTER I.
BIG DATA ANALYTICS LIFECYCLE

Topic 1. Classification and methods of big data computing

In today's world, big data plays an important role as the volume and variety of data is constantly growing. This data can be used to gain new knowledge, make decisions, and improve various processes. Connected IoT devices are driving an increase in the volume and variety of data, opening up new opportunities to collect and use information from the real world. Data can be structured (such as database tables) or unstructured (such as text, images, audio, video). Cloud computing technology allows access to computing resources over the Internet and is used to provide data processing, storage, and collaboration. Fog computing technology is designed to provide computing resources and data processing at the "fog" level, closer to the data source, ensuring fast processing and low latency.

1.1. Big data. The growth of data in the modern world

The concept of "Big Data" does not have a strictly defined numerical value, as the amount of data that can be considered "big" depends on the context, available resources and technological capabilities at a given time. For example, Paul Zikopoulos of IBM stated that to be classified as big data, it is necessary to have a volume of 200 to 600 terabytes (1 terabyte (TB) is a unit of measurement of data volume, equivalent to 1 trillion (10^12) bytes). The International Data Corporation (IDC) believes that a data set of 100 terabytes can be classified as Big Data. If the data is streaming, its volume can be less than 100 terabytes, but still be considered Big Data, provided that the volume of this data increases by more than 60% every year. According to the National Institute of Standards and Technology (NIST), "The concept of big data involves dividing data systems into independent resources that are linked horizontally, allowing for the scalability needed to efficiently process large amounts of information." Big data can be considered when the volume is so large that standard processing and analysis methods become inadequate or unusable. This can be, for example, when the volume of data is measured in petabytes (1 petabyte (PB) is a unit of measurement of data volume that is equivalent to 1 quadrillion (10^15) bytes or 1,024 exabytes) or when data processing requires the use of distributed computing systems.

Let’s look at some real-world examples of big data generation. For example, an Airbus A380 engine generates 1 petabyte of data during a flight from London to Singapore. The Large Hadron Collider (LHC) creates 1 gigabyte of data every second. The world’s largest radio telescope, the Square Kilometre Array (SKA), produces 20 exabytes of data every day, which is 20 billion gigabytes per day.

The Internet of Things collects data from various sensors. This data can come from, for example, temperature and humidity sensors used in agriculture. Today, sensors are installed almost everywhere: in smartphones, cars, jet engines and household appliances. The number of devices with sensors is growing every year, which contributes to the rapid increase in the volume of big data.

For a rough understanding, big data is usually characterized by the term "3V" , which indicates three main properties or aspects of such data: volume, variety, velocity. Big data refers to large amounts of information that exceed the capabilities of traditional databases and data processing tools. In this context, big data can be data volumes reaching several terabytes, petabytes, or even more. Big data can include a variety of data types, such as text, numeric, video, audio, image files, social media posts, sensor data, geospatial data, etc., which defines its diversity.

The speed of data flow is also an important characteristic of big data. Data can flow at high speeds, for example, This can be data generated in real-time (real-time data) that needs to be processed and analyzed without delay.

In addition to the "3Vs", some experts add a "V" for the property "Validity", which refers to the correctness and accuracy of the data, and a "V" for the property "Volatility", which refers to the change in the volume and nature of data over time.

Today, with the growth of computing resources and technological capabilities, some large companies can process volumes of data that were considered large a few years ago, using efficient tools and distributed systems. The growth of data is exponential and is called "data explosion". This requires the development of new technologies for collecting, storing, processing and analyzing data. Processing and analyzing big data can help to obtain new knowledge for various industries, from business to medicine, transportation and science.

Storage and analytics of big data bring with them a number of

challenges related to their volume, variety and speed of arrival. In particular, processing and storing large amounts of data require powerful computing resources and infrastructure, which can be expensive. Storing big data can lead to significant costs for server hardware, disks and data storage systems. Unstructured data (such as text, images, video) requires specialized processing and analysis methods, which can be a difficult task. Integrating data from different sources can be problem due to the variety of data formats and structures.

Some applications and complex software systems require real-time data analysis, which requires speed and low latency in processing. Processing and storing big data can raise questions about protecting data from unauthorized access and cyberattacks. Examples of the dangers of storing private data include unauthorized access to personal information, identification data, financial details, and medical history. This can lead to the theft of personal information and its use for fraud, cyberattacks, or even blackmail. In addition, insufficient data security can lead to a breach of confidentiality, which can have a negative impact on the reputation of an individual or organization.

The increase in the volume of big data poses threats such as potential privacy and data security issues, the complexity of managing and analyzing large amounts of information, the possibility of erroneous or inaccurate conclusions due to excessive data volume, and the need for investment in high-tech infrastructure and personnel. Large volumes of data can create problems with their efficient storage, backup, and archiving, and may also require improved data processing and analysis methods to obtain valuable information.

Processing and analyzing big data may require the use of specialized algorithms and software tools that may be difficult to access or require specialized training.

1.2. The growth of Internet of Things devices

The growth of big data is associated with the expansion of Internet of Things (IoT) devices, an increase in the number of devices equipped with sensors and capable of generating data. IoT describes a network of physical objects (devices, sensors, devices) that are equipped with technology to collect and exchange data via the Internet without direct human interaction. These objects can collect, transmit and analyze information that may include

various parameters and actions (sensors in cars, household appliances, medical devices, etc.). Each such device can generate large amounts of data even in a short period of time. For example, these can be smartphones, medical devices, smart home appliances, IoT-enabled cars, etc. The Internet of Things, created in 1999 by Kevin Ashton, has been attracting more and more attention over time due to the increase in the number of connected devices and, as a result, the increase in the volume of data. In the era of big data, recording data from multiple environments and from different users is extremely valuable from both a statistical and a business and economic perspective. Nowadays, almost every device present in everyday life contains embedded electronics that turn it into a potential IoT node.

IoT nodes are capable of receiving information and transmitting it through a communication interface. The IoT paradigm has had enormous potential, impacting both consumers' lives and business models due to the reduction in the cost of implementing these devices and the increase in demand. This trend is expected to grow rapidly (Fig. 1.1).

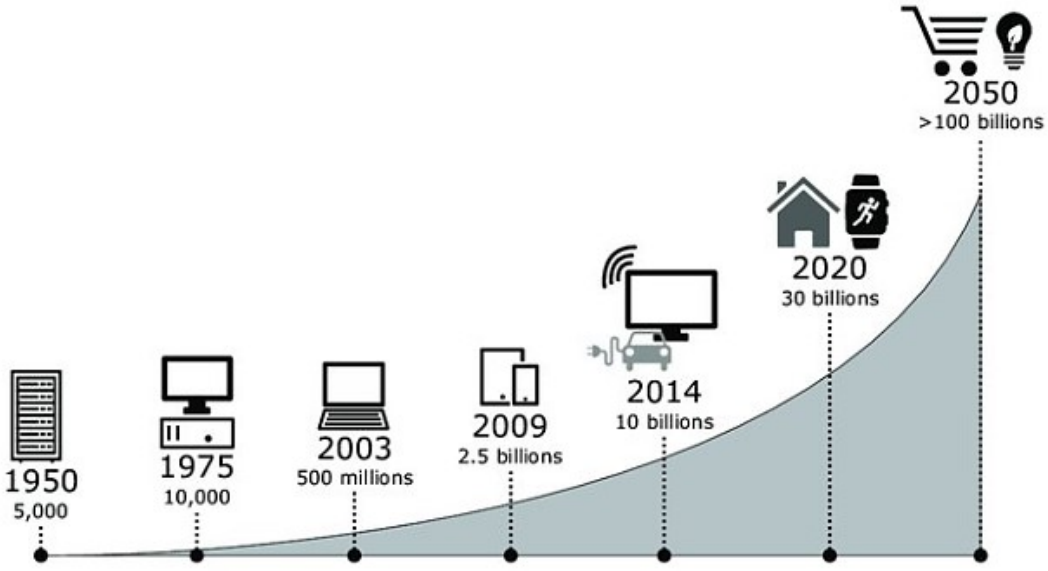


Fig. 1.1. Expected growth in the adoption of IoT devices

IoT nodes must perceive and collect data related to their specific task. There can be many tasks, ranging from a "smart" household appliance, sleep monitoring, tracking a person's physical activity, etc. Such information, collected from thousands of people, must guarantee absolute confidentiality for end users.

According to the edge computing paradigm, part of the workload is decentralized and distributed among IoT nodes, transforming them from simple sensors into more powerful and intelligent embedded systems. Thanks to this approach, the collected data can be further analyzed directly in the system itself, allowing for faster post-processing operations after data transmission. Many consider the Edge Computing paradigm to be a greener alternative to cloud computing due to its ability to limit the amount of data being transferred, and therefore reduce energy consumption. Typically, the data that needs to be stored in memory comes from peripheral devices, which are represented by sensors, transceiver modules for communication, or connectors. The steps performed in the embedded program can be represented by receiving data from sensors, processing it through a microcontroller , and transmitting the results using wireless communication (Bluetooth, cellular network, Wi-Fi, Zigbee, Z-Wave, LoRaWAN, special data transmission protocols, etc.).

MCU (Microcontroller Unit) is a compact integrated device (microcircuit) that integrates a central processing unit (CPU), memory, and peripherals to control electronic systems. Such devices are commonly used for embedded systems such as household appliances, automotive systems, medical devices, robots, industrial equipment, and other applications where control, data processing, or management of various functions is required. MCUs are small in size, low in cost, and low in power consumption, making them an ideal choice for many embedded systems (Fig. 1.2).

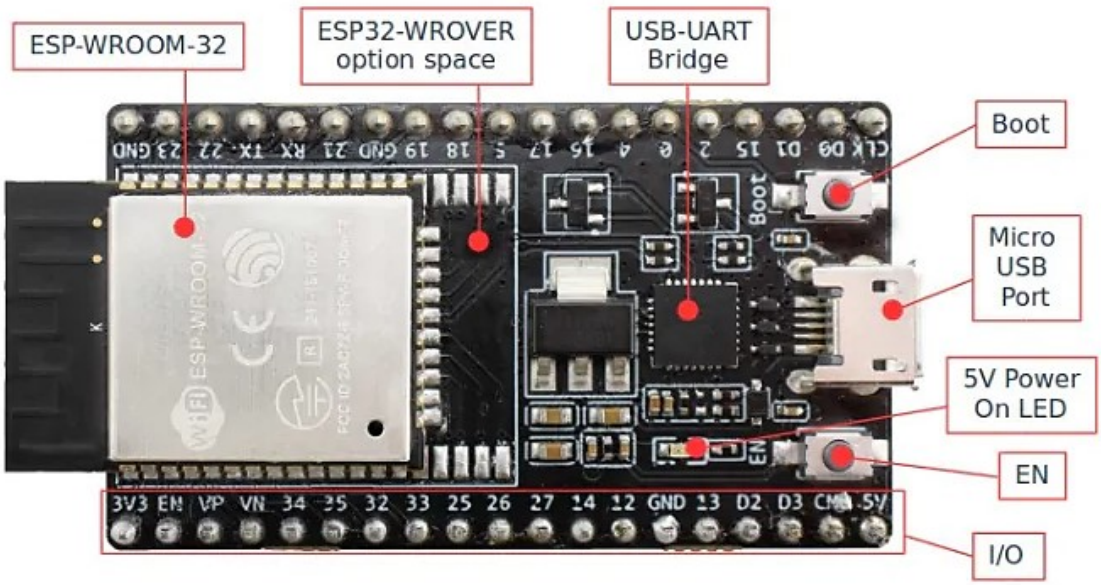


Fig. 1.2. ESP32 microcontroller from Espressif Systems

The ESP32 series from Espressif Systems combines Wi-Fi and Bluetooth Low Energy (BLE) in a single SoC (System on a Chip), which is widely used in IoT applications due to its energy efficiency and versatility. During the wake-up phase, which is triggered by an external event, the MCU state is restored along with the power supply and clock signal. After completing the task, the MCU stores its current state before reentering the low-power state. This behavior is predictable, so the host system can wake up and put to sleep different participants, including itself, by resorting to well-defined power management.

ESP32 from Espressif Systems is a series of microcontrollers, the main characteristics of which are the integration of wireless interfaces, the possibility of energy-efficient operation, as well as a large number of peripheral modules. ESP32 is equipped with a dual-core Xtensa LX6 processor from Tensilica, which can operate at clock speeds of up to 240 MHz. There are also versions with a single-core processor to reduce power consumption. The processor has a 32-bit architecture with support for both the main core for calculations and an additional core for background tasks.

The ESP32 has 520 KB of built-in SRAM and support for external RAM (PSRAM) up to 8 MB, allowing it to handle large computational tasks or data. The ESP32 has built-in flash memory, typically 4 MB to 16 MB, which is used to store firmware and user data.

ESP32 supports 802.11 b/g/n wireless standards at 2.4 GHz. It can operate in both access point (AP) and client (STA) modes, and supports both Bluetooth 4.2 and Bluetooth Low Energy (BLE), making it suitable for energy-efficient solutions in IoT devices.

The ESP32 has the following peripheral interfaces:

- GPIO (General-Purpose Input/Output), containing up to 34 multi-function pins for working with external components that support protocols such as SPI, I2C, I2S, UART;
- built-in analog-to-digital converters (ADC, Analog-to-Digital Converter) and digital-to-analog converters (DAC, Digital-to-analog converter) for measuring and outputting analog signals;
- the microcontroller supports Pulse-Width Modulation (PWM), which allows users to adjust power or control motors;
- Built-in cryptographic accelerators for secure data exchange over networks, AES, SHA and RSA hardware encryption is supported. There is

also a Secure Boot function that prevents unauthorized software from running.

Thanks to its built-in wireless modules, the ESP32 can perform several tasks simultaneously - for example, manage Wi-Fi network connection and exchange data via Bluetooth, which makes this microcontroller versatile for creating IoT devices that require the simultaneous use of several types of communication.

The ESP32 supports several low power modes:

- Deep Sleep – consuming only a few microamps, allowing the microcontroller to remain in a standby state with minimal power usage;
- Light Sleep – used to perform periodic tasks, while keeping power consumption low.

These modes make the ESP32 ideal for battery-powered solutions or devices that need to run for long periods of time without recharging.

The ESP32 can perform multiple tasks simultaneously thanks to FreeRTOS, a real-time operating system integrated into the firmware. This enables the creation of more complex applications with clear task management.

ESP32 can be programmed in several languages:

- C/C++ via Arduino IDE – widely used due to its simplicity and the availability of a large number of libraries.
- MicroPython is an interpreted programming language that allows users to quickly prototype and execute scripts without the need for compilation.
- ESP-IDF (Espressif IoT Development Framework) is an official SDK that allows users to use all the capabilities of ESP32 for developing high-performance applications.

With integrated Wi-Fi and Bluetooth, the ESP32 is an ideal choice for building “smart” devices such as sensors, “smart” lights, and access control systems. Support for a wide range of interfaces allows users to use the ESP32 for home or industrial automation. Thanks to its energy-efficient modes, the ESP32 is used in projects that require long battery life. With support for the I2S protocol, the microcontroller can process audio and be used to create audio devices or stream audio.

The advantages of the ESP32 are high performance with low power consumption, support for Wi-Fi and Bluetooth on a single chip,

multifunctional peripheral interfaces, flexibility in programming, and support for various languages.

The ESP32 is one of the most popular microcontrollers due to its capabilities and accessibility for a wide range of developers.

Along with hardware systems, data processing involves the CPU, as it must retrieve data from memory or sensors, process it (usually using a special program algorithm), and store it in memory.

From a data processing perspective, energy savings can be achieved by increasing parallelism, namely the amount of data per cycle that the CPU can process, or by optimizing the architecture for power consumption per cycle. Microcontrollers have evolved from the original 8-bit to the 32- bit modern devices.

The number of things connected to the Internet of Things is growing every year. From smartphones and household appliances to industrial sensors and transportation systems, more and more devices are becoming part of the IoT network and are used in medicine, the automotive industry, agriculture, logistics, urban planning and many other industries.

Processors are becoming more powerful and algorithms are becoming more sophisticated. This enables IoT devices to analyze and interpret data at the point of collection, ensuring quicker and more efficient responses and actions. As the number of connected devices increases, so does the risk of cyberattacks and data breaches.

The application of artificial intelligence helps IoT devices analyze data more effectively, identify patterns, and reach new levels of automation and optimization.

Artificial intelligence (AI) is a branch of computer science that aims to create systems that can independently perform tasks that usually require human intelligence. The development of common standards and protocols ensures compatibility and interoperability between different devices and systems, which makes the implementation of IoT more efficient.

The application of IoT in geospatial information systems helps in the collection and analysis of location data, which is useful for urban planning, transportation and other industries. The development of more efficient and long-lasting power sources allows IoT devices to operate continuously and for longer. The development of IoT continues to grow dynamically and opens up many opportunities for technological innovations in various areas.

The IoT market demonstrates resilience, despite the general economic downturn. For example, in 2022, spending by US enterprises on IoT increased by 21.5% to $201 billion. The IoT market in Ukraine is also actively developing, there is a growing interest in the implementation of such technologies in industry, agriculture, medicine and urban development.

In Ukraine, IoT solutions are being developed and used to improve production processes, increase productivity, reduce costs, and there is an increase in investments in startups specializing in IoT developments.

In agriculture, IoT solutions are being implemented for comprehensive control, crop management, livestock tracking and efficient use of resources, which increases productivity and optimizes costs (Fig. 1.3).

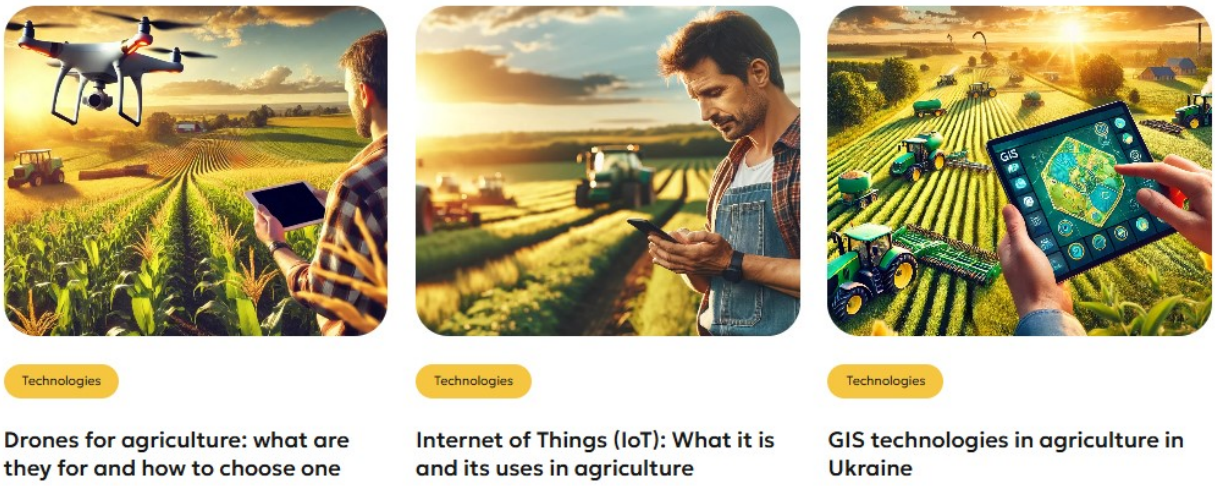


Fig. 1.3. Site publications https://weagro.com.ua

In urban planning, the concept of "Smart City" is gaining popularity, focusing efforts on integrating modern technologies to improve the management of urban resources and infrastructure. This includes the implementation of intelligent systems for managing transport, energy consumption and utilities, which improves the quality of life of city residents.

Investments in Ukrainian IoT startups are also growing. For example, in 2016, the “smart” dumbbell project Helko received $40,000 from IoT Hub Kiev and $3,000 from the coworking space “Platforma” for an 11% stake in the startup. Ukrainian startups attracted $209 million in venture capital in 2023, which indicates a slowdown in the decline in investment and signs of market recovery. In 2024, the Ukrainian startup Vidar Systems, specializing in the development of IoT solutions, attracted $450,000 in investment from

the Czech venture fund Presto Ventures and a group of investors, which indicates growing interest in Ukrainian IoT developments.

1.3. Classification of big data

Big data can be classified in various aspects according to their characteristics, processing technologies and usage. Consider the main classification categories of big data. By volume (Volume), data can be divided into small and large.

Small data is a relatively small amount of data that can be efficiently processed by traditional methods, typically Small Data can encompass data that is on the scale of 1 to tens of gigabytes (1 gigabyte (GB) is 1024 megabytes (MB) or approximately 1 billion bytes, this is the amount of memory used to store data such as text, images, video, audio, etc.). For example, it can be data from a separate dataset, text files, data from measuring instruments that can be processed on local computers or servers without involving large infrastructure.

Big data is large volumes of data that exceed the capabilities of traditional methods and require specialized approaches. Typically, Big Data encompasses volumes of data measured in petabytes (PB) , exabytes (EB) and larger. For example, big data can include genome sequence arrays, IoT sensor data, video surveillance data, and other voluminous datasets.

1 petabyte (PB) is 1024 terabytes (TB) or 1 quadrillion (10^15) bytes. This volume can include large data sets, large databases, video archives, and other objects with a large amount of information.

1 exabyte (EB) is 1024 petabytes (PB) or 1 quintillion (10^18) bytes. An exabyte can include enormous amounts of information, such as long-term data archives, large scientific studies, global data sets, and other large storage resources.

Private big data is data that belongs to specific organizations or individuals and is stored in closed systems or databases. It may contain confidential or sensitive information, such as personal data, financial data, or commercial information.

Public big data is data that is available for public use and distribution. It can be collected from various sources such as government organizations, research institutions, social networks, etc., and can contain a variety of information reflecting social, economic, and other aspects of society.

Differences between private and public data consist of the level of availability and confidentiality. Private data is restricted in access and has restrictions on use, while public data is available to a wide range of users and can be used for various purposes without restrictions. Private data is associated with an expectation of confidentiality and is regulated by a particular country/government (Fig. 1.4).

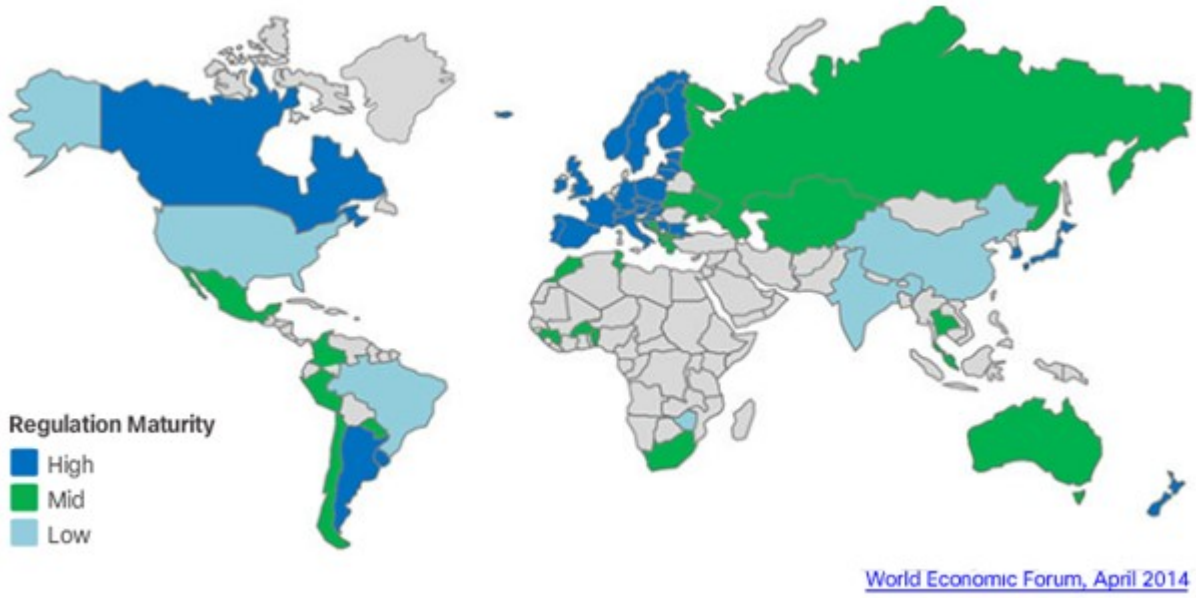


Fig. 1.4. Level of legislative regulation regarding the use private data in different countries

The legal regulation of private data varies between countries, depending on the approach to protecting the privacy of citizens and businesses. Consider examples of the regulation of the use of private data in several countries. There is no single, comprehensive privacy law in the United States. Instead, there are several federal and state laws that govern specific industries or aspects.

- Health Insurance Portability and Accountability Act (HIPAA) – regulates the protection of medical data.
- Children's Online Privacy Protection Act (COPPA) – protects the privacy of children under 13 years of age on the Internet.
- California Consumer Privacy Act (CCPA) is one of the most modern and stringent laws at the state level, providing California residents with the right to control how their data is collected, stored, and used by companies.

- Gramm-Leach-Bliley Act (GLBA) – applies to financial institutions and requires them to protect the privacy of customer data.

In the European Union, the General Data Protection Regulation (GDPR) law came into effect in 2018, the main requirements of which are as follows.

- Citizens' rights to access, rectify, delete and restrict the processing of their data.
- Strict rules for collecting consent for data processing.
- High fines for violations (up to 20 million euros or 4% of the company's annual turnover).
- Mandatory notification of data breaches within 72 hours.

The GDPR is a model for privacy legislation in many countries outside the EU.

The legislation of Ukraine on the protection of personal data is regulated by the following acts.

1. Law of Ukraine "On Personal Data Protection" (2010) is the main law, which stipulates that personal data may be processed only with the consent of the data subject or on the grounds specified by law. It imposes an obligation on organizations to notify about the processing of personal data. It requires ensuring confidentiality and data protection. In particular, organizations must obtain the explicit consent of citizens to process their personal data.
2. National Commission for Personal Data Protection is a state body responsible for monitoring compliance with data protection legislation.

Ukraine is moving towards harmonizing its legislation with European standards, in particular with GDPR, as part of European integration processes.

In Canada, the Personal Information Protection and Electronic Documents Act (PIPEDA) applies to commercial activities and ensures citizens' rights to protect their personal data.

In Brazil, the Lei Geral de Proteção de Dados (LGPD) is an analogue of the GDPR law, which came into force in 2020 and sets strict rules for the collection, storage and use of private data.

Each country or region may have its own specific legislation, but the common trend is to strengthen control over the use of personal data and increase the responsibility of organizations for protecting the privacy of citizens' data.

According to the speed (Velocity) of processing big data, batch processing and real-time data processing methods are distinguished.

Batch processing is the processing of data in batches, usually during periods of low system load. It is a method of processing data or tasks in which a group of input data or tasks is processed simultaneously, in a batch or batch. The data is processed in predetermined batches, rather than one at a time.

Batch processing has its advantages when we need to process large amounts of data or perform tasks that can be grouped together to optimize resources and time. For example, when processing large amounts of data, such as financial transactions, server logs, or research data, it is important to group them into batches for efficient processing and analysis. In graphics applications, batch processing is often used to apply the same filters or changes to a group of photos at the same time. Batch processing can also be used to execute a batch of SQL queries to a database at the same time, such as inserting, updating, or deleting records.

In financial systems, batch processing can be used to process a group of financial transactions or payments simultaneously. In natural language processing or text processing, batch processing can be used to analyze texts in large data corpora.

Batch processing allows for efficient use of resources such as computing power and time, as multiple tasks are processed simultaneously. This approach can also lead to delays if one task in the batch takes longer than the others.

Real-time processing is the processing of data immediately as it arrives, requiring high performance and speed. For example, real-time processing of financial transactions allows instant tracking and analysis of cash flows. In medical systems, real-time data processing allows users to monitor the patient's condition, analyze indicators and respond to changes in a timely manner. Real-time data processing in telecommunications systems allows users to instantly transmit, process and analyze data from a large number of users. Real-time processing of data from sensors and IoT devices allows users to track the status of objects and interact with them immediately. Processing data from various sensors in production processes allows users to immediately respond to problems and optimize equipment performance. Real-time processing technology requires high-speed hardware and software, as well as algorithms that can perform operations quickly and efficiently.

Compared to batch processing, where data is typically grouped and processed together, real-time processing occurs individually for each piece of data or event, allowing for instant response to changes and events.

By source, big data is divided into social media data – data from social platforms such as Twitter, Facebook, Instagram, etc.; sensor data – data from various sensors such as GPS, accelerometers, thermometers, etc.; data from IoT devices (IoT Device Data) – data from connected IoT devices.

By value, there are "golden data" and data of little value.

"Golden data" is data that has great value for the organization and helps make strategic decisions.

Low-value data is data that has less strategic significance and is used less.

1.4. Structured and unstructured data

By variety, structured, unstructured, and semi-structured data are distinguished (Fig. 1.5). Structured and unstructured data are the two main categories of data used to describe information. They differ in the way they are organized and the format in which they are stored.

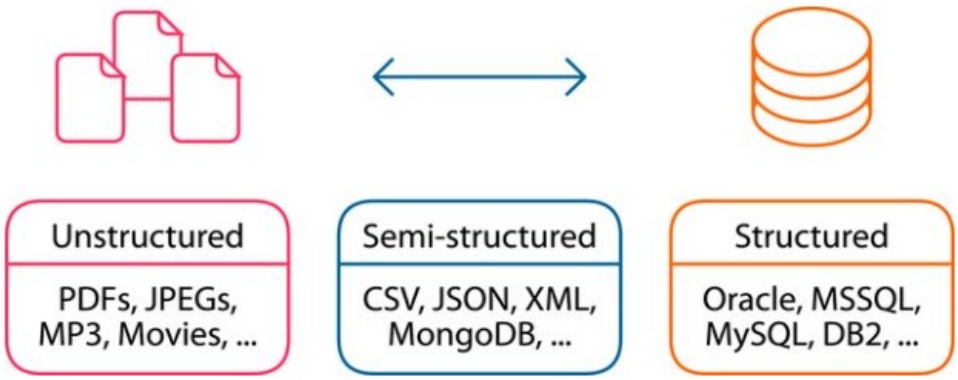


Fig. 1.5. Examples of structured, unstructured and semi-structured data

Structured data is data that is organized into a clear, systematic structure that is identified and interpreted using standard rules. It is typically stored in relational databases and tables, where each row and column has clear semantics. Let's look at examples of structured data.

Example 1. The diagram above illustrates a structured database schema designed for an experimental study. The schema consists of multiple

interconnected tables, each with a specific purpose and clearly defined relationships (Fig. 1.6).

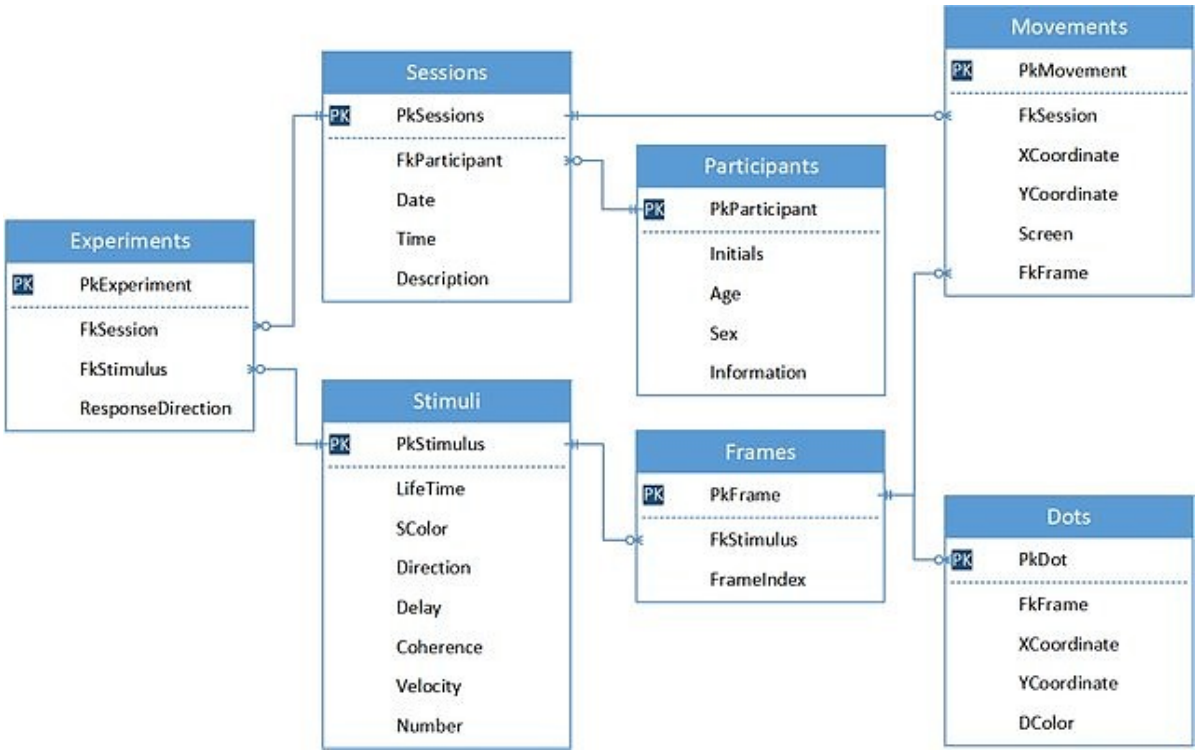


Fig. 1.6. Structured data of a relational database example

Experiments table contains details about individual experiments, linking to *Sessions* table through *FkSession* and *Stimuli* table through *FkStimulus*. *Sessions* table records information about experimental sessions, including the participant (*FkParticipant* linked to *Participants* table), date, time, and a description of the session. *Participants* table stores demographic and identifying information about the individuals involved, such as their initials, age, sex, and additional details. *Stimuli* table captures information about the experimental stimuli, including properties such as lifetime, color, direction, delay, coherence, velocity, and number. *Stimuli* are further broken down into frames, represented in the *Frames* table, which stores individual frame data and links to the stimuli they belong to through *FkStimulus*.

Movements table tracks movement data within sessions, recording coordinates and screen data while connecting to both *Sessions* and *Frames* tables via foreign keys. *Dots* table represents individual elements within frames, storing details such as their coordinates and color, and linking to *Frames* table. This structured approach to data organization enables

systematic storage, retrieval, and analysis of complex relationships between participants, sessions, stimuli, movements, and visual elements, showcasing the power and clarity of relational databases in handling structured data.

Example 2. A bank transaction journal, where transactions are stored as records, each with fields for date, amount, transaction type, etc.

Example 3. A spreadsheet with financial data that contains columns for dates, income, expenses, and balance.

Unstructured data is data without a clear structure, such as text files, images, videos, audio, etc. It is difficult or even impossible to interpret using standard processing methods, as it can contain text, graphics, images, videos, etc. Let's look at examples of unstructured data.

Example 1. Text documents, such as articles, emails. Text can contain information of various types – from simple text to complex analytical materials.

Example 2. Images and photographs, such as camera shots, pictures, and graphics. They can contain a large amount of information, but this information cannot always be automatically recognized.

Example 3. Audio and video recordings (music, videos, podcasts, etc.). The main information is the sound or visual image, but their interpretation requires specialized analysis.

Example 4. Comments, posts, messages from social networks (Facebook, Instagram, Twitter, LinkedIn) – this data may contain text, photos, links, and other embedded objects.

Example 5. Scanned documents, paper documents that have been converted to digital form, such as handwritten notes or official letters.

In today's world, data can be diverse, combining structured and unstructured elements.

Semi-structured data is data that has a partial structure, such as JSON or XML files. The characteristics of semi-structured data include the ability to accommodate disparate types of information in a single set, variability in the format and structure of the data, and the ability to add new fields or changes to existing data without having to change the entire database. This makes semi-structured data useful in situations where flexibility in storing and processing information is required, such as in weblog analysis or processing data from different sources.

Semi-structured data in weblog analysis may include information about

user queries, visit times, IP addresses, query types, and other parameters that allow us to understand user behavior on the website and optimize its functioning.

Relational databases meet the needs of analytics in most applications. With the growth of information volumes, such as web transactions and machine-generated data, large-scale analytics are becoming increasingly difficult to manage with just a Relational Database Management System (RDBMS). Relational databases typically have three components: database management software, the physical servers on which that software is installed, and the disks on which the data is stored. As databases grow, more powerful servers are needed to process and store larger amounts of data. With traditional RDBMS infrastructure and the amount of data that new applications consume, it is easy to reach a point where even a very powerful server and large amount of storage cannot handle the processing.

An alternative solution is distributed processing in the cloud, which breaks up large amounts of data into smaller pieces. These smaller pieces of data are distributed across multiple locations and processed by multiple computers. Each computer in this distributed architecture analyzes only its portion of the data, spreading the workload across multiple machines.

Some relational databases allow this type of distribution, but sometimes a different database is needed to maximize the efficiency of these systems and/or to manage flexible and unstructured data.

NoSQL is a database that stores and retrieves data differently than relational databases. NoSQL is often called "non-relational" because it does not organize data in tables with rows and columns that follow a specific schema. Instead, data is stored in an unstructured or semi-structured form, which simplifies database design.

There are four main types of NoSQL databases. Let's look at one example of each type to understand how they differ from relational databases.

Type 1. Document databases store documents in machine-readable formats such as JSON, BSON, or XML.

Although these databases are less efficient than relational databases at managing relationships between documents, they are much better suited for reading and searching documents. In addition, the structure of one document does not necessarily have to match the structure of another document in the database. This allows new elements to be added to documents without having

to change tables or schemas (Fig. 1.7). The diagram illustrates a collection of documents within a NoSQL database, specifically representing user data. Each document contains an id field for unique identification and an *isActive* field indicating the user's activity status. The structure of the *fullName* field varies across documents, demonstrating the flexibility of schema design in NoSQL databases. In Document 1, the user is identified with a name field storing a single string. Document 2 uses a *fullName* field, also represented as a single string. In contrast, Document 3 organizes the *fullName* field as a nested object with first and last attributes, representing the user's first and last names separately. This variability highlights the schema-less nature of NoSQL databases, allowing documents to have different structures while still residing in the same collection.

Fig. 1.7. Example of a fragment of a document database

Type 2. Graph databases store "nodes" that are connected to other "nodes" in a network. A node might be, for example, a user in a social network who has connections to all of their friends.

Another example is points on a map that are connected in real life, making it easy to find routes between them.

Graph databases are optimized for fast query execution and navigation through these relationships much more efficiently than relational databases, which connect tables to find relationships. They are specifically designed to store large volumes of interconnected nodes.

In Fig. 1.8 shows a fragment of a graph database about movies, showing movie nodes (purple), person nodes (orange), and the connections between them (arrows) reflecting their role in the movie (e.g., "ACTED_IN", "DIRECTED", "PRODUCED") .

This structure allows for modeling the complex relationships that may exist between many entities, making it ideal for working with linked data, such as the film industry. The advantage is that it allows for quick access to relationships between objects and allows for the expansion of the model by adding new nodes or relationships, which is difficult with traditional relational databases.

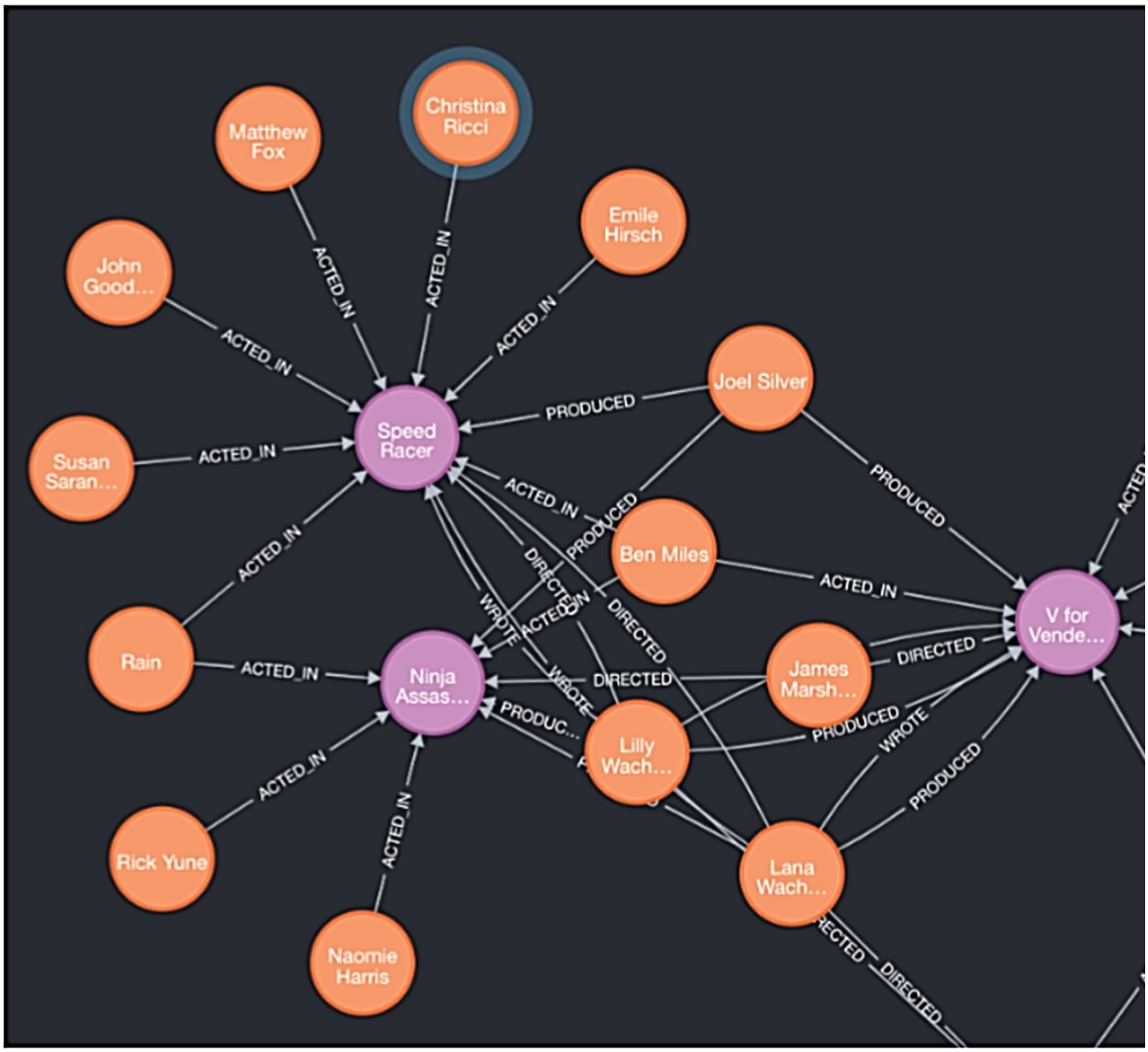


Fig. 1.8. Example of a fragment of a graph database about movies

Type 3. Key-value databases store data in key-value pairs (key-value database or key-value store). The key acts as a unique identifier that allows users to quickly find the corresponding value (Fig. 1.9) . This type of database is characterized by simplicity and high speed of data access, as they do not require complex search operations.

Key -value databases are suitable for situations where we need to store large amounts of simple data, such as configuration settings or user session data. They are less suitable for handling complex queries or when the data has complex relationships.

Key			
8923bc87e4f6a234	Name: Diego	Age: 46	SSN: 000-00-0000
d2135cab54345dd3	Color: Red		
4234fcca33e4bb32c	Drink: Orange Juice	Menu: Combo 6	

Fig. 1.9. Example of a key-value database fragment

Type 4. Wide column stores are a type of database that uses columns as the primary structure for storing and processing data. Unlike traditional relational databases, where data is organized in tables with a fixed number of columns, wide column stores work with tables where each row can have its own unique set of columns. This makes this type of database flexible for storing large amounts of data that may have different structures. Each row in such a database can be organized as a family of columns, where data is grouped into logical blocks (Fig. 1.10).

	Movie_1	Movie_2	Movie_3	Movie_4
User_1	10/22/2020			
User_2			08/29/2022	
User_3				
User_4		09/02/2019		11/13/2021

Fig. 1.10. Example of a Wide Column Stores database fragment

Wide column stores databases are optimized for distributed systems such as clusters, allowing them to efficiently handle large amounts of data. They are also well suited for high-volume systems because they can scale horizontally by adding new nodes to the cluster without significant performance degradation. Examples of such databases include Apache

Cassandra and HBase, which are widely used in modern storage systems for rapidly changing or irregularly structured data.

In most cases, we can imagine ways to perform the same operations that NoSQL databases perform using a relational database, but in many situations, a relational database will not be as efficient as the solutions considered. This efficiency comes at a “price”. Relational databases are known for their reliability, correctness, and version control, while NoSQL databases can leave more room for errors and access to data that is not up-to-date. This is an important factor to consider when choosing a NoSQL solution. NoSQL is often a good choice when working with large amounts of data that change frequently, or when the data has flexible formats that do not fit the relational database model.

Consider the advantages of NoSQL databases compared to relational ones.

- NoSQL databases are designed for large, unstructured data sets.
- The ability to add new data whose structure differs from existing data, which is not available for relational databases or files with a flat structure.
- Rapid scaling to support data growth.

Consider some of the problems of using NoSQL databases.

- There is no way to check fields for compliance with existing data the way SQL databases do.
- Discrepancies are possible due to access to different versions of data.
- Less support for NoSQL applications.
- Lack of standards for queries in NoSQL databases.

1.5. Cloud computing

Cloud computing is a model for providing access to computing resources, such as computing power, storage, and software, over the Internet. Instead of having to equip and manage their own physical servers and infrastructure, users can rent these resources from providers. cloud services.

The diagram in Fig. 1.11 illustrates a cloud computing ecosystem that enables the integration of various technologies and services.

The center depicts "Cloud Computing", which includes components

such as machine learning and artificial intelligence, databases and data centers, networks, containers, computing, virtual machines, data storage and security. Arrows indicate the relationships between these components, which allow for the implementation of comprehensive solutions for storing, processing data and performing calculations in a cloud environment. This approach provides scalability, flexibility and ease of use of resources for various areas of activity.

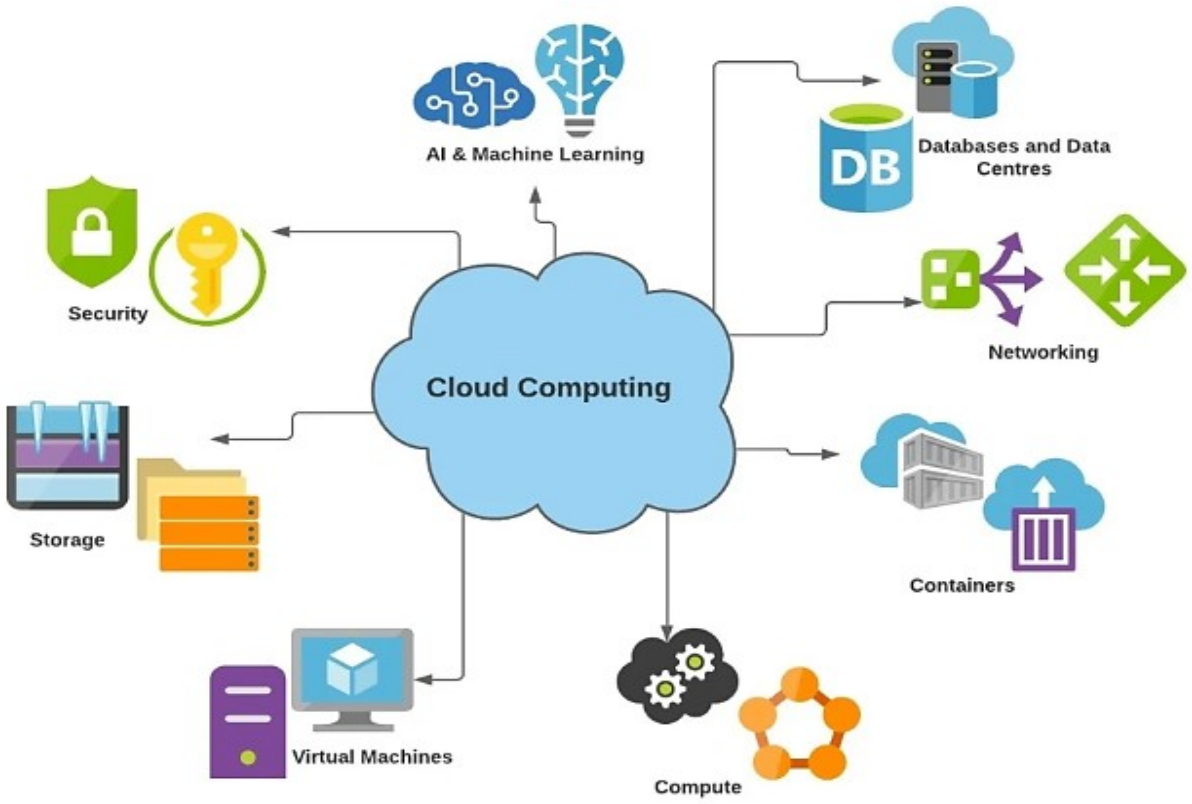


Fig. 1.11. Cloud computing

Many companies use cloud services for data storage, computing, email, and document collaboration. IoT devices can transmit and process data to cloud services for analysis and management. The main advantage of using cloud services is their flexibility and scalability, which allow companies to quickly adapt to changing workloads and resource needs without the need for large investments in physical infrastructure.

Cloud platforms provide access to cutting-edge technologies and innovations, such as machine learning and artificial intelligence, with the ability to pay only for the resources actually used. This allows companies to

optimize their costs, increase efficiency, and improve their competitiveness in the market.

Users can rent virtual servers in the cloud, which allows them to host and manage websites and applications. Developers can use cloud services to build, test, and deploy applications , and to create and configure infrastructure (servers, networks) using code (Table 1.1).

Table 1.1. **Basic characteristics of cloud computing**

Characteristic	Description
Flexibility	The ability to scale computing resources depending on user needs.
Accessibility	Computing resources are available at any time and from any location via the Internet.
Sharing	Multiple users can use and share resources.
Pay-per-use	Users pay only for the resources used, which allows for more efficient use of funds.
Scalability	Cloud resources can be scaled to meet growing needs.

Infrastructure as code (IaC) is an approach to automating IT infrastructure management in which infrastructure resources (such as virtual machines, network resources, storage, etc.) are configured and managed using code, typically in the form of scripts or configuration files. This allows for the automation of infrastructure deployment and management processes, providing greater speed, reliability, and consistency, as well as facilitating scaling and development.

Google Colab (Google Colaboratory) is a free cloud computing service that allows users to run and explore Python code in a web browser. Using Google Colab , software developers can create and execute Jupyter notebooks in the Python programming language (files with the extension *.ipynb*), which allows users to describe code, add explanatory comments, and visualizations.

Google Colab comes pre-installed with most popular scientific computing libraries, including TensorFlow, PyTorch, NumPy, Pandas, Matplotlib, and more. Developers can store their notebooks on Google Drive and share them with other users.

Google Colab provides a virtual machine with installed libraries for computing, training deep neural networks (Deep Learning), data analysis, and many other tasks (Fig. 1.12). The Google Colab service was launched by Google in the spring of 2017.

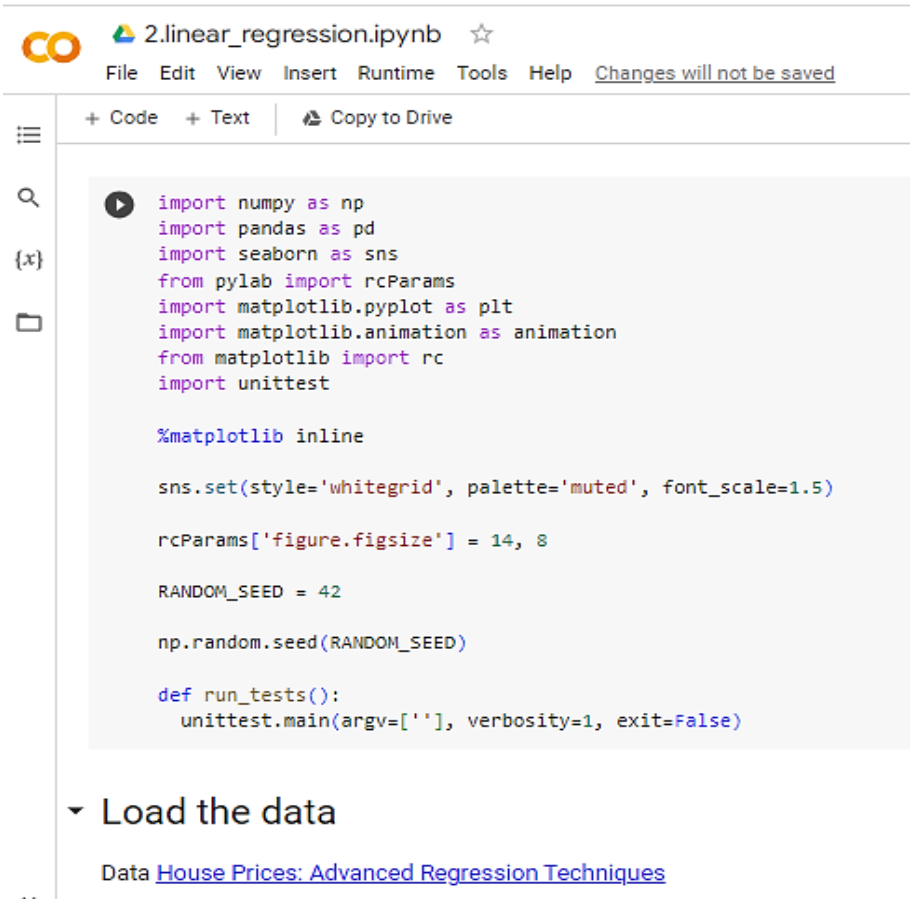


Fig. 1.12. Example of using Google Colab for regression data analysis (code snippet)

Google Colab is available for free use, it has limitations on computing resources and usage time. Colab computing capabilities may vary depending on the current load on Google servers and resource availability. A Google Colab session automatically ends after approximately 12 hours of inactivity or 90 minutes of active use. Google Colab provides limited access to resources such as the amount of random access memory (RAM), the number of CPU cores, and available compute time. It is important to be aware of the file size limits that can be uploaded to Colab. The limits on the size of files that can be efficiently uploaded and processed in Colab depend on the available RAM and disk space.

Free sessions usually have limitations on RAM and storage space,

which can be around 12-17 GB for RAM and up to 100 GB for disk space (including all files saved in the current session). Because Colab sessions have a limited duration, large files may take longer to download or process, which may result in an unwanted session termination. The speed of internet connection can also significantly affect the time it takes to download large files. Despite these limitations, Google Colab remains a powerful and useful tool for developing and testing machine learning and big data applications.

Although Google Colab allows developers to perform many tasks, for large projects or tasks with high computational complexity, it may be desirable to use more powerful servers or cloud platforms that provide more resources (e.g. AWS, GCP, Microsoft Azure, IBM Cloud).

Amazon Web Services (AWS) is one of the most versatile cloud platforms, offering a wide range of services, including high-performance computing instances (e.g., EC2 instances), managed databases, data storage, machine learning, and more.

Google Cloud Platform (GCP) offers compute resources such as Compute Engine, managed Kubernetes clusters (GKE), machine learning services, and high-performance storage options. GCP is known for its networking infrastructure and integration with other Google products.

Microsoft Azure is a platform with a wide range of computing services, including virtual machines, Kubernetes services, AI and data analytics. Azure offers flexibility in choosing operating systems and integration with Microsoft tools.

IBM Cloud provides compute resources, including virtual servers and containers, as well as specialized compute options for AI and machine learning through Watson. The IBM Cloud platform also focuses on private and hybrid cloud solutions.

Google Colab provides access to GPUs for computational acceleration, as well as an application-specific integrated circuit (ASIC) designed to accelerate AI computations, enabling rapid neural network training. Traditionally, machine learning computing has used central processing units (CPUs) and graphics processing units (GPUs).

GPU (Graphics Processing Unit) is a specialized processor optimized for graphics and visualization processing, widely used in calculations that require a large number of parallel operations. Known as "graphics cards" or "video cards" in the context of personal computers, GPUs can significantly

accelerate image and video processing, as well as perform complex mathematical calculations for scientific research, machine learning, deep learning, and other specialized applications.

CPUs are versatile and can perform a variety of tasks, but they are not always suitable for specialized machine learning computations due to their architecture. GPUs can process large amounts of data in parallel, making them more efficient for machine learning tasks, but they are not exclusively specialized for this.

Unlike traditional CPUs, which are optimized for sequential processing, GPUs contain hundreds or even thousands of small cores capable of executing thousands of threads simultaneously. This is used for tasks that can be parallelized into a large number of independent computations. Due to their parallel processing capabilities, GPUs can offer significantly higher computing power compared to CPUs when large data sets need to be processed, such as when training machine learning models. In addition to graphics computing, GPUs are also effectively used in scientific research, engineering simulations, financial modeling, big data analysis, and many other areas.

Tensor Processing Units (TPU) are specialized hardware devices (computing chips) developed by Google to accelerate computations related to machine learning and artificial intelligence. They are part of the Google Cloud infrastructure and are designed to optimize the performance and efficiency of computations related to training and using neural networks. They offer significant acceleration of computational processes compared to traditional CPUs and GPUs. TPUs provide high computing speed with low power consumption, which is important for large data sets and complex models. These chips are optimized for computing with tensors – multidimensional data arrays, which are a key element in many machine learning algorithms.

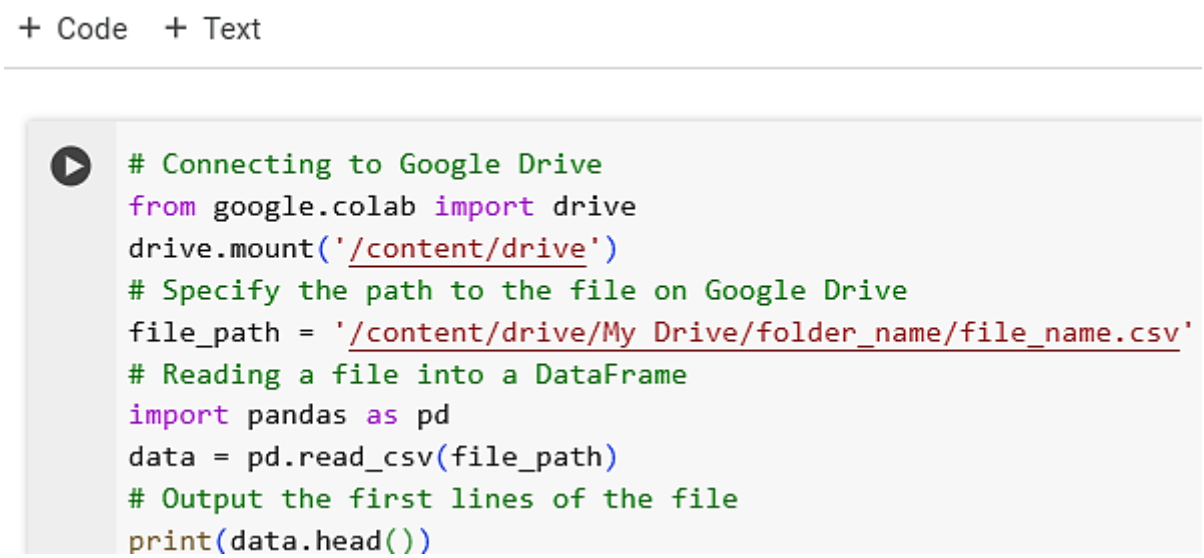
TensorFlow is an open-source machine learning and deep learning library developed by Google that provides a powerful framework for building, training, and deploying artificial neural networks. TensorFlow provides a user-friendly interface for creating and manipulating computational graphs, including optimization and automatic parallelization of operations. This allows developers to efficiently use computing resources and obtain optimal results. Compared to other libraries, TensorFlow is

distinguished by its broad support, active developer community, and set of tools for training and deploying machine learning models. TensorFlow has high performance and the ability to use specialized devices such as Tensor Processing Units (TPUs), making it one of the most efficient platforms for working with graphs and performing deep learning calculations.

Google offers TPU as part of its cloud infrastructure, allowing users to scale their resources based on project needs. TPUs are integrated with other Google Cloud services and tools, making it easier to develop and deploy machine learning models. Colab also integrates with other Google tools, such as Google Docs, allowing for collaborative editing of notebooks and work. Developers can use Colab for a variety of tasks, from simple calculations to complex scientific research.

To get started with Google Colab, we need to have own Google account. In Google Colab, we can create new notebooks, import existing notebooks, run code, visualize data, and perform various computational tasks. It is a popular tool among researchers, developers, and scientists because it allows developers to perform complex calculations without having to install and configure a local environment.

To read a file from Google Drive in Google Colab, we need to mount the Drive using `drive.mount('/content/drive')`, specify the path to the file (for example, `'/content/drive/My Drive/folder_name/file_name.csv'`), and read it into a DataFrame using `pd.read_csv(file_path)`:



This code will automatically generate a link for authorization. After the code is executed, the file will be read into a DataFrame for further analysis.

In Google Colab, developers can use the CPU of a virtual machine to perform calculations. Colab's computational capabilities may vary depending on the current load on Google's servers and resource availability. Although Colab allows developers to perform many tasks, for large projects or tasks with high computational complexity, it is sometimes desirable to use more powerful servers or cloud platforms that provide more resources. Google Colab is used for many types of tasks, especially for experimentation, prototyping, training machine learning models, and data analysis.

The amount of RAM available in Google Colab can vary depending on the type of virtual machine the developer is using and the resources available on the server at the time. Typically, the memory can range from 10 to 25 gigabytes. Cloud computing technologies are used to provide efficient processing of data and resources without the need for own physical infrastructure. Cloud computing allows users to access computing resources over the Internet without owning any hardware. Cloud services such as Google Drive, Dropbox, and iCloud allow to store files, photos, and documents online, making them accessible from any device connected to the network. Cloud email services Gmail, Yahoo Mail, and Outlook use cloud technologies to provide continuous access to email, contacts, and calendar from any device. Streaming services such as Netflix, Spotify, and YouTube use cloud technologies to deliver video and audio content, allowing users to instantly play movies and music without having to download them to their device. Platforms such as Facebook, Instagram, Twitter use cloud technologies to store and share images, videos, and other data between millions of users in real time. Smart devices (such as Amazon Alexa, Google Home, smart thermostats, and cameras) process data via the cloud, allowing users to remotely control devices and receive analytics.

Cloud software Microsoft Office 365, Google Workspace, and Adobe Creative Cloud allow users to access productivity and creativity apps, store documents in the cloud , and collaborate on projects. Cloud gaming services such as Google Stadia, Xbox Cloud Gaming, and NVIDIA GeForce Now allow users to play games without powerful hardware, as game processing takes place in the cloud. Cloud technologies provide secure access to banking services, allowing users to manage accounts, pay bills, and track financial

transactions through online banking and mobile apps.

Consider the main models of cloud computing.

IaaS (Infrastructure as a Service) – virtual servers, networks, and storage that users rent for computing.

PaaS (Platform as a Service) are platforms for developing and deploying applications where users focus on coding without having to worry about configuring the infrastructure.

SaaS (Software as a Service) are ready -made programs available via the Internet.

Oracle Cloud specializes in databases and enterprise applications, offering solutions for data management, analytics, computing, and the autonomous database (Oracle Autonomous Database).

Alibaba Cloud is a leading cloud service provider in Asia and offers a wide range of services for storage, data processing, AI, security, and analytics.

Salesforce specializes in cloud-based CRM solutions for customer relationship management, analytics, marketing automation, and customer service.

DigitalOcean provides simple and affordable cloud services aimed at developers looking for easy-to-use solutions for hosting applications, servers, and data storage.

Vultr is a cloud services provider focused on small and medium-sized businesses, offering virtual machines and computing resources at affordable prices, and also supports cloud storage.

These providers offer a variety of services that meet the needs of both individual developers and large corporations, providing solutions for data storage, information processing, analytics, security, and artificial intelligence support.

1.6. Fog computing

Fog computing is a computing architecture paradigm that involves processing data and performing computations closer to the data source than is typically done in cloud computing. Fog computing extends cloud capabilities by bringing them closer to end devices and IoT, allowing computation to be performed at the "fog" level, which is at the interface between the cloud and end devices (Fig. 1.13).

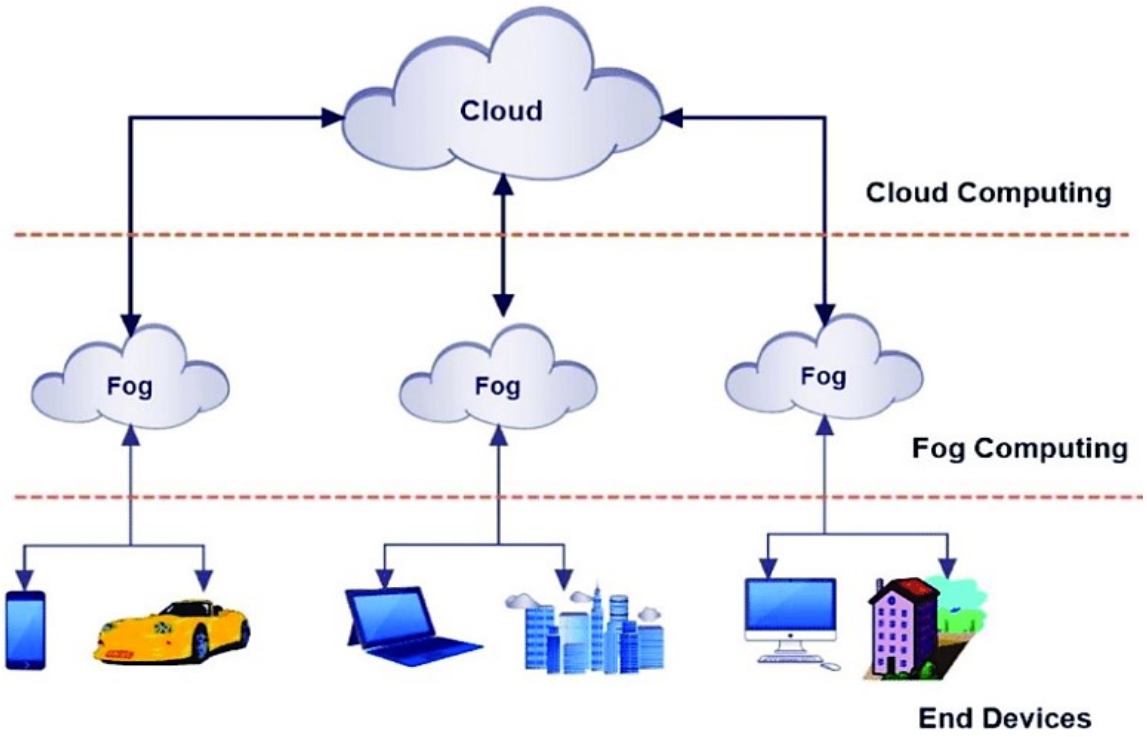


Fig. 1.13. Fog computing

Fog computing enables advanced data processing at the edge of the network, which reduces latency, improves bandwidth efficiency, and provides greater real-time reliability for IoT applications and other mission-critical systems. Consider examples of the use of fog computing.

Example 1. In urban management systems, fog computing can be used to process data from street light sensors, street surveillance cameras, and other IoT devices.

Example 2. Manufacturing companies can use fog computing to analyze data from sensors on the production site and take appropriate safety and optimization measures.

Example 3. In medical systems, fog computing can help track patient health indicators and provide reliable communication between medical devices.

Example 4. Real-time video data analysis in video surveillance systems, for example, to detect dangers or unusual events on city streets.

Fog computing extends the capabilities of cloud computing by enabling data processing to be performed closer to its sources, making it valuable for applications that require low latency, real-time analysis, and reliability (Table 1.2).

Table 1. 2. **Basic characteristics of fog computing**

Characteristic	Description
Locality	Fog computing involves performing calculations closer to the location of data collection, which helps reduce latency and network bandwidth consumption.
Reducing the load on the cloud	Computations can be performed at the end device or local node level, reducing the load on cloud servers.
Network bandwidth	Transferring only processed data over a network that uses bandwidth efficiently.
High availability and reliability	Fog computing can work even in the absence of a connection to the cloud, which ensures high availability and reliability of the system.
Real-time analysis	Performing calculations at the fog level allows for data analysis even in real time, which is important for many applications such as medical monitoring or manufacturing processes.

1.7. Supercomputers and quantum computing

Parallel computing is a data processing method in which a task is divided into multiple smaller parts that are executed simultaneously on multiple processors or cores. This approach significantly increases the speed of processing large amounts of data.

Distributed computing involves running tasks on multiple computers connected to a single network. Each machine processes its own part of the task, and the results are then combined to produce a final solution. This approach provides high performance and allows for scalability of computing resources. The main difference between these two approaches is that parallel computing focuses on the use of multi-core processors or computing clusters, while distributed computing involves resources located in different locations. This allows for global-scale tasks to be performed with minimal dependence on the physical location of the equipment. The idea of parallel computing arose to speed up the execution of complex computational tasks. In the 1950s, the first attempts to use multi-core processors began, and in the 1970s, supercomputers were actively developed. In the 1990s, distributed systems

appeared, which allowed combining the resources of several computers to solve common tasks.

Supercomputer is a term used to describe the most powerful computer systems available. They are designed to solve extremely complex scientific and engineering problems that require processing large amounts of data or performing millions of mathematical operations in a short period of time. It is important to note that the concept of a supercomputer is relative and depends on the development of technology at a particular point in time. The power of such systems is assessed in comparison with other widely available computer technologies.

Blue Gene supercomputer, located at Argonne National Laboratory, is one of the most advanced computing systems designed for large-scale scientific research. This high-performance machine was created to solve complex problems in various fields of science, such as modeling physical processes, research in biology, chemistry, and analysis of climate change. Blue Gene is distinguished by its architecture, which allows it to efficiently perform a huge number of calculations simultaneously thanks to a high level of parallelism. This system uses a large number of computing nodes that work together, providing fast data processing. This allows researchers to solve problems that require super-powerful computing resources, such as modeling the behavior of molecules, studying climate change, or creating new materials. Blue Gene supercomputer has become an important part of the scientific infrastructure of Argonne National Laboratory, providing scientists with a tool to perform computational experiments that would be impossible to perform on conventional computers (Fig. 1.14).

Fig. 1.14. Blue Gene supercomputer

Petaflop (PFLOPS) is a unit of computing performance that is equal to one quadrillion (10^15) floating-point operations per second. It is used to measure the performance of supercomputers, reflecting the number of calculations a system can perform per second. Systems with petaflop performance are widely used to solve complex problems in scientific research, climate modeling, big data processing, and artificial intelligence.

The Supercomputing Center of National Technical University of Ukraine "Igor Sikorsky Kyiv polytechnic institute" performs not only the internal tasks of the university, but also cooperates with scientific institutes of Ukraine and international organizations. The Center actively works with the Institute of Theoretical Physics, the Institute of Mathematics and the State Space Agency (Fig. 1.15).

Fig. 1.15. Supercomputing Center of National Technical University of Ukraine "Igor Sikorsky Kyiv polytechnic institute"

Supercomputer Center of the V.M. Hlushkov Institute of Cybernetics of the National Academy of Sciences of Ukraine is engaged in solving problems in the following areas:

- modeling of natural disasters;
- analysis of geophysical data for oil and gas exploration;
- weather forecasting and pollution effects;
- research in molecular dynamics, quantum chemistry, hydrodynamics, and aerodynamics;

- development of technologies in the fields of defense, medicine and agriculture.

Supercomputers are used for numerical modeling and processing of large amounts of data in real time.

The main areas of application include the following.

- Mathematics (cryptography, statistics).
- High-energy physics (study of nuclear processes, plasma, modeling of nuclear reactors).
- Earth science (weather forecasting, climate study, earthquake research, and geological exploration).
- Computational biology and chemistry (protein research, DNA, drug development).
- Physics and engineering (aerodynamic and hydrodynamic models, development of new materials, engineering design).

Supercomputers use software solutions based on MPI and PVM, as well as open systems such as Beowulf and openMosix, which allow the creation of virtual computing clusters. Parallel computing technologies are gradually being integrated into conventional computer systems, blurring the boundaries between supercomputers and personal computers.

World powers are actively investing in the development of supercomputer systems. Consider examples.

- Japan is funding the creation of a national supercomputer.
- IBM has released the Blue Waters and Sequoia supercomputers, which have a performance of up to 20 petaflops.
- China is building systems approaching exascale performance.
- Meta, Microsoft, and xAI are developing powerful computing systems based on GPUs and quantum technologies.

Current trends in the use of supercomputers include cloud and cluster computing for scaling performance; artificial intelligence and machine learning, which require powerful algorithms and computing resources; and quantum computing.

Quantum computing is a new technology in computer science and physics that uses the principles of quantum mechanics to create computing systems capable of solving problems beyond the capabilities of classical computers. In quantum computing, information is processed using quantum

bits (qubits), which use the phenomena of quantum superposition, entanglement, and interference.

Consider the basic principles of quantum computing.

Qubit (quantum bit) – unlike a classical bit, which can only take on the value 0 or 1, a qubit can simultaneously be in the state of 0, 1, or a superposition of them thanks to quantum mechanics. This allows for computing a large number of options simultaneously.

Formally, a qubit is described as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1.1}$$

where α and β are complex numbers that represent probability amplitudes. The square of their modulus determines the probability of obtaining a certain state during a measurement:

$$(|\alpha|^2 + |\beta|^2 = 1). \tag{1.2}$$

Qubits can be in a superposition of multiple states at the same time. This means that a quantum computer can perform many calculations in parallel.

Quantum entanglement is the phenomenon where two or more qubits become linked in such a way that the state of one qubit depends on the state of the other, regardless of the distance between them. This allows for ultrafast information transmission and optimization of computational algorithms.

Quantum interference is a process of interaction between quantum states that can enhance correct answers and suppress incorrect ones, increasing the efficiency of algorithms.

Consider the differences between quantum and classical computers.

1. *Computational elements*. In classical computers, the basic element is a bit (0 or 1), and in quantum computers, a qubit.

2. *Parallelism.* Thanks to superposition, a quantum computer can process many states simultaneously.

3. *Speed.* For certain tasks, quantum computers can perform calculations much faster than classical ones.

4. *Algorithms.* Quantum computers use special algorithms, such as Shor's algorithm (for factoring numbers) or Grover's algorithm (for fast database searches).

Consider key quantum algorithms.

Shor's algorithm developed by Peter Shore, this algorithm enables the factoring of large numbers into prime factors in polynomial time, which is

critical for cryptography, as modern encryption systems such as RSA are based on the complexity of this problem.

Grover's algorithm allows searching an unordered database in quadratic time compared to classical search. For a classical computer, the search time is proportional to N, while for a quantum computer, it is $\sqrt{N}$.

Quantum computers can effectively model complex physical systems containing large numbers of quantum particles. This is extremely important for chemistry, materials science, and biology.

Quantum Monte Carlo calculations allow to speed up statistical calculations by using quantum principles to reduce the variance of estimates.

Thanks to Shor's algorithm, quantum computing can break current cryptographic systems. Instead, post-quantum cryptography is being researched to be resistant to such attacks.

Quantum computers allow for precise modeling of the behavior of molecules and chemical reactions, which helps create new medicines and materials. Quantum computing can speed up neural network training and other machine learning tasks. Quantum simulations allow us to explore complex phenomena in quantum physics that cannot be modeled on classical computers. Qubits are very sensitive to external influences, which leads to the loss of quantum states. To combat this, quantum error correction is used. Quantum computers require ultra-low temperatures (near absolute zero) to ensure the stability of qubits.

Building a quantum computer requires the creation of ultra-precise physical systems to control qubits. Current quantum computers still have a limited number of qubits (tens or hundreds), which limits their power. Due to the quantum nature of qubits, it is very difficult to maintain their state for a long time without errors.

Although quantum computers are still in their infancy, they have the potential to revolutionize computing. Tech giants such as Google, IBM, and Microsoft are actively working to create robust quantum systems.

Google Quantum AI Lab is a research laboratory specializing in the development of quantum computing, bringing together experts in physics, computer science, and computer engineering . The team is working to create powerful quantum processors and algorithms to solve problems that are beyond the capabilities of classical computers.

In 2019, Google Quantum AI announced that it had achieved quantum supremacy, performing a calculation that would take the most powerful supercomputer tens of thousands of years to complete.

The laboratory's research is aimed at developing new technologies for artificial intelligence, optimization, quantum chemistry, and simulations of complex systems, in particular through the Google Quantum AI platform, which provides access to quantum computing based on cloud technologies (Fig. 1.16).

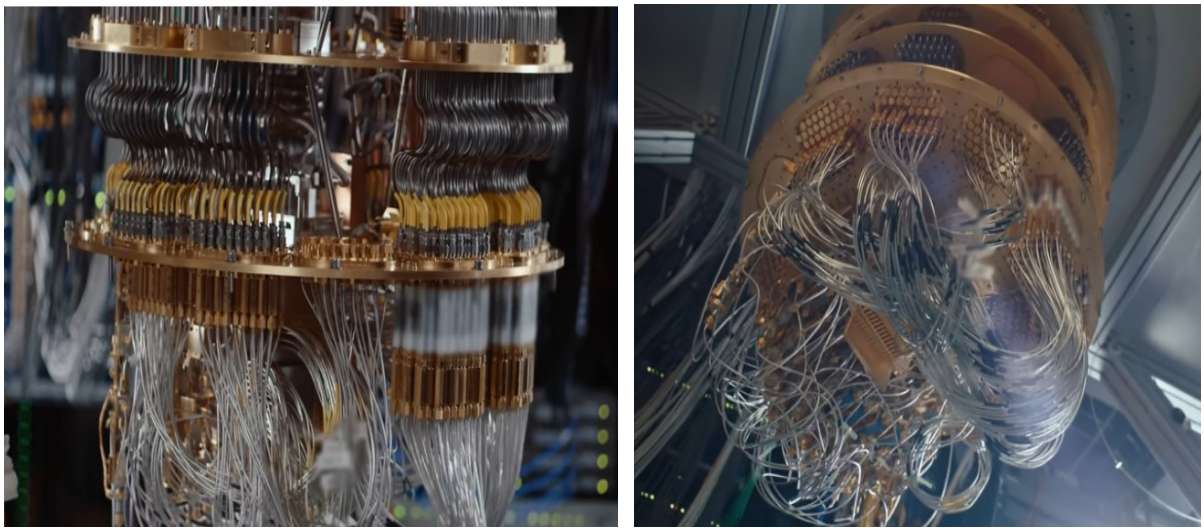


Fig. 1.16. Google Quantum AI lab

IBM is developing quantum processors, such as the IBM Quantum System One, that deliver quantum computing through cloud platforms. With the development of hardware solutions and algorithms, quantum computing can become a key element for solving the most complex scientific, technological, and engineering problems in the future.

One of the key advantages of quantum computing lies in its ability to perform computations that are infeasible for classical computers. Quantum processors leverage the principles of superposition and entanglement, enabling them to explore vast solution spaces simultaneously. This makes them particularly suitable for applications such as optimization problems, cryptography, and drug discovery. For example, researchers can utilize quantum algorithms to simulate molecular interactions with unprecedented precision, potentially revolutionizing the field of chemistry and leading to the development of new materials and medicines.

IBM's approach also emphasizes the democratization of quantum computing through its cloud-based platform, IBM Quantum Experience. This platform provides access to quantum processors for educational institutions, researchers, and developers worldwide, fostering collaboration and innovation. The ecosystem around IBM Quantum integrates tools like Qiskit, an open-source quantum programming framework, allowing users to experiment with and implement quantum algorithms.

By promoting widespread access to quantum technology, IBM is accelerating the pace of quantum research and paving the way for practical applications in industries ranging from finance to logistics.

Control questions and tasks for topic 1

List of theoretical questions

1. What is big data and why is it important in today's world?
2. What factors are driving the growth of data volume today?
3. What are the main characteristics of big data that determine its classification?
4. What are the benefits and challenges associated with big data storage and analytics?
5. What opportunities does the connection of Internet of Things devices provide in the context of big data?
6. What are the main differences between structured and unstructured data?
7. How does cloud computing technology facilitate the processing and storage of big data?
8. What are the main features of the Google Colab environment that distinguish it from traditional on-premises software development environments for data analysis and machine learning?
9. How does Google Colab enable collaboration and sharing of Jupyter notebooks in the cloud and what integration capabilities does it provide with other Google services and external data?
10. What is the concept of fog computing and how does it relate to big data?
11. What industries and applications can use fog computing technologies ?

12. What are the advantages and limitations of using fog computing technologies?

13. What are supercomputers, what are the main possibilities of their application in the modern world?

14. How is the performance of supercomputers evaluated, and what does the unit of measurement FLOPS mean?

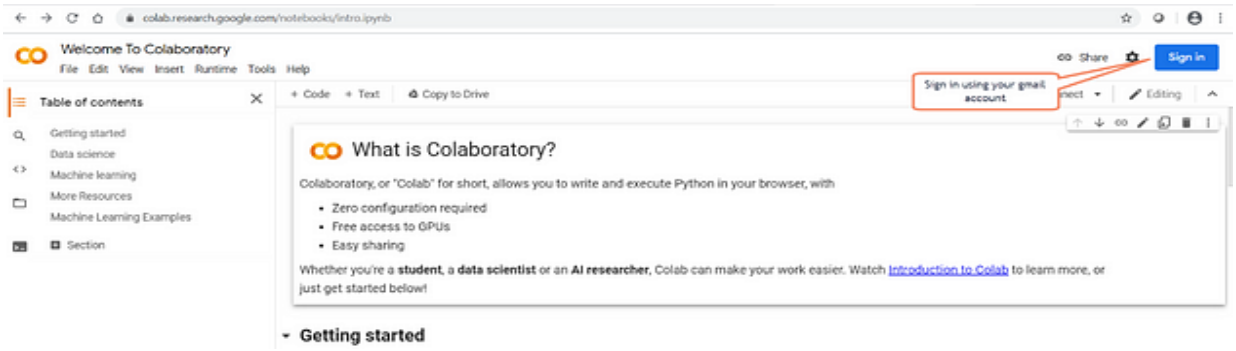
15. What is the principle of operation of quantum computers, and how do they differ from traditional computers?

16. What are qubits, and how are they used to store and process information in quantum computing?

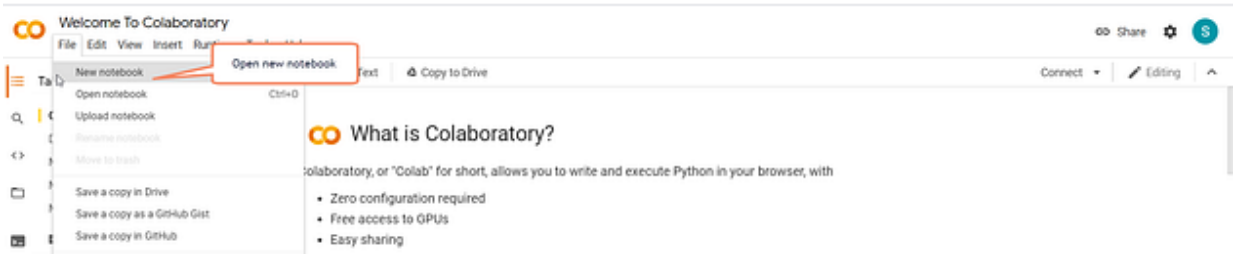
List of tasks for independent work

Practical task. Getting started in Google Colab.

Step 1. Sign in to Google Colab using your Gmail credentials.



Step 2. Click "Open New Notepad" on the File menu.



Step 3. Rename the notebook.

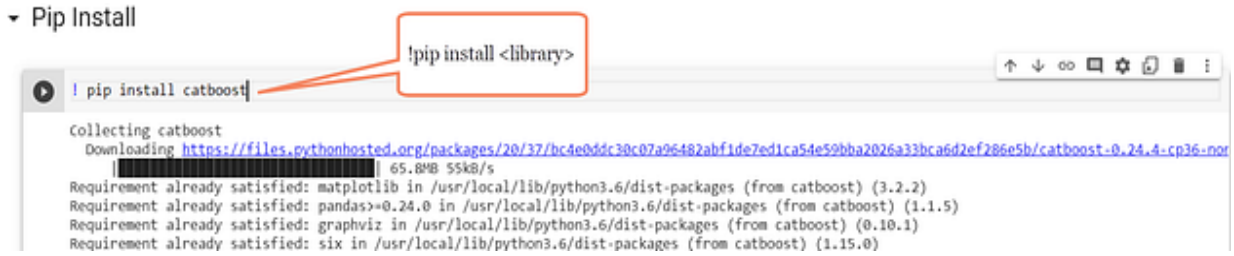
Step 4. Google Colab provides the ability to view information about the computing resources available for each session. Use the following Python code to view information about the virtual machine and its resources:

```
import psutil
psutil.virtual_memory()
```

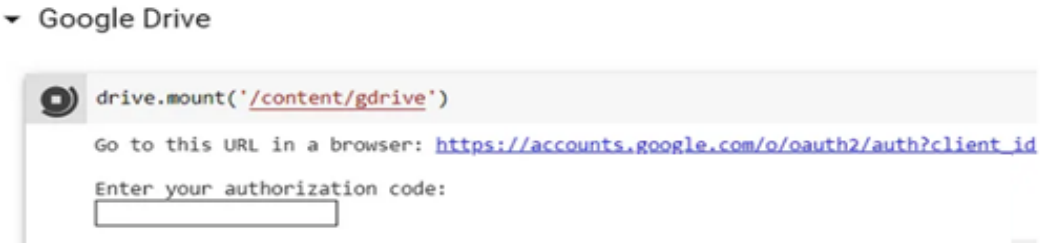
This code will output information about the total amount of available memory for the virtual machine, as well as the used and free memory.

Step 5. Write simple Python code, for example, to print "Hello, World!" or to perform basic arithmetic operations. Run the cell and verify that the code works correctly.

Step 6. Install and import Python packages. Google Colab provides pre-installed libraries for machine learning and deep learning, such as Keras, TensorFlow, and PyTorch. To install a specific library, run the command "`!pip install`".



Step 7. Connect Google Drive to the runtime virtual machine. After running the following script, click the link to get an authorization code. Paste the code into the text box and execute.



Step 8. Import datasets directly from your Google Drive or from a link from the web. Create code that imports the dataset using google.colab to access files on Google Drive. Demonstrate how you can process the imported data in Colab.

Step 9. Explore how to enable GPU support in your Colab notebook settings. Write simple code that uses `tensorflow` or `pytorch` to demonstrate the computational speedup of using a GPU compared to a CPU. Compare the execution times of operations on and off the GPU.

Step 10. Find the "File" option in the top menu and click on it to open the drop-down menu. Select "Save" or "Save a copy in Drive". If this is your first time saving a notebook, Colab will prompt you to save it to Google

Drive. If the notebook has already been saved and you want to update the saved copy, simply select "Save". If you want to save a new copy of the notebook to your Google Drive, select "Save a copy in Drive".

After selecting the save option, you will be prompted to rename your notebook. Enter the desired name and click "OK". After saving, go to your Google Drive to verify that the notebook is actually saved. By default, Colab creates a "Colab Notebooks" folder on Google Drive where all created notebooks will be stored.

List of used and recommended literature

1.1. Maurizio Capra, Riccardo Peloso, Guido Masera, Massimo Ruo Roch and Maurizio Martina. Edge Computing: A Survey On the Hardware Requirements in the Internet of Things World. Future Internet 2019, 11, 100. doi:10.3390/fi11040100 .

1.2. Understanding Cloud Computing Basics, Architecture And Characteristics. [Electronic resource]. – Access mode: https://ostechnix.com/cloud-computing-basics/

1.3. Atlam H.F., Walters R.J., Wills G.B. Fog Computing and the Internet of Things: A Review. Big Data Cogn. Comput. 2018, 2, 10. https://doi.org/10.3390/bdcc2020010

1.5. What is Colab. [Electronic resource]. – Access mode: https://colab.research.google.com/notebooks/intro.ipynb#scrollTo=5fCEDCU_qrC0

1.6. Eleanor G. Rieffel, Wolfgang H. Polak. Quantum Computing: A Gentle Introduction. MIT Press, 2014. 389 p. ISBN: 978-0262028397.

1.7. Rio Yokota, Weigang Wu, Satoshi Matsuoka (Eds.). Supercomputing Frontiers: 10th Asian Conference, SCFA 2020. Springer, 2020. R. 103-120. DOI: 10.1007/978-981-15-3444-3.

1.8. Thomas Sterling, Matthew Anderson, Maciej Brodowicz. High Performance Computing: Modern Systems and Practices. Morgan Kaufmann, 2017. 718 p. ISBN: 978-0124201583.

1.9. Jack D. Hidary. Quantum Computing: An Applied Approach. Springer, 2019. R. 45-72. DOI: 10.1007/978-3-030-21765-6.

1.10. Georg Hager, Gerhard Wellein. Introduction to High Performance Computing for Scientists and Engineers. CRC Press, 2010. 356 p. ISBN: 978-1439811924.

Topic 2. Big data analytics lifecycle

Big data analytics lifecycle represents the sequential stages of processing and analyzing data to obtain valuable information and make decisions. It includes the stages of data extraction, transformation, analysis, modeling, interpretation, decision-making, and monitoring. The life cycle provides a systematic approach to data analysis and helps use data to solve real-world problems.

Software methods for extracting, transforming, and loading data are key to preparing data for analytics. Data extraction involves identifying sources and methods for obtaining information. Data transformation involves cleaning, format conversion, aggregation, and other operations to prepare data for analysis. Data loading involves moving data to a primary repository for further use. The Python programming language has a powerful collection of libraries that provide a variety of capabilities for big data analysis.

With libraries such as Pandas, NumPy, Matplotlib, Scikit-learn, and TensorFlow, analysts can load, clean, visualize, model, and draw conclusions from large amounts of data. This makes Python one of the most popular programming languages for solving big data analytics problems due to its ease of use and large community of developers and researchers.

Web scraping and web crawling help collect information from the Internet, indexing creates systems for searching and distributing data, and web automation simplifies the process of interacting with web services. Machine-readable data and APIs play an important role in the interaction between programs, applications, and systems, allowing for efficient and automated exchange of information and functionality.

HTML parsers and DOM analysis allow to extract the necessary data from the structure of web pages. Recognizing semantic annotations helps understand the meaning of data for further analysis. Web page parsers using computer vision open up wide opportunities for analyzing objects, text, and actions on web pages.

2.1. Key stages of the analytics lifecycle

Big data analytics lifecycle is a process that includes various stages from collecting and preparing data to extracting valuable information and making decisions based on the analysis of this data. This cycle helps

organizations effectively use large amounts of data to achieve their goals and development. Consider the main stages of the big data analytics lifecycle.

Stage 1. The data collection stage involves determining the data sources (databases, logs, sensors, social networks) from which data will be collected.

Stage 2 . The data preparation stage involves cleaning and filtering data for inaccuracies, missing data, and anomalies, combining data from different sources and tables to create linked data sets, and transforming and aggregating data to prepare for analysis. This stage is crucial because the quality of the input data directly impacts the accuracy and reliability of the analysis. Cleaning data involves identifying and handling issues such as duplicates, inconsistent formatting, or outliers that can skew results. Missing data is addressed using techniques such as imputation, where reasonable estimates replace missing values, or exclusion, where incomplete records are removed, depending on the context and requirements of the analysis.

Data integration plays a significant role in this process, as organizations often store information in multiple systems and formats. Combining data from different sources, such as databases, APIs, or flat files, ensures a holistic view of the dataset. This step may also involve resolving discrepancies between data schemas, reconciling naming conventions, or standardizing units of measurement to create a cohesive dataset suitable for analysis.

Transformation and aggregation are final but equally important components of data preparation. Transformation involves restructuring data to meet the specific requirements of the analytical tools or models being used. This might include normalizing data, encoding categorical variables, or creating new derived variables. Aggregation, on the other hand, summarizes data at different levels of granularity, such as calculating average sales per month or total revenue by region, to simplify complex datasets and highlight key trends or patterns.

Proper execution of the data preparation stage not only improves the quality of analysis but also ensures that subsequent stages, such as modeling and visualization, are carried out more efficiently.

Stage 3. The data analysis and visualization stage involves using statistical methods, machine learning, and other analytical tools to extract valuable information from the data, and visualizing the analysis results using graphs, charts, and other visual tools. This stage begins with selecting the appropriate analytical methods and tools based on the type of data and the

objectives of the analysis. Statistical methods, such as regression, clustering, or hypothesis testing, are often used to identify patterns, correlations, and trends. Machine learning techniques, including classification, predictive modeling, or natural language processing, provide advanced capabilities for extracting information from complex and unstructured data. These methods enable analysts to uncover hidden relationships, forecast future trends, and make data-driven recommendations.

Visualization is a critical part of this process, as it transforms raw data and complex analysis results into intuitive visual representations that are easier to interpret and communicate. Graphs, charts, heatmaps, and dashboards allow stakeholders to explore data interactively and gain a clearer understanding of the underlying patterns. For example, a time-series graph might highlight seasonal trends, while a scatterplot can reveal correlations between variables. Modern visualization tools, such as Tableau, Power BI, and Python libraries like Matplotlib and Seaborn, provide a wide range of options for creating compelling visual outputs.

Clear visualizations help bridge the gap between complex analytics and practical application, allowing business leaders, researchers, or policymakers to grasp the key findings quickly and take informed actions. Ultimately, the data analysis and visualization stage plays a pivotal role in turning raw data into meaningful information. It empowers organizations to identify opportunities, mitigate risks, and optimize processes, leveraging the power of data to drive strategic outcomes. When executed effectively, this stage not only answers existing questions but also raises new ones, paving the way for continuous improvement and innovation.

Stage 4. At the modeling and forecasting stage mathematical models are being developed to forecast and predict future events based on data analysis, using machine learning algorithms to build models that can detect patterns and dependencies in data. During this stage, the process begins with selecting the appropriate type of model based on the nature of the data and the goals of the analysis. For instance, regression models are widely used for predicting continuous outcomes, while classification models are employed to categorize data into predefined groups. Machine learning algorithms, such as decision trees, neural networks, or support vector machines, are applied to build models capable of recognizing complex patterns and relationships in the data that might not be evident through traditional statistical methods.

Model training is a key component of this stage, where algorithms learn from a subset of the data, known as the training set. The model is trained by optimizing its parameters to minimize errors and accurately capture the dependencies in the data. Once trained, the model is validated and tested using separate data subsets to assess its performance and ensure that it generalizes well to unseen data. Metrics such as accuracy, precision, recall, or mean squared error are commonly used to evaluate model quality and identify areas for improvement. In forecasting, these models are then used to make predictions about future events or trends. For example, a time-series model might predict future sales based on historical data, or a machine learning model might identify the likelihood of equipment failure in industrial settings. Scenario analysis can also be conducted by simulating various conditions and observing their potential impact, allowing organizations to prepare for different outcomes.

Stage 5. Next comes the interpretation of the results, in establishing connections between data and conclusions for decision-making. This is the process of understanding and taking into account the obtained conclusions and patterns for making management decisions or improving business processes. This includes analyzing data to identify key differences and trends, assessing the impact of factors on business processes, and making strategic decisions based on the conclusions obtained. The results of the analysis are used to develop strategies, optimize business processes, and implement changes in the organization in order to achieve greater efficiency and competitiveness.

Stage 6. Decision making provides in making recommendations and changes based on data analysis. Decision-making based on data analysis can involve a variety of scenarios in different industries. For example, in e-commerce, customer purchase data can be used to personalize product and service recommendations, which helps increase sales and customer satisfaction. In the financial sector, data analysis can help detect fraudsters by identifying anomalous patterns in financial transactions, for example, recognizing unusual transactions, such as unusually large amounts of money transferred at night. In healthcare, data can be used to predict the spread of diseases (for example, to predict the spread of influenza by analyzing patient visits to hospitals in different regions) and to develop effective treatment strategies.

Stage 7. At the monitoring and optimization stage the results of the decisions made are tracked and their effectiveness is assessed, adjustments are made and the strategy is improved based on the data obtained. Monitoring and optimization occurs through constant tracking of the results of the decisions made and their effectiveness is assessed based on the data collected. In a marketing strategy, tracking customer reactions to advertising campaigns can be used to determine their effectiveness. For example, in a marketing strategy, analyzing clicks on advertising banners and purchases after viewing an ad helps to assess the effectiveness of the campaign.

After analyzing the results, adjustments can be made, such as changing the target audience or advertising channels, to optimize the strategy. In the field of manufacturing, monitoring can include tracking the costs of materials and labor, which can help identify overspending of materials at a certain stage of the production process.

The data obtained can be used to improve processes and strategies, allowing enterprises to continuously improve their productivity and competitiveness.

2.2. Methods for big data extracting, transforming and loading

Extraction, transformation, and loading (ETL) of big data is the process of processing and preparing large amounts of information for further analysis and use. These techniques play a key role in a distributed big data architecture, where data is collected from various sources, transformed and cleaned, and then loaded into storage systems or data warehouses for further analysis. It is a process in which unstructured or structured data is extracted from disparate data sources.

The main stages of ETL include the processes of extracting, transforming, and loading data (Fig. 2.1). In the extraction phase, data is collected from multiple sources, including databases, APIs, or flat files. During transformation, the raw data is cleaned, formatted, aggregated, or enriched to meet the analytical or operational requirements. Finally, in the loading phase, the processed data is stored in the target system for querying and analysis. ETL is essential for integrating and preparing data to ensure accuracy, consistency, and accessibility for decision-making processes.

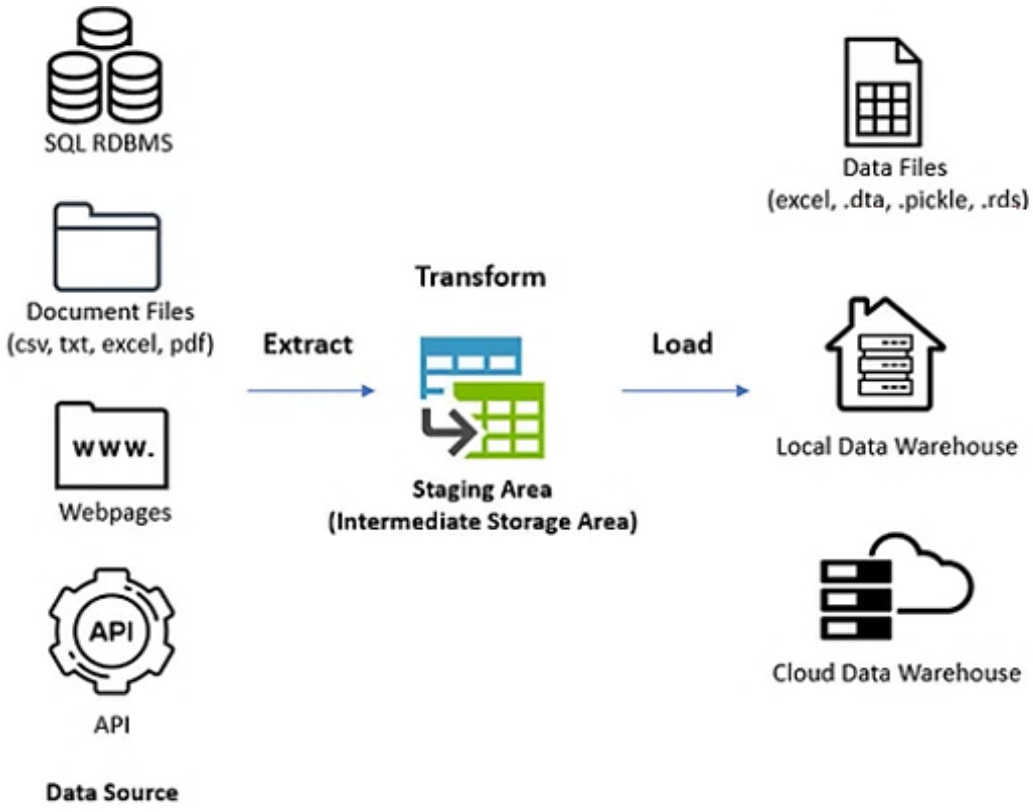


Fig. 2.1. ETL process (Extraction, Transformation, Loading)

Data warehouse is a specially organized data repository designed to store and analyze large volumes of structured and semi-structured data from various sources. This repository serves to integrate data from various sources, cleanse, transform, and store it in a single centralized system, which simplifies analytics and provides access to consistent, analysis-ready data.

Data Warehouse is commonly used to support business analytics, reporting, and data-driven strategic decision-making. For example, in large corporations that have a large amount of transactional and customer data, Data Warehouse allows information to be stored in one centralized location for further analysis and reporting. This repository can contain data from various sources, such as transactional databases, CRM systems, web servers, and other sources.

Data Warehouse provides a convenient interface for selective access to data, as well as for executing complex analytical queries, which helps in making strategic decisions based on data.

Data extraction begins with identifying the data sources from which information needs to be extracted. These can be databases, files, APIs, web services, and other sources (Fig. 2.2).

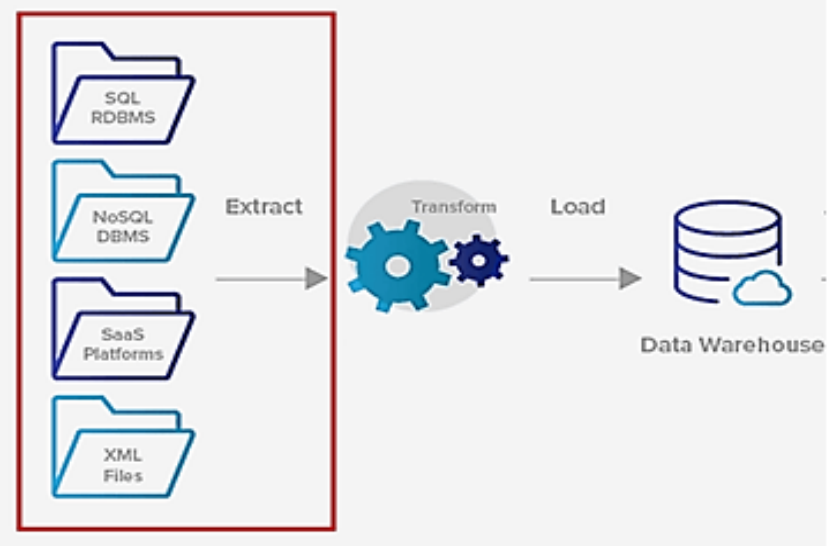


Fig. 2.2. Data extraction process (Extract)

The pipeline must first extract or accept information and data from a source or multiple sources.

Big data source systems often include:

- API (Application Programming Interface);
- websites;
- data lakes are centralized repositories designed to store, process, and protect large volumes of structured, semi-structured, and unstructured data that can store data in its native format and process any of its varieties, ignoring size limitations;
- SaaS (Software as a Service) applications;
- relational databases, or transactional databases.

Pipelines can receive raw data in real-time, i.e. immediately after it appears in the source. They can also use a batch processing workflow, which means that new changes are picked up at a set interval. The optimal way to extract data from the sources is chosen. This can be batch extraction, incremental extraction, or full extraction. Data mining is the actual process of retrieving data from a source and storing it in a repository, such as a relational database or cloud storage.

Data transformation is the second step of the ETL process. In this step, the raw data received from the source is transformed into data that is compatible with a specific purpose and use case. For example, a business intelligence tool may require JSON data that conforms to a specific schema,

but the source data is in CSV format. The transformation step of an ETL pipeline can change data from one format to another.

This is just one example, and the use cases are very diverse. But almost always, data needs to be transformed at some point during its lifecycle. In an ETL pipeline, this transformation occurs almost immediately after data extraction. The most common types of data processing include processes such as:

- filtering;
- aggregation;
- data cleaning;
- feature extraction;
- reformatting data according to the schema.

These relatively simple transformations serve a critical purpose. They ensure that when a data set enters a repository or other application, it is "clean" consistent, error-free, and ready to be used to achieve business goals. In general, the transformation phase can be divided into the processes of data cleansing, transformation, and validation.

Data cleaning is the identification and correction of inaccuracies, missing data, anomalies, duplicates, and other deficiencies in data.

Data conversion is the changing of the format, structure, and values of data to suit the needs of the analysis. This may include joining tables, calculating metrics, aggregating data, and other operations.

Data validation is the validation of data after transformations to ensure that it meets expectations. For example, data validation can be used to verify the correctness of orders and payments in an online store. The system can check whether the data entered by the user during the order process meets the expected formats and constraints (for example, whether the credit card number is entered correctly or whether the order value is within the acceptable range).

Loading is the process of defining a data warehouse, selecting a storage system to load the processed data into. It can be a relational database, a NoSQL database, a cloud storage system, or another system. It is the process of moving the processed data from the temporary storage to the main storage. It can be a one-time load or a variable (incremental) load.

Data indexing and optimization is the process of creating indexes and other optimization structures to speed up data access.

Creating indexes on tables in SQL is used to speed up query execution. For example, creating an index on the `customer_id column` in the `orders table`.

Data caching is used to store frequently used data in memory to reduce access time. Using structures such as inverted indexes is used to quickly search large text arrays, as in search engines. Consider the Python code that allows users to use indexing in the Pandas library to optimize data access:

```
import pandas as pd
# Creating a test DataFrame
data = {
'order_id': [101, 102, 103, 104, 105],
'customer_id': [1, 2, 3, 1, 2],
'order_amount': [250, 450, 300, 200, 400]
}
df = pd.DataFrame(data)
# Setting an index to optimize access
df.set_index('order_id', inplace=True)
# Select a specific row by index
order_details = df.loc[101] # Quick access to order data with ID 101
# Output the result
print("Order details with ID 101:")
print(order_details)
# Query optimization using filtering
filtered_data = df[df['order_amount'] > 300] # Orders with an amount greater than 300
print("\nOrders with an amount greater than 300:")
print(filtered_data)
```

```
Order details with ID 101:
customer_id       1
order_amount    250
Name: 101, dtype: int64

Orders with an amount greater than 300:
          customer_id  order_amount
order_id
102                 2           450
105                 2           400
```

We created a DataFrame with orders. We set the `order_id` column as an index, which allows us to quickly access records by this identifier.

We used an index to select an order with a specific `order_id`. Filtered to select records with an order amount greater than 300. This indexing significantly improves performance when working with large data sets.

A modern enterprise typically has many different data sources, as well as different goals that require data to be used in different ways and by different systems. For example, these can be security measures, marketing and sales goals, financial optimization needs, and organization-wide solutions. How do we make sure that data is in the right place in the right format? This is where data pipelines, including ETL pipelines, come in. They allow enterprises to automatically move data between systems with data transformation. ETL software methods help ensure the quality and consistency of data that will be used for analysis. They also help prepare data for further use in various analytical tools and ensure efficient information exchange between different systems.

Batch-based ETL pipelines are easy to implement but introduce delays that are unacceptable in situations such as detecting financial fraud, managing rapidly changing inventory in a warehouse, or engaging customers with online sales in real time. In such cases, real-time or streaming ETL pipelines are required to process and analyze data as it arrives, ensuring immediate results and actions.

2.3. The role of the Python programming language for big data analytics

The Python programming language was created by Guido van Rossum and released in 1991. The language was the result of an attempt to create a simple programming tool based on an object-oriented approach.

The language gradually gained popularity among programmers due to its simplicity, efficiency, and wide range of applications, including web development, scientific computing, artificial intelligence, and data analysis. The Python programming language plays an important role in big data analytics. Python has a large number of libraries specialized in data analysis, such as NumPy, Pandas, Matplotlib, Seaborn, SciPy, Scikit-learn, and others. These libraries provide powerful tools for processing and analyzing large amounts of data. Python also has many tools for data visualization, which allow to create graphs, charts, and other visual elements to better understand data. Matplotlib and Seaborn are examples of such libraries.

The Python language has a number of multifunctional libraries that allow to perform big data analysis and related tasks (Table 2.1).

Table 2.1. **Capabilities of Python libraries for solving big data analytics tasks**

Task	Python library capabilities
Loading and cleaning data	The Pandas library provides powerful tools for loading, transforming, filtering, and cleaning data from various sources such as CSV, Excel, databases, and many others.
Basic data analysis	The Pandas library allows to perform statistical analysis, calculate means, medians, standard deviations, and other data characteristics. The NumPy library provides capabilities for working with numerical data, arrays, and mathematical operations.
Data visualization	Matplotlib, Seaborn, Plotly, and other libraries allow to create graphs, charts, scatter plots, and other visuals to understand data.
Machine learning and deep learning	Libraries such as Scikit-learn, TensorFlow, and PyTorch provide tools for building machine and deep learning models to predict outcomes, classify, cluster data, and perform other scientific tasks.
Processing large amounts of data	Dask allows users to distribute computations to work with large data sets that do not fit in RAM. Vaex is a library that is optimized for working with big data and provides fast analysis without loading data into memory.
Streaming data processing	Libraries such as Apache Kafka (with clients for Python), Faust for building streaming applications, PySpark Streaming for distributed stream processing, and Streamz for simple streaming computing.
Text analysis and natural language processing	NLTK, spaCy, TextBlob, and other text analysis libraries allow to perform natural language processing, extract information from text, classify text data, and much more.
Social media analysis	Some libraries, such as NetworkX, allow to analyze the structure and interactions of large social network graphs.
Working with distributed computing	For large data analysis and machine learning tasks, libraries such as Dask and Apache Spark can be used to run on clusters of compute nodes.

Python is one of the most popular programming languages, which fosters an active developer community. This means there is always access to resources, documentation, and help from other experts. Python integrates with other programming languages and tools, making it an excellent choice for developing analytical applications and web services.

Pandas library allows efficiently process large amounts of data, including working with data in CSV, Excel, SQL, and other formats. The name of the library "pandas" comes from the term "panel data", which refers to the multidimensional structured data sets that the library is designed to work with. The Python language is a popular choice for developing machine learning models and working with libraries such as TensorFlow, Keras, PyTorch, and Scikit-learn.

2.4. Web scraping, web crawling, indexing, and web automation

Web scraping, web crawling, indexing, and web automation technologies and software methods play an important role in collecting and processing information from web resources. They allow to obtain data from the Internet, structure it, and use it for analysis, indexing, or automation of various tasks. The main methods and technologies include web scraping, web crawling, indexing, and web automation.

Web scraping is the process of automatically collecting data from websites. Web scraping uses programs (scrapers) that can analyze the HTML code of web pages and extract the necessary data. Web scraping can include extracting text content, images, tables, product information, etc.

Software web scraping methods include various techniques and libraries that allow automatically collect data from web pages.

Let's explore some of the most widely used techniques for web scraping.

One common web scraping method is to use libraries such as `requests` in Python to make HTTP Requests to websites and retrieve the HTML code of the pages. Another method is to use HTML and XML parsers such as BeautifulSoup (bs4) or lxml in Python to analyze and extract specific data from the HTML structure of a web page.

HTML parsers are tools and libraries that help extract and process information from the HTML code of web pages. They are often used for web scraping, web crawling, and analyzing large amounts of data from the Internet. For example, a company might use an HTML parser to extract product prices, reviews, and availability information from e-commerce websites to analyze competitors' pricing strategies. Similarly, researchers can

use parsers to gather and analyze data from news websites or social media platforms to study trends, sentiment, or public opinion on a specific topic.

PyQuery is a library that uses CSS selector syntax to extract data from HTML and provides a convenient interface for parsing and processing HTML code. For instance, PyQuery can be used to scrape the titles, authors, and publication dates of articles from a news website by selecting specific HTML elements such as `<h1>` for titles or `<span>` with class attributes for authors. This makes it an efficient tool for developers who need to extract structured data from web pages without writing complex parsing logic.

Goutte is a PHP library for parsing HTML using CSS selector syntax, which allows interaction with HTML code on the server side. For example, Goutte can be used to automate the extraction of product details, such as names, prices, and descriptions, from an e-commerce website. By leveraging CSS selectors to target specific HTML elements, developers can efficiently gather data for analysis or integration into other applications, all while handling HTTP requests and responses seamlessly on the server side.

JSoup is a Java library for parsing and processing HTML code, allowing to extract data using CSS selectors and perform various HTML manipulations. For example, JSoup can be used to scrape and process news articles from a website by extracting specific elements like titles, publication dates, and article content using CSS selectors. Additionally, it can be employed to clean up messy HTML, modify elements, or validate and fix improperly structured HTML documents, making it a powerful tool for both web scraping and data preprocessing tasks.

HtmlAgilityPack is an HTML parsing library on the .NET platform (C#) that allows users to extract and process data from HTML code. For instance, HtmlAgilityPack can be used to extract product information, such as prices and descriptions, from e-commerce websites by parsing the HTML structure. It is also widely used for web scraping, cleaning up malformed HTML documents, or automating tasks like extracting metadata, hyperlinks, or specific tags from large volumes of web pages in .NET applications.

The considered HTML parsers help provide efficient and accurate parsing of HTML code, which is important for web scraping, web page analysis, and data extraction from the Internet for further processing and analysis of large volumes of data.

DOM analysis (Document Object Model analysis) is the process of interacting with the DOM structure of a web page to extract, process, and analyze data. The DOM structure represents a web page as a tree of objects that represent various elements of the page, such as HTML tags, text blocks, links, images, and so on.

Big data can include a large number of web pages, and DOM analysis for big data involves using efficient and optimized methods to process these pages.

Consider the main stages of DOM analysis for big data.

Step 1. First, we need to get the web pages from which we want to extract information. Big data can include thousands or even millions of pages, so effective web scraping or web crawling is an important part of DOM analysis.

Step 2. After retrieving the pages, their HTML code must be parsed to create a DOM structure. This is an object tree that displays the hierarchy of page elements.

Step 3. We can select the data from the DOM structure using CSS selectors or XPath queries. The selection can be targeted, where specific elements are removed, or bulk, where many pages are processed simultaneously.

Step 4. After data sampling, further processing, filtering, aggregation, and data analysis can occur. This can include pattern recognition, trend detection, statistical analysis, and more.

Step 5. The final DOM analysis results can be saved as structured data, such as CSV, JSON, or a database, for later use.

Beautiful Soup is a Python library for parsing HTML and XML data, which provides a simple and user-friendly interface for efficiently extracting data from web pages, allowing to analyze the structure of HTML code, search for and extract the necessary information.

BeautifulSoup allows quickly perform various data manipulations, such as searching for elements by tags, classes or attributes, as well as extracting text, URLs, links and other information. The library allows users to extract data from various parts of HTML using CSS selectors or XPath queries and is a powerful tool for web scraping and data analysis from web pages in Python.

Let's look at an example of DOM parsing using the Python programming language and the BeautifulSoup library. In this example, we will use DOM parsing to extract article titles from a news webpage. Suppose we have a webpage with a list of articles in the following HTML structure:

```
<!DOCTYPE html>
<html>
<head>
<title>News</title>
</head>
<body>
<div class="article">
<h2>First article</h2>
<p>Description of the first article...</p>
</div>
<div class="article">
<h2>Second article</h2>
<p>Description of the second article...</p>
</div>
<!-- other articles... -->
</body>
</html>
```

Let's write Python code to parse this page and extract article titles:

```
from bs4 import BeautifulSoup
# HTML code of the web page (can be read from a file or a web request)
html_content = """
<!DOCTYPE html>
<html>
<head>
<title>News</title>
</head>
<body>
<div class="article">
<h2>First article</h2>
<p>Description of the first article...</p>
</div>
<div class="article">
<h2>Second article</h2>
<p>Description of the second article...</p>
</div>
<!-- other articles... -->
</body>
</html>
```

```
# Creating a BeautifulSoup object for HTML parsing
soup = BeautifulSoup(html_content, 'html.parser')
# Find all <h2> elements in the "article" class and output their texts
articles = soup.find_all('div', class_='article')
for article in articles:
title = article.find('h2').text
print("Article title:", title)
```

We used the BeautifulSoup library to parse the HTML code of the page, found all `<div> elements` with the class "`article`", and extracted the title text from the `<h2> elements`. The result of running the code will look like this:

```
Article Title: First Article
Article title: Second article
```

This example demonstrates how to use DOM analysis and the BeautifulSoup library to extract information from the HTML code of web pages for further analysis and processing of large amounts of data.

DOM analysis for big data can be a challenging task due to the volume of data and the processing of a large number of pages. Optimization and effective selection of methods for web scraping, parsing, and processing are important aspects of successfully completing such tasks.

CSS selectors can be used to select elements on a page that should be removed. Let's look at an example.

```
from bs4 import BeautifulSoup
# Suppose we have HTML code in the html_content variable
html_content = """
<html>
<head>
<title>Example</title>
</head>
<body>
<div class="content">
<h1>Heading</h1>
<p>This is paragraph 1</p>
<p class="important">This is paragraph 2</p>
<a href="https://www.example.com">Link</a>
</div>
</body>
</html>
"""
# Create a BeautifulSoup object
soup = BeautifulSoup(html_content, 'html.parser')
# Use a CSS selector to select all <p> elements with the class "important"
important_paragraphs = soup.select('p.important')
```

In this example, we use the `select() method` to select all `<p> elements` with the class "`important`" on a web page.

Regular expressions can be used to find and remove text content that matches certain patterns. Let's look at an example of Python code using regular expressions.

```
import re
# Suppose we have a string of text
text = "This is sample text with several phone numbers: +1234567890, +9876543210."
# Template for searching phone numbers in the format +1234567890
phone_pattern = r'\+\d{10}'
# Find all phone numbers in text using regular expression
phone_numbers = re.findall(phone_pattern, text)
# Display the found phone numbers
print("Phone numbers found:")
for phone_number in phone_numbers:
print(phone_number)
# Remove all phone numbers from the text
cleaned_text = re.sub(phone_pattern, '', text)
# Output the cleaned text
print("\nText without phone numbers:")
print(cleaned_text)
```

In this example, we use the `re module` to create a regular expression to search for phone numbers in a text. We then use the `findall()` and `sub() methods` to find all phone numbers in the text and remove them.

Using browsers in "headless" mode (without a graphical interface) can be used to display and interact with web pages. The web browser is launched without a graphical interface, that is, without a window display. Instead, it runs in the background, performing all the operations that would normally occur in a regular browser, but without a visual display. This allows the browser to be used in automation scripts, web page testing, web content scraping, etc., without having to open a browser window. This mode is especially useful in situations where we want to interact with a web page using code rather than through a graphical interface.

Another method of web scraping is API access, which is the use of an application programming interface to retrieve structured data from a website. In particular, some websites provide public APIs for retrieving data. One example of using API access for web scraping in Python is to use the `requests library` to interact with the website's API and retrieve data in a structured format.

For example, if a website has an API for retrieving information about books, we can use the following code to retrieve a list of books in JSON format:

```
import requests
url = 'https://example.com/api/books'
response = requests.get(url)
if response.status_code == 200:
books = response.json()
for book in books:
print(book['title'], book['author'])
else:
print('Failed to fetch data:', response.status_code)
```

In this example, we perform a GET request to the URL https://example.com/api/books, receive the response in JSON format, and process it, displaying the title and author of each book.

Scrapy is a Python web scraping and crawling framework that has a built-in page processing system and capabilities for extracting data from web pages. Using the Python web scraping framework Scrapy allows developers to create powerful scrapers with own logic and data processing. Let's look at an example of Python code that uses the Scrapy framework to scrape data from a web page:

```
import scrapy
class MySpider(scrapy.Spider):
name = 'myspider'
start_urls = ['http://example.com']
def parse(self, response):
for quote in response.css('div.quote'):
yield {
'text': quote.css('span.text::text').get(),
'author': quote.css('span small::text').get(),
'tags': quote.css('div.tags a.tag::text').getall(),
}
next_page = response.css('li.next a::attr(href)').get()
if next_page is not None:
yield response.follow(next_page, self.parse)
```

In this example, we create a Spider `named` MySpider that starts with the starting URL 'http://example.com'. The `parse() method` is called to process the response for each page.

We use CSS selectors to select specific elements on the page, and then extract the text and other data we are interested in. We navigate to the next

page, if there is one, using the `response.follow() method`, and continue scraping.

Services such as Import.io and Octoparse offer users graphical interfaces that allow them to directly define patterns for extracting data from web pages without the need for programming. These services usually have built-in tools for processing and filtering data, as well as the ability to automate scraping to regularly update information. They provide various integrations and export options for convenient use of the obtained data in other programs or systems. Thanks to such services, even users without deep technical knowledge can quickly access the necessary information from the Internet.

Web crawling is the automatic movement of a program through different pages of a website to collect data.

Crawlers (bots) can scan sites, follow links, analyze site structure, and collect large amounts of information. Crawlers can use schedulers to determine the order in which page requests are made. This helps to avoid overloading the server and ensure efficient data collection.

Regular expressions and HTML and XML parsers can also be used to find and follow links on pages and analyze the structure of pages and extract links.

Robot processing is the checking of the robots.txt file, which specifies rules for crawlers regarding which pages can be viewed.

Web crawling takes server limitations into account by using "delays" between requests to avoid server overload and unfriendly crawler perceptions.

Web crawling, or web scraping, involves a variety of methods and approaches for collecting data from websites. This includes the use of sessions and cookies to interact with sites in order to perform actions in user mode.

Headless browsers are often used to display and interact with web pages without a graphical interface. In addition, rule-based robots allow to create bots that act according to certain rules and patterns to collect data. Focusing on specific sections of a website allows for targeted crawling to collect specific information.

Processing dynamic web pages that load content using JavaScript is possible with tools such as Selenium, which allow to interact with these pages and retrieve the necessary data.

Indexing is the process of structuring and organizing collected data for the purpose of its subsequent quick and efficient retrieval. During indexing, data can be arranged in the form of an index or database, which allows for quick access to the desired information. The main goal of indexing is to provide fast access to data, reduce search time, and increase performance.

Let's look at some indexing methods for big data.

B-trees are one of the most common data structures for indexing. B-trees are a type of self-balancing search tree. They are widely used in database and file systems due to their efficient search and insertion operations.

The letter "b" in B-trees stands for "balanced" because these trees are designed to maintain balance even as data is constantly being added or removed. B-trees have a variable number of child nodes, typically from two to several, which allows them to efficiently process large amounts of data. The structure of B-trees ensures that the height of the tree remains small, resulting in faster search times. Overall, B-trees are a powerful data structure for processing large data sets.

Shown in Fig. 2.3. A B-tree is an example of a B+Tree because it differs from other B-trees in that it uses two files: one containing the indexes and the other containing the indexes and data. The index file is much smaller than the sum of all the indexes because it contains only the first index of each page in the index file. In simplified form, B+Tree looks like this:

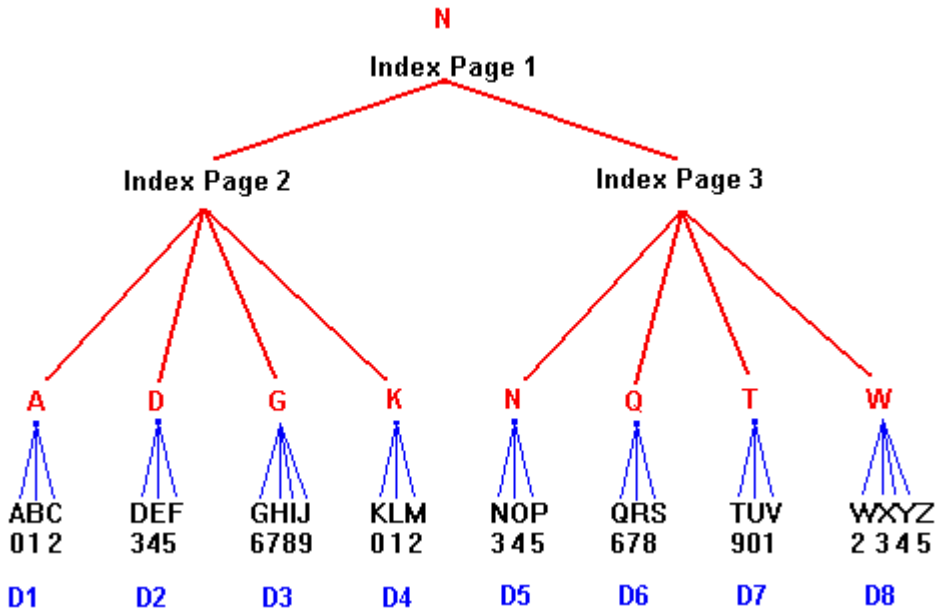


Fig. 2.3. Example of constructing a B-tree

In this example, we need 9 keys to access 26 pieces of data. A web page can contain so much more data that to find out which data page contains a particular piece of data, we only need to look at 2 index pages (i.e. access the disk twice). Let's say we want the data to be stored under index "R". We access the root index page (this is always the first page we open to when searching the index – in this case Index Page 1). "R" is greater than "N", so we look at index page 3. "R" is greater than "Q", but less than "T", so it must be on data page 6 (D6). So we find that the data corresponding to "R" is 7.

The number of levels is determined not only by the amount of data stored in the file, but also by the size of the keys used and the size of the data and index pages. Very large keys on small pages will increase the number of index pages, and therefore the number of index "levels". There are two different types of levels in a tree. Index pages 2 and 3 are called "leaf" levels of the tree, that is, they contain pointers to data pages. Index page 1 is at the "non-leaf" level and contains pointers to other index pages. Since non-leaf levels are used primarily for tree navigation, they have a different structure than leaf levels. Non-leaf index pages always have one more page pointer than indexes. This is because indexes are used for comparison.

In this illustration, “N” is the only index, but we have two pointers, one to IndexPage 1 and the other to IndexPage 2. If the index we want to find is less than “N”, we go to IndexPage 1, if it is greater than or equal to N, we go to IndexPage 2. This reduces the number of keys needed in each non-leaf block of the index. If there are n levels in a B+ tree, there will always be 1 leaf level and n-1 non-leaf levels. Now consider a less trivial example. Let’s say we want to store 2000 records with an index length of 25 and a data length of 50, such as a set of names with addresses and phone numbers. If we set the data page size and index page size to 512 bytes, we get the following calculations for maintaining the tree:

data+index – each equals 75 + overhead (3);
the data page overhead is 6;
506/78 = 6 index/data pairs per page;
2000/6 = 334 pages of data required;
index – each is 25 + overhead (3);
506/28 = 17 indexes per index page;
334/17 = 20 index pages are required at the leaf level (level 0);
503/ 28 = 16 + 1 implicit key = 17 indexes per index page;

20/17 = 2 index pages required at level 1;

RootIndexPage 1;

Total pages – 23;

Disk access for data sampling – 4.

This comes at a file size penalty. In this case, the index file size is 24 pages (including 1 control page) or 12 KB, and the data file size is 335 pages (including 1 control page) or 167.5 KB – a total of 179 KB for 150,000 bytes of data. Using 512-byte index pages and 4096-byte data pages, we end up with 40 data pages (160 KB) and 5 index pages (2.5 KB), and only need 3 disk accesses to fetch the data. All data levels are stored in one file (SOMENAME.DAT), and the index levels are stored in another (SOMENAME.IND).

Only three general operations are required to maintain a B+ tree database: insert, retrieve, delete. Other special operations are used to sequentially traverse the tree or to efficiently delete large amounts of data. Operations are also required to open and close the tree.

Hash indexes create hash tables for fast lookups using hash functions. They are good for fast access to precisely defined values, but may be less efficient for range queries. For example, hash indexes are commonly used in scenarios where quick retrieval of specific records is essential, such as in caching systems, user authentication databases, or key-value stores. In these cases, the hash function maps keys (such as user IDs or session tokens) directly to their corresponding values, enabling near-instant lookups. Since hash indexes do not maintain data in a sorted order, they are less suitable for operations like finding records within a certain range, such as retrieving all users with IDs between two values.

Bitmap indexes use bit masks to indicate the presence or absence of certain values. They are effective for analytical queries and data filtering. For example, bitmap indexes are widely used in data warehousing and business intelligence applications to speed up analytical queries. In scenarios such as filtering large datasets by categorical attributes (e.g., filtering customer data by gender or region), bitmap indexes create compact representations using bitmaps, where each bit indicates whether a record matches a specific value. This allows for efficient combination of filters through bitwise operations, significantly reducing query execution time compared to traditional indexing methods.

Inverted indexes are commonly used for text search. They create a mapping between words and lists of documents where those words occur. This allows for fast retrieval of textual information. For example, inverted indexes are a core component of search engines like Elasticsearch and Apache Lucene. In a document retrieval system, an inverted index maps each unique word in a collection of documents to a list of document IDs where that word appears. This enables efficient full-text search by quickly locating and retrieving documents containing specific keywords or phrases. For instance, when a user searches for "machine learning," the inverted index quickly identifies all documents containing those terms, allowing for rapid and relevant search results.

Column-store indexes are used to optimize analytical queries. Data is stored in columns rather than rows, which allows for more efficient aggregation and filtering operations. For example, column-store indexes are widely used in data warehousing solutions like Amazon Redshift and Microsoft SQL Server's Columnstore Index feature. In a scenario where a business needs to analyze sales data across multiple regions and products, a column-store index can significantly speed up aggregation operations like summing up total sales or calculating average sales per product category. By storing and processing only the relevant columns (e.g., "sales_amount" and "region") instead of entire rows, the system reduces the amount of data read from storage, making analytical queries much faster and more efficient.

Full-text indexes are used for full-text search in text content. They allow to effectively search for keywords and phrases. Full-Text Indexes are used for full-text search in text content. They allow for effectively searching for keywords and phrases, improving search speed and accuracy. For example, in an online bookstore, a full-text index could be created on the book titles, descriptions, and author names. When a user searches for a specific phrase like "science fiction," the system quickly returns relevant results from the database, even if the exact phrase doesn't match the search terms exactly. This is particularly useful for large datasets, such as product catalogs or articles, where finding the right match quickly is crucial for user experience.

Spatial indexes are used to index geographic data, such as coordinates on a map. They allow for quick search for features located in specific geographic areas. Spatial Indexes are used to index geographic data, such as coordinates on a map. They allow for quick searches for features located in

specific geographic areas, improving the efficiency of spatial queries. For instance, in a mapping application, spatial indexes can be used to find nearby restaurants when a user provides their current location. By indexing geographic coordinates, the system can quickly return results such as restaurants within a defined radius, enabling faster and more accurate location-based searches. This is particularly valuable in services like GPS navigation or location-based recommendations, where real-time search is crucial.

Adaptive indexing is used to automatically adapt indexes depending on the type of queries and system behavior. Caching can be considered a form of indexing where certain queries and their results are stored in memory for quick access. Adaptive Indexing is used to automatically adapt indexes depending on the type of queries and system behavior. It helps optimize performance by adjusting the indexing strategy in real-time based on the query patterns. For example, in a large e-commerce platform, if certain product categories are queried frequently, adaptive indexing can dynamically adjust to optimize the index for those categories, speeding up future searches.

Caching can be considered a form of indexing where certain queries and their results are stored in memory for quick access. This technique is often used to reduce the load on databases by keeping frequently requested data in memory. For instance, in a news website, the most popular articles of the day may be cached so that when users visit the site, these articles load almost instantly, without needing to query the database each time. Both adaptive indexing and caching enhance performance by ensuring that the most relevant data is readily available, reducing query time and improving the overall user experience.

In-memory databases are used to store and index data directly in memory, allowing for fast data access. Some examples of In-Memory databases include Redis, Memcached, Apache Ignite, and VoltDB. Redis is used for caching and performance, Memcached for page and object caching, Apache Ignite for real-time processing of large amounts of data, and VoltDB for real-time transactional operations with high performance. These databases are widely used in web development, financial services, and data analytics for fast access to information.

Web automation is the use of programs and scripts to automate routine actions on websites. It can include automatically filling out forms, interacting

with pages, retrieving and processing data, and other actions. Web automation can be useful for testing web applications, automating routine tasks, or even creating web bots.

Selenium is one of the most popular web automation libraries that allows users to control web browsers (such as Chrome, Firefox, Edge) from code, perform actions, input and read data from web pages. It is a powerful toolkit for web browser automation that allows users to perform various tasks on web pages, such as automated testing, web scraping, automated interaction with web elements, and more. Selenium supports various programming languages such as Python, Java, C #, and has various tools such as Selenium WebDriver for controlling different browsers and Selenium IDE for recording and replaying actions on web pages. Flexibility and extensive capabilities make Selenium a popular tool for automating web applications.

Puppeteer is a Node.js web automation library developed by the Google Chrome team that allows users to interact with web pages, perform actions, and retrieve data from web pages.

The technologies and software methods discussed play an important role in providing access to information from the web, analyzing it, and using it for various purposes. It is important to adhere to ethical and legal norms when using these methods, as unscrupulous use may violate the rights of data owners or website owners.

2.5. Machine-readable data and APIs

Machine-readable data and API are two concepts that are closely related to the exchange of data between computer systems and programs.

Machine-readable data refers to data that is structured and organized in a way that is understandable to computers and programs. The basic idea is that the data is formatted so that computers can automatically analyze and process it without the need for manual interpretation.

Examples of machine-readable data include:

- data in JSON, XML, CSV format, which has a clear structure and field designations;
- data presented in the form of tables or databases with known schemas;
- data using standardized codes, such as ISO country codes.

Machine-readable data is used to exchange information between different programs, applications, and systems, and it helps automate data processing and analysis processes.

API (Application Programming Interface) is a set of rules and protocols that allow different programs and systems to interact with each other. An API defines how programs can exchange data and perform functions with each other. An API can include different types of queries, methods, and parameters that programs can use to exchange data.

Examples of APIs include:

- Web APIs – web services that allow applications to interact using standard HTTP protocols, such as RESTful APIs and GraphQL APIs;
- libraries of program code that provide certain functionality that can be called from other programs;
- Operating system APIs, which allow programs to interact with various aspects of the operating system, such as working with files, networks, processes, and so on.

APIs allow developers to create applications that can use the functionality of other applications or services without having to know the inner workings of those applications.

APIs are often used to enable interoperability between different big data analytics tools and services.

Many big data analytics systems, such as Hadoop, Spark, and databases, provide APIs to interact with them. This allows users to perform data loading, sampling, processing, and analysis operations.

Cloud computing services such as Amazon Web Services (AWS), Google Cloud, Microsoft Azure provide APIs for managing computing resources, creating and managing computing clusters used to process large amounts of data.

Social networks such as Twitter, Facebook, Instagram provide APIs to access public data and metadata. Analysts can use these APIs to study user behavior, analyze trends, and measure impact. A variety of software methods and tools are used to analyze data from social networks, including Facebook. Consider some of them.

Graph analysis of social networks is based on the study of relationships between users. It helps to establish which users communicate with each other,

who are influential, which groups are formed, etc. Examples of tools for analyzing social graphs are Gephi and Cytoscape.

Text analytics allows users to analyze text content such as posts, comments, and messages. Natural Language Processing (NLP) algorithms are used to detect sentiment, keywords, themes, and other characteristics of the text. Tools that help with this include NLTK, spaCy, and TextBlob.

Machine learning allows users to create models that can predict certain phenomena or patterns based on data. For example, we can use machine learning to predict the popularity of posts, classify users by interests, etc. Python language with Scikit-Learn and TensorFlow libraries can be used to develop such models.

It is important to visualize data to better understand patterns and trends. Visualization tools such as Matplotlib, Seaborn, D3.js allow to create graphs, charts, heat maps, etc.

Sentiment analysis assesses the emotional tone of textual content, such as comments and messages, and can indicate whether users are positive, negative, or neutral about certain topics. Sentiment analysis tools include Vader, AFINN.

Cluster analysis helps group users or objects based on common characteristics. For example, we can use cluster analysis to find groups of users with similar interests.

Influence and popularity analysis allows users to determine which posts, users, or topics are the most popular and influential. The number of likes, comments, reposts, etc. is analyzed.

Consider an example of analyzing text content from pages on the Facebook social network using the Python programming language and the Facebook-scraper library. In this example, we will analyze the latest posts from a public page.

To install the Facebook-scraper library, run the following command in the command line (terminal):

```
!pip install facebook-scraper
```

The provided code uses the `facebook_scraper` library to extract and analyze text from the latest posts of a public Facebook page. First, the URL of the public page is specified, and the `get_posts` function is used to fetch the last 10 posts from the page.

The text content of each post is then processed to count the frequency of individual words. To achieve this, the script splits the text of each post into words, normalizes them to lowercase, and excludes non-alphanumeric characters to avoid counting punctuation.

A dictionary is used to track the word counts, updating the count for each word as it appears. After processing all posts, the script sorts the dictionary by word frequency in descending order and extracts the top 10 most frequently mentioned words. Finally, it prints the top words along with their respective mention counts, providing a simple analysis of word usage in the page's recent posts. Example of data analysis:

```
from facebook_scraper import get_posts
# Public page URL
page_url = 'https://www.facebook.com/facebook'
# Collect the last 10 posts from the page
posts = get_posts(page_url, pages=10)
# Dictionary to store the number of mentions of each word
word_count = {}
# Let's go through each post
for post in posts:
# Get the post text
post_text = post['text']
# Split the text into individual words
words = post_text.split()
# Iterate over each word and update the counter
for word in words:
word = word.lower() # Convert to lowercase
if word.isalnum(): # Exclude punctuation
if word in word_count:
word_count[word] += 1
else:
word_count[word] = 1
# Display the top 10 words by number of mentions
sorted_words = sorted(word_count.items(), key=lambda x: x[1], reverse=True)
top_words = sorted_words[:10]
for word, count in top_words:
print(f'{word}: {count} mentions')
```

This code allows users to get the last 10 posts from a Facebook page, process the text of each post, count the number of mentions of each word, and display the top 10 words by number of mentions.

Consider an example of sentiment analysis of Facebook posts using the Facebook-scraper library and the TextBlob library for text sentiment analysis:

Installing libraries:

```
!pip install facebook-scraper textblob
```

The following code allows users to analyze the sentiment (positive, negative, neutral mood) of text posts on a Facebook page. Using the TextBlob library, the text of each post is analyzed for sentiment, and its field value (sentiment.polarity) helps determine the mood of the text. Depending on this mood, posts are classified as positive, negative, or neutral.

Sentiment analysis example:

```
from facebook_scraper import get_posts
from textblob import TextBlob
# Public page URL
page_url = 'https://www.facebook.com/facebook'
# Collect the last 10 posts from the page
posts = get_posts(page_url, pages=10)
# Sentiment analysis and counters for positive, negative and neutral posts
positive_count = 0
negative_count = 0
neutral_count = 0
# Let's go through each post
for post in posts:
# Get the post text
post_text = post['text']
# Using TextBlob for sentiment analysis
blob = TextBlob(post_text)
sentiment_score = blob.sentiment.polarity
# Determining the mood of the post
if sentiment_score > 0.2:
positive_count += 1
elif sentiment_score < -0.2:
negative_count += 1
else:
neutral_count += 1
# Output the analysis results
print(f'Positive posts: {positive_count}')
print(f'Negative posts: {negative_count}')
print(f'Neutral posts: {neutral count}')
```

Financial institutions provide APIs to access financial data such as stock prices, exchange rates, financial reports. This allows for market analysis, risk calculations, etc.

Geodata services such as Google Maps provide APIs to access geographic data, maps, and geolocation services. This can be used to identify places, analyze movements, and locate.

Some companies provide APIs to analyze user feedback and comments on public forums, product reviews, and social media. This helps determine public sentiment and reactions to news and products.

APIs can be used to collect, analyze, and visualize logs from various sources, such as websites, applications, and network equipment. This helps identify anomalies, problems, and improve performance.

Machine-readable data and APIs play an important role in the interaction between programs, applications, and systems, allowing for efficient and automated exchange of information and functionality.

2.6. Semantic annotation recognition

Semantic annotation recognition for big data is the process of identifying and understanding meaningful elements or concepts in text, visual, or other types of data. Semantic annotation involves assigning labels or tags that indicate specific semantic or meaning aspects of the data. This process helps to take into account the context and meaning of the data for further processing, analysis, and interaction.

Consider the main stages of recognizing semantic annotations.

Stage 1. First, the data (text, video, images) is broken down into a small set of basic elements, such as words, sentences, phrases, objects, etc.

Token is a small piece of text that has a specific meaning or function within a program or programming language. Tokens can be keywords, operators, identifiers, literals, and other elements of program code. A token can be any element of program code that has a specific meaning to the interpreter or compiler. For example, in Python, the keyword `if`, the operators +, -, identifiers (variable names), literals (numbers, strings), and custom identifiers for functions and classes are all tokens.

Consider example to understand tokens in Python code:

```
y = 10  # Tokens: y (identifier), = (assignment operator), 10 (literal)
while y > 0:  # Tokens: while (keyword), y (identifier),
 > (comparison operator),
 0 (literal), : (colon)
    y -= 1  # Tokens: y (identifier), -= (assignment operator), 1 (literal)
    print(y)  # Tokens: print (built-in function), "(" (bracket),
    y (identifier), ")" (bracket)
```

In this example, each element of the code is broken down into tokens. For instance, `y` is an identifier representing a variable, `=` and `-=` are assignment operators, and 10 and 1 are literals that represent constant values. The keyword `while` defines a control flow structure, `while >` is a comparison operator that evaluates the condition.

The colon “:” signifies the beginning of the loop block, and `print` is a built-in function to output values. Each token has a distinct meaning and function, forming the building blocks of the program's logic.

Tokenization is the process of breaking text into separate parts – tokens. Tokenization is used to recognize and extract individual words, characters, phrases or other syntactic elements from a text document for further processing. Tokenization can be performed according to different rules, depending on the requirements of a specific task. For example, text can be broken into tokens by spaces, punctuation marks, regular expressions or using specialized algorithms for specific languages or programming languages.

After tokenization, the text can be processed, for example, for syntax analysis, keyword detection, entity recognition, or other analytical operations. Tokenization is an important part of many text processing and analytical systems, such as natural language processors, compilers, parsers, and others.

Stage 2. Definition of semantic features, identification of key concepts, events, objects, persons, etc. that have semantic meaning for a given context.

Stage 3. Marking or tagging, when each basic data element (for example, words in text or objects in an image) is assigned semantic labels or tags indicating their semantic characteristics.

Step 4. Building connections. The recognized semantic elements can be linked together to define relationships and context. For example, connections can be established between the concepts "author" and "book" in the text.

Stage 5. Semantic model creation. The resulting semantic annotations can be combined into semantic models or graphs that reflect the structure and relationships between semantic elements.

In text analysis, semantic annotations can help identify key topics, named entities (people, places, companies), and semantic relationships between different pieces of text.

Consider an example of using the SpaCy library to perform semantic text annotation in Python.

The following code uses NLTK (Natural Language Toolkit) to tokenize a sentence and annotate each word with its part of speech.

The `word_tokenize` function splits the text into individual words or tokens, such as "The," "quick," "brown," etc. Next, the `pos_tag` function assigns a part-of-speech tag to each token, such as `DT` (determiner), `JJ` (adjective), `NN` (noun), or `VBZ` (verb).

These tags provide grammatical information about the tokens, which is essential for understanding the structure and meaning of the text.

The output enables us to analyze the sentence grammatically, which can be applied in tasks like syntax analysis, sentiment analysis, or text parsing.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag

# Sample text
text = "The quick brown fox jumps over the lazy dog."

# Tokenize the text into words
tokens = word_tokenize(text)

# Perform part-of-speech tagging
tagged_tokens = pos_tag(tokens)

# Display the tokens and their corresponding POS tags
for word, tag in tagged_tokens:
    print(f"{word}: {tag}")
```

Semantic annotations can be used to analyze the content and semantic relationships between users, posts, hashtags, etc. in social networks.

In image analysis, semantic annotations can help identify objects in an image, their characteristics, and interactions.

In video analysis, semantic annotations can indicate key events, objects, and changes in a video sequence.

Identifying semantic annotations allows users to determine the context and meaning of data, which is key for further analysis, obtaining valuable conclusions, and making further decisions.

2.7. Web page analysis using computer vision

Computer vision is a branch of artificial intelligence and image processing that develops methods and technologies for recognizing, understanding, and interpreting images and videos in order to automate processes that would normally require human vision. This field includes tasks such as face recognition, object classification, motion detection, object tracking, distance measurement, and many others. Computer vision is used in various fields, including medicine, automotive, security, agricultural technology, multimedia, and many others.

One of the main tasks of computer vision is to create algorithms and machine learning models that can "learn" to recognize and analyze images using data sets. In general, computer vision aims to help computers understand the information contained in images and videos, just as the human eye does. Computer vision web crawlers are tools and technologies that use computer vision to analyze and understand the content of web pages, images, and videos based on the identification of objects, regions of interest, semantic elements, and the interactions between them. They recognize objects, understand their characteristics, and relationships, which helps analyze the content of web pages and image elements at a higher level of detail.

Consider the main components and stages of web page analysis using computer vision.

Stage 1. Analysis begins with selecting a web page or image for further analysis, for example, reading the HTML code of the page or downloading the image from the appropriate source.

Step 2. Images may require some pre-processing, such as downscaling, highlighting, or noise removal, to allow for more accurate analysis.

Step 3. Using computer vision techniques, the analyzer identifies objects in an image or web page. These can be objects of various types, such as faces, vehicles, products, text blocks, etc.

Stage 4. Classification and recognition. Once objects are detected, analyzers can classify them based on learned patterns, recognizing their types, categories, or characteristics. For example, recognizing a human face.

Step 5. Segmentation and extraction. The analyzers identify regions of interest (ROIs) and extract them for further analysis. Sometimes it is important to analyze the interactions between objects in an image or page. This may involve recognizing objects that interact with each other.

Stage 6. For web pages or images containing text, parsers can recognize and interpret the text content, including extracting named entities, keywords, etc.

Stage 7. After analysis, analyzers can create inferences about the content, meaning, and interaction of objects on a web page or in an image.

Analyzers can be used to identify the content of photographs on websites, for example, identifying product information in images. Analyzers also allow to identify key events or scenes in videos, which can be used to analyze and evaluate video content (Fig. 2.4).

Websites can be analyzed to identify types of content, images, or links for further processing and analysis.

Fig. 2.4. Examples of web page image analysis using computer vision

Web pages can be analyzed to detect errors such as broken links or misplaced objects. Web page data analysis using computer vision uses programming languages such as Python with the OpenCV, TensorFlow, and Keras libraries, Java with JavaCV, JavaScript with TensorFlow.js, C++ with OpenCV, R with ImageMagick, and MATLAB with the Computer Vision Toolbox, which provide tools for image processing, object recognition, and web page analysis. Python is often considered the best programming language for web page analysis tasks using computer vision due to its simplicity, extensive library support, and active community. Libraries such as OpenCV, TensorFlow, and Keras provide robust tools for image processing, object detection, and machine learning, enabling efficient analysis of web pages to detect errors like broken links or misplaced objects. Python's flexibility and integration with other tools, such as Selenium for web scraping and automation, make it a powerful choice for end-to-end web page analysis pipelines. Its vast ecosystem, combined with ease of use, positions Python as the top choice for both beginners and experienced developers tackling computer vision tasks. Some analyzers use an AI voice assistant that can analyze the content of a web page and answer the user's questions.

Computer vision-based web analyzers open up a wide range of possibilities for analyzing content that includes large amounts of data. Facial recognition systems are able to automatically identify people's faces in images or videos. This is used in security systems, photo archives, social networks, and many other areas.

Recognizing plants, fields, and other details in satellite images can help in managing agricultural areas. Visual inspection systems are used in production to detect defects in products or materials, such as cracks, scratches, irregularities, etc. (Fig. 2.5). These systems leverage computer vision technologies to analyze images or video streams in real time, ensuring consistent quality control and reducing the need for manual inspections.

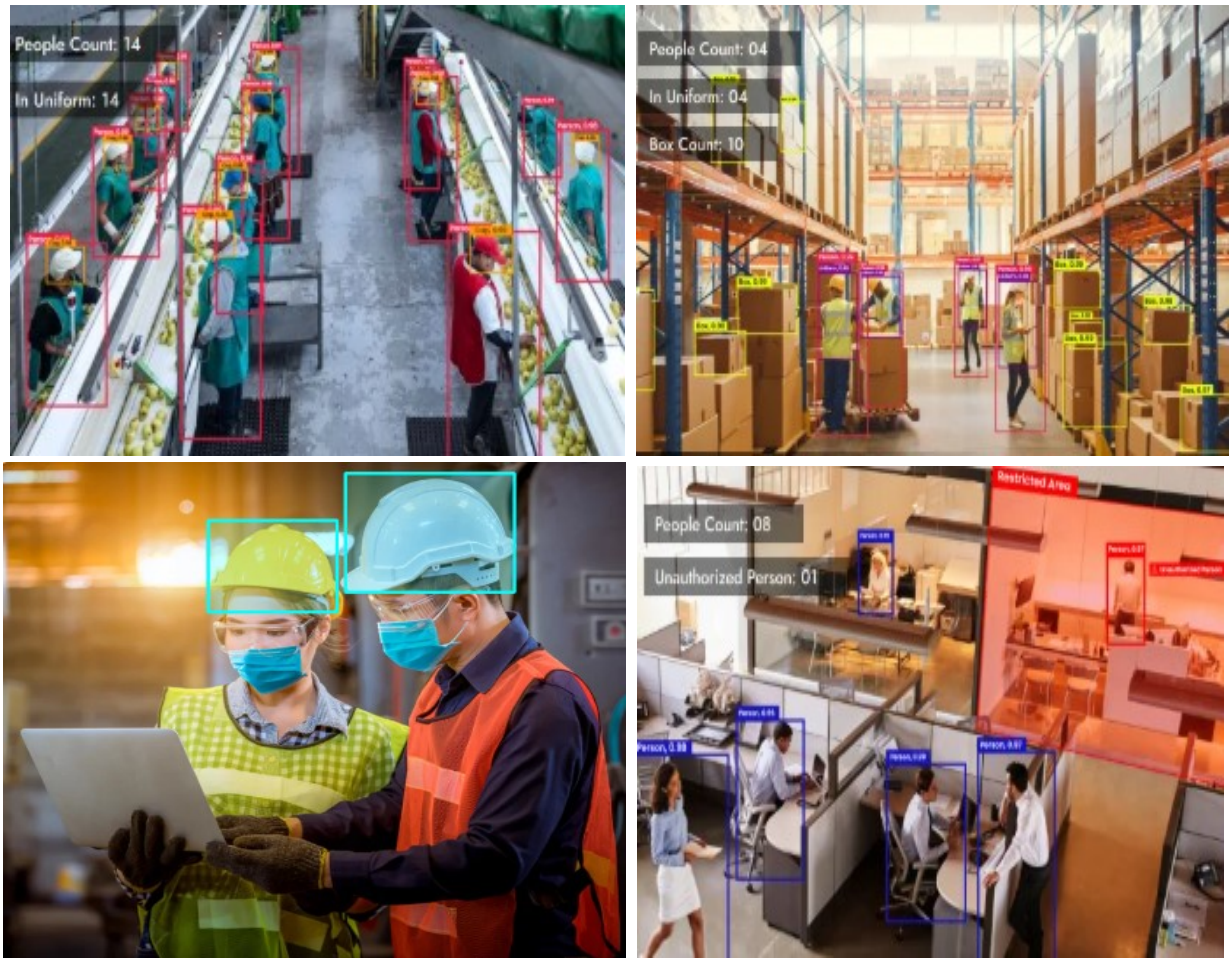


Fig. 2.5. The use of computer vision in manufacturing and in surveillance and security systems

In financial or business analysis, systems can help detect anomalies, fraudulent activity, or unusual changes in large data sets. Traffic sign recognition technologies can detect various types of signs in images or videos from dashcams and help drivers comply with traffic rules. Real-time object detection is an important feature for the safety and navigation of autonomous cars, including other vehicles.

Control questions and tasks for topic 2

List of theoretical questions

1. What are the main stages in the big data analytics lifecycle and why are they important?
2. Why is it important to load data into the storage system after preparing it in the previous stages?
3. What are the possible benefits of monitoring and optimizing results in the "monitoring and optimization" phase?
4. What tools or platforms can be used to perform programmatic ETL methods in the context of big data?
5. What is the role of the Python programming language for big data analytics?
6. What is web scraping and in what areas is it used?
7. What are the main software methods used for web crawling, and what is their purpose?
8. How does indexing help organize and store large amounts of collected data ?
9. How does an API simplify access to data from the Internet and why is it important for big data analysis?
10. What are HTML parsers and how do they help web scraping?
11. How is DOM analysis used to analyze the structure of web pages?
12. How does semantic annotation recognition facilitate data analysis on web pages?
13. How is computer vision used to analyze web pages?
14. What are the benefits of web automation for performing routine tasks on web services?
15. What programming languages and libraries are used to implement web scraping, web crawling, and other big data analysis technologies?

List of tasks for independent work

Practical task. Extracting image URLs from a web page.

Step 1. Execute the following Python code to retrieve the HTML content of a web page using the requests library and extract all image URLs from the page using the BeautifulSoup library.

```
import requests
from bs4 import BeautifulSoup
```

```
    # URL of the page to scrape
    url = 'https://www.example.com'
    # Make an HTTP request to the page and get the HTML
content
    response = requests.get(url)
    html_content = response.text
    # Use BeautifulSoup to parse the HTML
    soup = BeautifulSoup(html_content, 'html.parser')
    # Find all image tags in the page
    image_tags = soup.find_all('img')
    # Extract and print the URLs of the images
    for img in image_tags:
        img_url = img.get('src')
        if img_url:
            print('Image URL:', img_url)
```

Before implementing this code, ensure that we comply with ethical standards and website policies. Some websites explicitly prohibit automated scraping through their robots.txt file or terms of service. Always respect these guidelines and avoid overloading the server with excessive requests.

Step 2. Extracting product information using Python and Scrapy. Execute the following Python code with the Scrapy framework to collect product information (names and prices) from an e-commerce website. If Scrapy is not already installed, we can install it using:

```
! pip install scrapy
```

Create a Scrapy project with the command (replace product_scraper with any project name):

```
scrapy startproject product_scraper
```

Define the data structure we want to collect. Open the file product_scraper/items.py and add the following:

```
import scrapy
class ProductItem(scrapy.Item):
    name = scrapy.Field()
    price = scrapy.Field()
```

Create a new Spider for extracting product information. Open product_scraper/spiders/product_spider.py and add the following:

```
import scrapy
```

```
    from product_scraper.items import ProductItem
    class ProductSpider(scrapy.Spider):
        name = 'product'
        start_urls = ['https://example-ecommerce-
site.com']
        def parse(self, response):
            for product in response.css('div.product'):
                item = ProductItem()
                item['name'] = product.css('h3.product-
name::text').get()
                item['price'] =
product.css('span.price::text').get()
                yield item
```

Run the spider with the following command to save the results in a JSON file:

```
scrapy crawl product -o products.json
```

This Scrapy setup defines a ProductSpider that crawls an e-commerce site, extracts product names and prices from the HTML structure, and saves the results into a JSON file called products.json. Each product's information is stored in a structured format using the ProductItem class. As with all web scraping tasks, ensure adhere to the website's terms of use and ethical guidelines, including checking the robots.txt file and avoiding excessive requests. This helps maintain compliance and avoids unnecessary server load.

Step 3. Utilize the Shapely library for spatial queries by creating and working with geometric objects, such as polygons and points, to analyze spatial relationships. Installing the library:

```
!pip install shapely
```

Example of working with geometric objects and spatial queries:

```
from shapely.geometry import Point, Polygon
# Define a polygon (e.g., a rectangular area)
polygon = Polygon([(x_min, y_min), (x_min, y_max),
(x_max, y_max), (x_max, y_min), (x_min, y_min)])
# Create a list of points (e.g., sample geographic
data)
points = [
    Point(1, 1),
```

```
        Point(2, 2),
        Point(3, 3),
        Point(4, 4)
    ]
    # Filter points that fall within the polygon
    points_in_polygon = [point for point in points if
polygon.contains(point)]
    # Output results
    print("Total points:", len(points))
    print("Points inside the polygon:",
len(points_in_polygon))
    for idx, point in enumerate(points_in_polygon, 1):
        print(f"Point {idx}: {point}")
```

This code uses the Shapely library to create geometric objects and perform spatial queries. A polygon is defined to represent a bounding area, and a list of points is analyzed to determine which ones fall inside the polygon. The code efficiently identifies and lists points that meet the spatial criteria. Make sure to replace `x_min, y_min, x_max`, and `y_max` with actual coordinates. This approach is suitable for analyzing spatial relationships in small datasets or when precise geometric operations are required.

List of used and recommended literature

2.1. Gheorghe M., Mihai F.-C., Dârdală M. Modern techniques of webscraping for data scientists. International Journal of User-System Interaction, 2018 , 11. P. 63–75.

2.2. Alrashed T., Almahmoud J., Zhang AX, Karger DR ScrAPIr:Making Web Data APIs Accessible to End Users. Proceedings of the 2020 Conference on Human Factors in Computing Systems , 2020 . R. 1–12. NewYork, NY, USA: Association for Computing Machinery. DOI:10.1145/3313831.3376691 .

2.3. Fiesler C., Beard N., Keegan BC No Robots, Spiders, or Scrapers: Legal and Ethical Regulation of Data Collection Methods in Social Media Terms of Service. Proceedings of the International AAAI Conference on Web and Social Media, 2020 , 14(1). R. 187–196.

Topic 3. Virtualization technologies for big data analytics

Virtualization is a concept and technology that allows users to access, combine, and analyze data from different sources without having to physically move, copy, or merge the data . The primary role of virtualization for big data analytics is to make the process of processing and analyzing data more efficient, convenient, and cost-effective.

3.1. The role of virtualization for big data analytics

Virtualization allows users to create connections from different data sources, regardless of their physical location, and present them as a single, collated source for analysis.

Virtualization is the creation of a virtual version of a physical IT resource, such as a server or desktop.

Virtual machine (VM) is software or hardware that emulates a computer and allows programs and processes that would normally run on a real computer or server to run. A virtual machine is created using virtualization software or hardware (in the case of hardware virtualization) and can have its own operating systems, drivers, and applications. A virtual machine is isolated from other virtual machines, allowing it to operate independently. Virtual machines allow to efficiently use server resources and create separate environments for deploying and running various programs and processes.

A virtual machine is created on a physical host device and behaves just like a physical device, and can run different OSes from the host. Virtual machines are also called guest machines. Multiple guest virtual machines can run on a host device. For example, we can create a virtual server running Windows on a physical device running Linux. A virtual machine is a fully functional Windows server that uses the resources of the host. We can run multiple Windows servers, Linux servers, etc. on a single host server.

Physically moving large amounts of data from one location to another can be cumbersome and expensive. Virtualization allows users to work with data where it is stored, reducing the need for unnecessary movements. A virtual machine operates as an isolated environment that contains the operating system and applications (Apps). Virtualization allows a virtual machine to simulate hardware resources such as the processor (CPU), memory, disk, network and video cards, peripherals (e.g., mouse, keyboard), and other components (e.g., CD/DVD). The operating system and

applications on the virtual machine interact with these hardware resources through a software layer that provides flexibility and isolation from the physical infrastructure (Fig. 3.1).

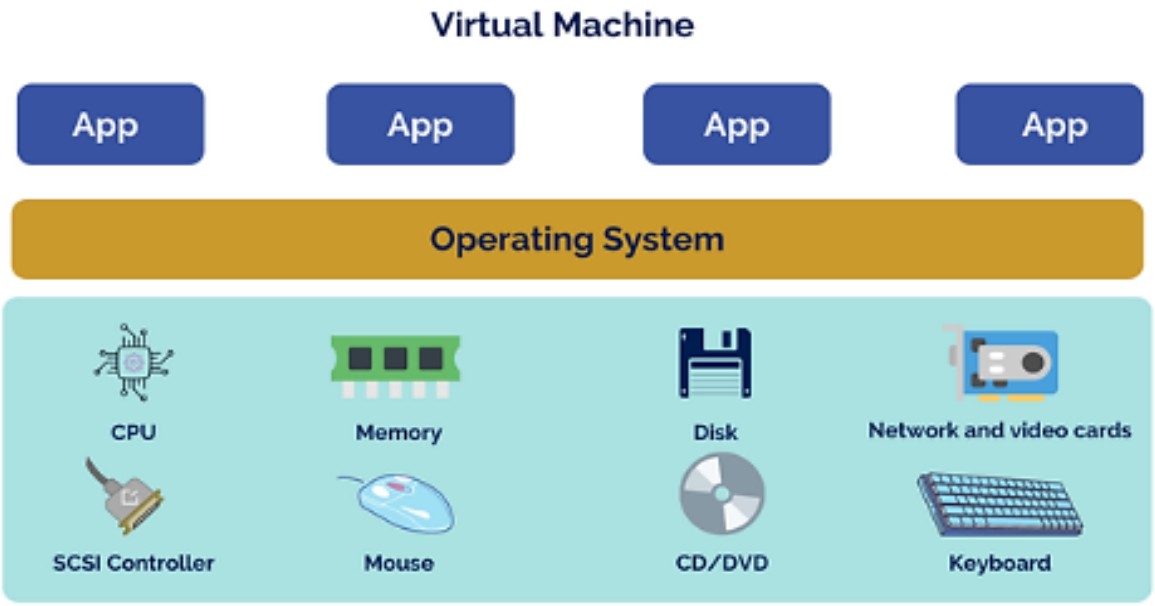


Fig. 3.1. Hardware virtualization applied to physical computer

Data sources can use different formats, structures, and schemas. Virtualization can reflect these differences and provide a single, standardized access to data.

Virtualization allows users to forward queries to data sources and process them where possible, rather than loading all the data into an analytics platform. This can help distribute the load and reduce the strain on storage systems.

Virtualization can also provide real-time or near-real-time access to data, allowing analysts to work with up-to-date information and respond quickly to events.

Virtualization can help avoid the cost of physically creating and maintaining copies of data. This can reduce the time to deploy new analytical solutions and the overall cost of data processing. In addition, virtualization allows more analysts and users to work with data simultaneously without the need for increased computing resources.

3.2. Virtualization technologies for big data analytics

Virtualization technologies are key components for creating virtual environments that allow to work with resources, data, and software in a more efficient and abstracted way. Consider some of the main virtualization technologies used in the context of big data analytics.

Data virtualization allows analysts to access data from disparate sources, such as databases and APIs, as a single, collated source. This helps avoid the need to physically merge data and deploy replicas, simplifying the analysis process.

Moving analytical tasks to cloud environments allows users to dynamically scale computing resources depending on needs, reducing costs and ensuring high availability.

Application virtualization allows users to analyze data using virtual applications running on servers, providing deployment and management.

Virtualized analytics environment – creating virtual environments for performing analytical work, which allows for timely provision of resources to analysts, even if their tasks require large computing power.

Virtual desktop infrastructure (VDI) enables analysts to work in virtual desktops, providing access to analytical tools from different devices.

Real-time data virtualization allows users to analyze data that arrives in real time by combining and transforming it without the need for physical movement or replication.

One example of the use of real-time data virtualization can be in the field of finance, where a company receives real-time financial transaction data from various sources such as banking systems, payment platforms, investment exchanges, etc. Instead of copying or moving this data to a central data repository, it can be virtualized, allowing it to be analyzed in real time without the need to physically move or replicate it. Such virtualization allows companies to receive updated information faster and provides more up-to-date analysis of financial data for making quick and effective decisions.

3.3. Layers of abstraction

Abstraction layers for big data virtualization define the levels of abstraction and interfaces between different components and resources that enable efficient processing, access, and analysis of large amounts of data.

Each layer is responsible for specific functionality and isolates the complexity of lower layers from the user.

Physical layer situated at the lowest level, it covers hardware such as servers, network devices, storage devices, etc. This layer provides the real resources that the higher layers will interact with.

Infrastructure layer includes virtualization systems such as hypervisors, container platforms, cloud computing platforms. It is responsible for resources and their allocation to virtual environments that allow to run and manage virtual machines, containers, etc.

Abstraction layer – at this level, abstraction of various data sources, infrastructure and services is implemented using data virtualization technologies, networks, resources and other components. Abstraction allows resources to be managed from a higher level, providing users with simplified access and management.

Data and applications layer – this layer houses data, applications, and analytical tools. This layer includes database management systems, data analysis tools, virtual applications, and components used to analyze and interact with data.

User layer or user interface is located at the highest level, providing access to analytical capabilities and data processing results. It can be a web interface, desktop applications, or mobile applications.

With each higher layer of abstraction, the details and complexity of the lower levels are isolated from the user. This allows for greater convenience, efficiency, and flexibility in processing and analyzing large amounts of data. Users can work with data and analytical tools without having to understand the technical details of the lower levels in detail.

3.4. Hypervisors

Hypervisor is software or hardware that allows users to create and manage virtual machines on a physical server. Hypervisors allow to divide the resources of a physical server among virtual machines and provide isolation between them. They control access to resources such as processor time, memory, network and storage resources, and allow many virtual machines to share the physical resources of the server.

Hypervisors can be classified as either type 1 (native or barrier) or type 2 (embedded or guest), depending on whether they run directly on the

hardware level or are installed on the host operating system. Both types have their advantages and uses, but for large amounts of data, type 1 hypervisors are more commonly used. Hypervisors are an important technology component for big data virtualization because they allow to create and manage virtual machines on a single physical server. This simplifies deployment, resource management, and provides flexibility in handling large amounts of data.

Type 1 (native) hypervisor is installed directly on the physical server and runs without the control of the operating system. It provides higher performance and lower overhead because it works directly with the hardware. It is most often used for server virtualization of large amounts of data. Examples: VMware vSphere/ESXi, Microsoft Hyper-V, KVM (Kernel-based Virtual Machine) and Xen.

Type 2 hypervisor (guest) is software that is installed on a host operating system and allows users to create and manage virtual machines. It runs at the operating system level and uses the resources and functionality of the host OS. A type 2 hypervisor is usually easier to install and configure, but may be less efficient than a type 1 hypervisor.

Hypervisors allow multiple virtual machines to run on a single physical server, sharing its compute resources and memory . This ensures efficient use of hardware and reduces costs. Each virtual machine is isolated from the others, which ensures the stability and security of analytical work. Hypervisors allow to scale virtual machines by adding or removing resources as needed.

Hypervisors provide interfaces for managing resources such as compute, memory, and storage. This allows users to dynamically change the resources allocated to virtual machines. Hypervisors allow to remotely start, stop, restart, and manage virtual machines. Hypervisors typically provide interfaces for monitoring the status of virtual machines, their resources, and performance. They provide the ability to configure automatic scaling of resources depending on data processing needs.

Hypervisors allow to virtualize data storage, create virtual disks, and combine them into storage systems. Hypervisors simplify resource management, increase hardware utilization, and ensure stability and security during data processing and analysis.

3.5. Virtualization architecture

Virtualization works by sharing the resources of a host device among virtual machines. This is done through software known as a hypervisor. The hypervisor acts as an intermediary between the virtual machines and the host device. When a virtual machine is created, the hypervisor allocates some of the host resources to it, such as CPU and memory. If the virtual machine requires more resources while running, the hypervisor allocates more host resources to it.

The host device's disk storage is emulated as virtual storage on the virtual machine. When the virtual machine tries to find files in its virtual storage, the hypervisor translates that request to find files in the actual physical storage on the host.

A hypervisor can manage multiple virtual machines by sharing host resources. Virtual machines can be moved from one host to another and continue to operate as before.

The virtualization architecture is based on the host hardware, including CPU, memory, and disk space. In the case of a type 2 hypervisor, the hypervisor does not require any host OS and runs directly on top of the host hardware. In the case of a type 1 hypervisor, the hypervisor runs on top of the host OS (Figure 3.2). The hypervisor receives resources from lower layers and divides them into virtual instances. Guest OSes run on top of the hypervisor and are unaware of each other, allowing them to behave as independent devices.

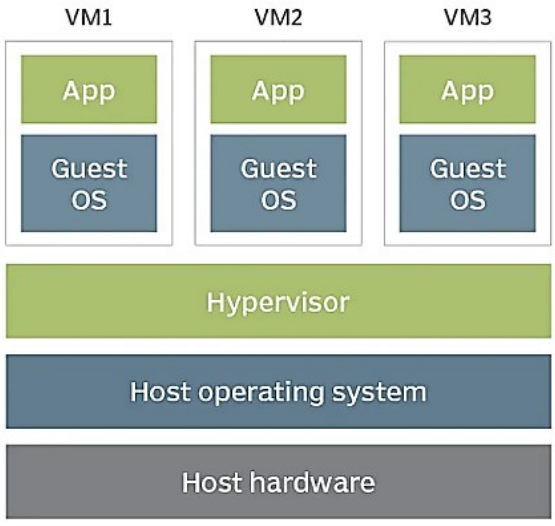


Fig. 3.2. Virtualization architecture

Virtualized devices are more versatile and efficient than their traditional physical counterparts. Consider some of the advantages of virtualization.

Servers are designed to deliver peak performance , but for most of their lifespan they operate at much lower loads. When multiple virtual machines are running on a host, they share the host's resources as needed. This results in efficient use of disk space and resources.

Since each virtual machine is an independent component, we can run multiple operating systems, software, and applications for different virtual machines from a single host device. Instead of purchasing multiple servers to run different software, a single virtualized cluster will suffice.

A snapshot of a virtual machine is a copy of that virtual machine at that point in time. We can take snapshots of a virtual machine as backups, run snapshots in another location, or restore them as needed. This is useful before performing risky operations. If something goes wrong, we can restore the previous version.

Virtual machine migration allows users to move virtual machines from one host to another and restore them to their original state. There are many uses for this. We can enable a failover system, in which snapshots of the virtual machine are frequently backed up to another host. If the primary host fails, the virtual machine can seamlessly switch to the secondary host.

A virtual machine is an isolated system that can be used to test software and run suspicious programs. Virtual machines can be started or stopped, run tests, and shut down as needed. With virtualization, we can access the host server from another device and access the virtual machines. Users from different locations can use their personal devices to access the host server. The host server connects the virtual machines to their devices over a network. The user's devices can be cheaper and use fewer resources.

Server virtualization is a widely used type of virtualization where a full-fledged server is emulated for use on a host device. Virtual servers perform the same roles as physical servers. They host websites and applications, act as databases, and perform computations. Multiple virtual servers run on a single physical server and share system resources. Servers can also be implemented as a load-balanced array, where virtual machines are shared across different host devices.

Server virtualization is offered by vendors such as VMware, Nutanix, Xen, and Hyper-V. Organizations can consolidate their server architectures

by replacing multiple physical servers with a virtualized cluster. A typical virtualized environment includes a host server, virtual servers, and virtual data storage. Virtual sprawl is a situation where virtual servers use too many host resources. The hypervisor cannot allocate enough resources to all virtual machines, and other virtual servers experience performance issues due to resource shortages. In network virtualization, network functionality (routing, switching, and data transfer) is virtualized using hardware and software components. There are two types of network virtualization: external and internal. When external network virtualization is A physical network is divided into multiple virtual networks. An example is a virtual local area network (VLAN). Multiple VLANs share the same network devices, switches, routers, controllers, and access points. But they are actually separated and act as different networks. Another example of this type is a virtual private network (VPN). A VPN provides access to private local area networks (LANs) over the Internet using tunneling and encryption protocols. In this case, the same local area network exists on network devices from multiple locations. Internal network virtualization simulates a network on a single host device using software. A server can contain multiple containers and other virtual instances. The network connections between them are simulated to form a virtual network. Virtual versions of hardware network devices such as routers and switches are created.

Software- defined networking (SDN) is another network virtualization program designed to speed up and improve the efficiency of network operations. In a traditional physical network, network devices such as routers and switches perform management operations as well as data operations separately. Control operations include creating a routing table, updating the topology, shaping traffic, etc. Data operations include receiving and transmitting data packets. In an SDN system, the SDN controller takes care of control operations, while network devices only handle data transmission. Administrators can use the controller to shape traffic according to their requirements. In an SDN architecture, control applications (Control App) interact with the network through an API that connects to the Network Operating System (Network Operating System). The Network Operating System provides control of hardware devices, such as programmable switches (Programmable Switch), through another API. At the switch level, there is a switch operating system (Switch OS) and a hardware component that includes

Merchant Silicon, a programmable hardware platform for performing network functions (Fig. 3.3). This provides modularity, programmability, and flexibility in network management.

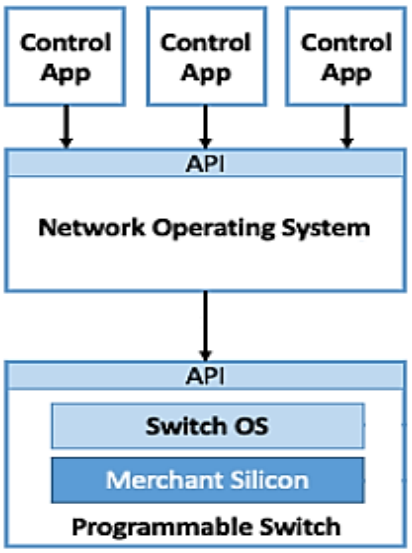


Fig. 3.3. General architecture of the SDN software stack

Storage virtualization involves emulating logical storage units on top of physical storage. In storage virtualization, individual storage units, such as disks, are often combined to create a storage network. This network is then managed by software. Virtual storage units can be created from the storage network. Virtualization makes storage systems faster, more resilient, and more reliable. Storage virtualization has advantages over conventional storage systems. Because physical storage systems are pooled together, data can be moved from one location to another, while the virtual data block remains the same. Similar to server virtualization, storage virtualization ensures efficient storage utilization because all storage is pooled together. When storage is running low, more storage devices can be added to increase capacity. Instead of dealing with multiple storage devices of varying capacities located in different locations, unified, clearly categorized virtual storage devices are handled.

Redundant array of independent disks (RAID) is a storage virtualization technology in which disks are combined to form a virtual storage unit. Several RAID levels are available with varying levels of speed, capacity, fault tolerance, and reliability.

In RAID 0 (Striping), data is distributed (striped) across two or more

disks. This increases read and write speeds because operations can be performed in parallel on multiple disks. This is used for high-speed workstations where performance is important and data storage reliability is not critical. The disadvantage is the lack of data redundancy. Losing one disk results in losing all data.

In RAID 1 (Mirroring), data is duplicated (mirrored) across two or more disks. Each disk contains a complete copy of the data. Used in systems where data loss is critical, such as database servers or file servers. The advantage is high reliability because the data is stored on multiple disks. If one disk fails, the data can be recovered from another disk. The disadvantage is the need for double the amount of disk space.

In RAID 5 (Striping with Parity), data and parity are distributed across three or more disks. The parity is used to recover data in the event of a disk failure. Used in servers and storage systems where a balance between performance and reliability is required. The advantage is efficient use of disk space, as well as a balance between speed, reliability, and volume. The disadvantage is the loss of performance when writing data due to the need to calculate checksums. Recovering data in the event of a disk failure can take a long time.

RAID 6 (Striping with Double Parity) is similar to RAID 5, but with an additional checksum. The data and two checksums are distributed across four or more disks. The advantage is higher reliability compared to RAID 5, as it allows data recovery in the event of a failure of two disks simultaneously. The disadvantage is reduced write performance due to double calculation of checksums and the need for more disks. Used in mission-critical data storage systems where high reliability is required.

RAID 10 (Combination of RAID 1 and RAID 0) combines the advantages of RAID 1 and RAID 0. Data is mirrored and distributed between disks (striping and mirroring). The advantage is high read/write speed and high reliability. The disadvantage is the need for a large amount of disk space (minimum four disks). Used in database servers and other high-performance systems where speed and reliability are required.

RAID 1 or RAID 10 are used to provide high reliability and fast data access. RAID 5 or RAID 6 are used for efficient use of space and reliable storage of large amounts of data.

RAID 0 is used for high-speed recording and playback of video

materials. RAID 6 is used to provide high reliability of data storage. The choice of RAID level depends on specific requirements for performance, reliability, and disk space.

A storage area network (SAN) is a dedicated network that connects storage devices, servers, and network devices. A SAN is not connected to a local area network, but has its own set of SAN switches to connect the storage devices. Access to a SAN is through a server, known as the host layer. SAN uses Fibre Channel or iSCSI protocols to provide fast and reliable data access.

The main advantage of SAN is the ability to centrally manage data storage, which simplifies administration, increases resource efficiency, and provides high performance. This is especially important for large enterprises where critical business applications require fast access to large amounts of data. SAN supports data backup and recovery in the event of a failure, ensuring continuous system operation. Another important feature of SAN is the ability to create data snapshots and clone them for backup or testing. Storage networks are used in environments with large amounts of transactional data, such as databases, virtualization systems, financial transaction processing, and to support cloud solutions. SAN allows users to scale storage as needed by adding new disk arrays or storage systems without interrupting the operation of existing servers and applications, providing high availability, scalability, and data storage performance required for modern IT infrastructures.

Unified computing system (UCS) is a converged data center architecture that combines server virtualization, network virtualization, and storage virtualization. UCS typically has a powered chassis with a cooling unit and slots for connecting blade and rack servers (Fig. 3.4).

Fig. 3.4. Cisco Unified Computing System 5108

Virtual hosts can be connected to server slots and interconnected by a

virtualized network. UCS can be connected to a SAN to provide virtualized storage capabilities. Unified Computing System allows data centers to be highly converged and efficient by significantly reducing cabling and space usage. UCS have management software that can monitor and control their operations. Most network infrastructures around the world have one or more types of virtualization technologies.

Virtualization has significant benefits for big data analytics tasks, providing flexibility, scalability, and efficient use of resources. With virtualization, compute, network, and storage resources can be dynamically allocated or scaled according to the needs of the analytics task, which is especially important for working with uneven or variable workloads.

Isolated virtual environments allow to run multiple different analytics platforms or frameworks (e.g. Apache Hadoop, Spark) on the same physical infrastructure, minimizing conflicts and simplifying management. In addition, virtualization increases the reliability and fault tolerance of analytics systems: in the event of a failure in one virtual machine, the others remain operational, and resources can be quickly transferred to other servers.

The use of virtual machine templates simplifies the deployment of analytics clusters, reducing setup and startup time. Virtualization also contributes to the efficient use of hardware, allowing to reduce infrastructure costs, as different analytical tasks can share physical resources without the need to purchase additional hardware. Due to these advantages, virtualization is a key technology for ensuring high performance, cost-effectiveness, and flexibility in big data analytics. Virtualization has some drawbacks that can cause problems for IT teams. The main ones are the complexity of virtual systems and the lack of visibility into virtual components. Unlike physical devices, which can be tested individually, virtual machines are logical units that have no physical differences.

Control questions and tasks for topic 3

List of theoretical questions

1. What role does virtualization play in improving the efficiency of big data analytics?
2. How does virtualization help in scaling infrastructure for big data analytics?
3. What are the advantages and disadvantages of virtualization for

big data storage and processing?

4. How does virtualization affect data security in big data analytics systems?

5. Name the main virtualization technologies used for big data analytics.

6. How does using containers like Docker help with big data analytics?

7. What are abstraction layers in the context of virtualization and how do they help in big data management?

8. Explain the role of a hypervisor in a virtualized infrastructure for big data analytics.

9. What are the types of hypervisors and what are their main differences?

10. How does virtualization architecture help optimize resources for big data processing?

11. What components does a virtualization architecture for big data analytics include?

12. How do Type 1 hypervisors differ from Type 2 hypervisors in the context of big data analytics?

13. What are the benefits of using virtualization technology in cloud platforms for big data analytics?

14. How does virtualization affect the performance of big data analytics systems?

15. Explain the concept of "virtual machines" and their importance in big data analytics.

List of tasks for independent work

Practical task 1. Deploy a cloud-based virtualization platform.

• Install a cloud management platform (e.g., OpenStack or Proxmox) on a dedicated server or virtual machine.

• Create a virtual data center within the platform, defining resource pools (e.g., CPU, memory, storage).

• Set up virtual machines (VMs): deploy at least two VMs with different operating systems (e.g., Ubuntu for backend services and Windows Server for front-end tasks).

• Configure a virtual network within the platform to ensure connectivity between the VMs.

• Assign storage volumes to each VM for additional data storage.

• Monitor resource usage through the platform's dashboard and generate reports on CPU, memory, and network utilization.

• Document the setup process and include screenshots of the platform interface, VM configurations, and resource usage reports.

Practical task 2. Containerized application deployment in a multi-node environment.

• Install a cloud management platform (e.g., OpenStack or Proxmox) on a dedicated server or virtual machine.

• Create a virtual data center within the platform, defining resource pools (e.g., CPU, memory, storage).

• Set up virtual machines (VMs): deploy at least two VMs with different operating systems (e.g., Ubuntu for backend services and Windows Server for front-end tasks).

• Configure a virtual network within the platform to ensure connectivity between the VMs.

• Assign storage volumes to each VM for additional data storage.

• Monitor resource usage through the platform's dashboard and generate reports on CPU, memory, and network utilization.

• Document the setup process and include screenshots of the platform interface, VM configurations, and resource usage reports.

List of used and recommended literature

3.1. Platform Virtualization – A Creation of Virtual Machines. [Electronic resource]. – Access mode: https://www.lumenci.com/post/platform-virtualization

3.2. Virtualization via Virtual Machines. [Electronic resource]. – Access mode: https://insights.sei.cmu.edu/blog/virtualization-via-virtual-machines/

3.3. What is virtualization. [Electronic resource]. – Access mode: https://www.manageengine.com/network-monitoring/what-is-virtualization.html

3.4. Cisco Unified Computing System. [Electronic resource]. – Access mode: https://www.datapac.com/services/network-storage-virtualisation/servers/cisco-ucs/

Topic 4. Container technology for executing program code

Containerized server-side code execution is an approach to deploying and executing applications that is based on the use of containers. A container is an isolated environment that contains all the necessary components to run an application, including code, libraries, system tools, and settings. Containers provide isolation of applications from the host operating system and from each other, which increases security and stability.

4.1. Container technology for executing program code

The main components of a container are a containerizer (e.g., Docker), images, and orchestrators (Fig. 4.1).

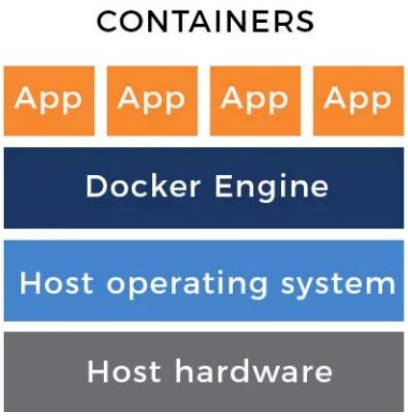


Fig. 4.1. Container virtualization

Containers are independent and isolated environments for executing applications. They include all the necessary components to execute an application, including code, libraries, environment, and configuration. Containers allow to package applications and their dependencies in a single image, making them easier to deploy and work with.

Images are container templates that contain all the information about how to create and run a container. They include the underlying operating system, code, and settings. Images are usually published in central repositories from which they can be downloaded and used.

Container orchestrators like Kubernetes help manage and deploy containers at scale. They automate the processes of load distribution, scaling, load balancing, monitoring, and disaster recovery.

Kubernetes is an open source system for automating the deployment, scaling, and management of containerized applications. Kubernetes allows users to group containers that comprise applications and manage them as a single entity, simplifying their lifecycle management.

Kubernetes provides automatic load balancing, self-healing applications, scaling to meet load, and non-disruptive upgrades. Using the concepts of pods, services, controllers, and namespaces, Kubernetes provides flexible and robust tools for building modern microservices architectures. With its support for various environments, from on-premises servers to cloud platforms, Kubernetes is becoming an indispensable tool for ensuring continuous integration and delivery in DevOps practices.

Containers provide a high level of isolation between applications running on the same server. Each container has its own environment, which avoids conflicts between applications and ensures their independence.

Container technology allows users to scale applications by adding or removing containers as needed, making it easier to respond to load increases and ensuring high availability. Containers enable applications to launch and terminate quickly because they share operating system kernel and server resources. This helps reduce infrastructure costs and improve performance.

Server-side container technology has become a popular solution for deploying and managing applications due to its efficiency, isolation, and ability to automate operations. It allows developers and engineers to focus on developing applications without spending a lot of time configuring and managing infrastructure.

4.2. Big data engineering

Big data engineering is the process of collecting, processing, storing, and analyzing large amounts of data to obtain valuable information and use it to make management decisions.

Consider the main aspects of big data engineering.

Data collection involves gathering data from various sources such as websites, sensors, social media, server logs, etc. Data collection and integration tools are used for this. Data collection methods can include web scraping (collecting data from websites), reading data from server log files, obtaining data from sensors and IoT devices, importing data from databases, etc.

Data cleaning and preparation includes removing duplicates, detecting and correcting errors, converting data to the required format, removing anomalous values, and aligning data.

Storing processed data requires powerful storage systems. Traditional databases may not be sufficient, so use distributed storage systems such as Hadoop HDFS, Amazon S3, or NoSQL databases like MongoDB or Cassandra.

Big data has the potential to extract valuable information. Data analysis uses machine learning algorithms, natural language processing (NLP), statistical analysis, classification, clustering, data visualization, and other techniques to identify patterns and trends.

The infrastructure for processing big data must be scalable and resilient. This includes the use of cloud computing, containerization (such as Docker and Kubernetes), automation and monitoring tools. Distributed computing platforms such as Apache Hadoop and Apache Spark, as well as cloud services such as Google Cloud and Amazon Web Services (AWS) are used. Data security is critical because processing large amounts of data can cause privacy and security issues. Therefore, it is necessary to use encryption and authentication and authorization methods to protect data.

Big data engineering must be scalable. Big data can grow exponentially, and the system must be ready to handle the increase without sacrificing performance. After analyzing the data, big data engineers must provide a convenient way to display the results for decision-making. Overall, big data engineering is an interdisciplinary field that requires an understanding of both the technical and business aspects of data processing and analysis. It allows organizations to find valuable information in data sets that would previously have been inaccessible or difficult to analyze using traditional methods.

4.3. SaaS, PaaS and IaaS

SaaS (Software as a service), PaaS (Platform as a Service), and IaaS (Infrastructure as a service) are main cloud service models that provide different levels of control and accountability for users and developers.

SaaS (Software as a Service) is a cloud computing model in which users are provided with access to ready-made software over the Internet. Users do not have to install, configure, or manage the applications, as the service provider does all of this (Fig. 4.2).

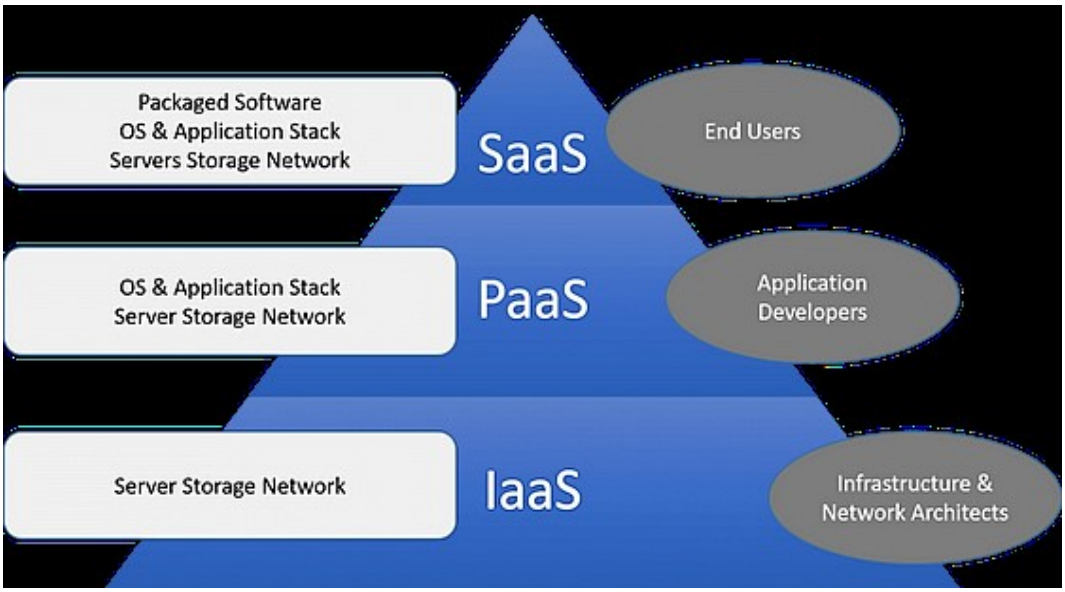


Fig. 4.2. Cloud service models

Examples of SaaS are Google Workspace (formerly G Suite), Microsoft 365, Salesforce, Dropbox, Slack.

For example, if a company uses Google Workspace for email, collaboration, and other business functions, that company is using SaaS in this case because there is no need to manage servers or install applications – it simply uses a web interface.

PaaS (Platform as a Service) is a model that provides users with a platform to develop, test, and deploy their own applications. This model includes a development environment, databases, and application building tools. Examples of PaaS are Google App Engine, Microsoft Azure App Service, Heroku, IBM Cloud Foundry.

For example, if a developer develops their own web application on the Heroku platform, which uses PaaS, Heroku provides the necessary tools for developing and deploying applications, and the developer can focus on their code without going into the details of infrastructure management.

IaaS (Infrastructure as a Service) is a model in which users have access to virtual infrastructure (servers, storage, networks) over the Internet. They can manage the infrastructure using APIs or other tools, but they are responsible for the operating system and additional software. Examples of IaaS are Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), DigitalOcean.

For example, if a company rents a virtual server on AWS and installs

its own operating system and applications on it, in this case IaaS is used, since the company has full control over the operating system and resources, but also needs to take care of their management. SaaS provides ready-made applications, PaaS provides platforms for development, and IaaS provides infrastructure for managing infrastructure and applications. The choice between these models depends on business needs and the level of control over the infrastructure and applications. SaaS delivers ready-to-use applications over the Internet, PaaS provides a platform for developing, testing, and deploying applications, and IaaS offers basic computing infrastructure, such as servers, storage, and networks, for user customization.

4.4. Lambda and Kappa architectures of big data analytics

Lambda and Kappa architectures are two different approaches to big data analytics that are used to process and analyze large amounts of information.

Lambda architecture is an approach to big data analytics that involves using two main ways to process and analyze data: batch and streaming. This architecture arose as a response to the need to analyze data in real time, but also to take into account the possibility of processing batches of data. Consider the components of the Lambda architecture (Fig. 4.3).

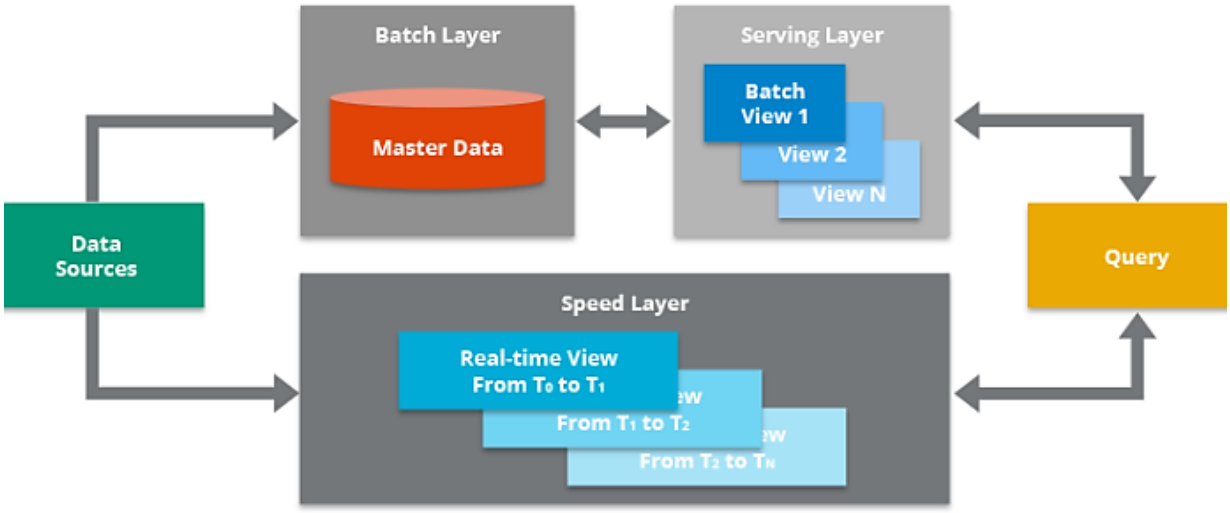


Fig. 4.3. Lambda architecture example

Data can be obtained from various sources and then analyzed. This component is often a streaming source, for example, Apache Kafka is an

intermediate store that can store data to serve both the batch layer and the velocity layer of the Lambda architecture. The data is delivered simultaneously to both the batch layer and the velocity layer to enable parallel indexing.

At the Batch Layer , data is processed in batches. Batch processes create "views" or preliminary results that can be used to analyze large amounts of data. These results can be stored in a database such as Hadoop HDFS or Apache Cassandra.

The Serving Layer combines the results of both other paths to prepare the data for presentation and analysis. This layer progressively indexes the latest batch views so that end users can query them. This layer can also re-index all the data to fix a coding error or create different indexes for different use cases. A key requirement of the Serving Layer is that the processing is done in a parallelized manner to minimize the time it takes to index the dataset. When an indexing job is running, the newly retrieved data will be queued for indexing in the next indexing job.

At the Speed Layer, data is processed in real time, arriving as a stream. Streaming processes allow data to be analyzed instantly and produce real-time responses to events. This layer complements the Service Layer by indexing newly added data that has not yet been fully indexed by the Service Layer. This includes data that the Service Layer is currently indexing, as well as new data that has arrived since the start of the current indexing job. Because there is an expected delay between the time the latest data is added to the system and the time the latest data becomes available for querying (due to the time it takes for the batch indexing job to run), the Speed Layer relies on indexing the latest data to reduce this gap. This layer typically uses streaming software to index incoming data in real time to minimize the delay in getting the data available for querying. When the Lambda architecture was first introduced, Apache Storm was the leading stream processing engine used in deployments, but other technologies have since gained more popularity as candidates for this component (e.g. Hazelcast Jet, Apache Flink, and Apache Spark Streaming).

The Query component is responsible for sending end-user queries to both the service level and the speed level and consolidating the results. This provides end-users with a complete query of all data, including the most recently added data, to provide a real-time analytics system.

The batch/service layers continue to index incoming data in batches. Since batch indexing takes time, the speed layer complements the batch/service layers by indexing all new, unindexed data in real time. This provides a consistent view of the data in the batch/service layers that can be replayed at any time, as well as a smaller index that contains the most recent data.

Data is indexed simultaneously at both the service tier and the velocity tier. Once a batch indexing job is complete, the new batch indexed data becomes available for querying, so the velocity tier copy of the same data/indexes is no longer needed and is therefore removed from the velocity tier. The service tier then starts indexing the latest data in the system that has not yet been indexed by that tier and has already been indexed at the velocity tier (and is therefore available for queries at the velocity tier). This continuous handoff between the velocity tier and the batch/service tiers ensures that all data is ready for queries and that the latency of data availability is low. When the service tier completes, it moves on to the next batch, and the velocity tier discards its copy of the data that the service tier just indexed.

Raw data is indexed at the service tier so that end users can query and analyze all historical data. Because batch indexing takes time, there tends to be a relatively large time window of data that is temporarily unavailable to end users for analysis. The velocity tier uses streaming technologies to immediately index recent data that is currently unavailable at the batch/service tiers, thereby narrowing the time window of data that is not available for analysis. This helps reduce the latency (i.e., the time it takes for data to become available for analysis) inherent in the batch/service tiers. One of the key ideas of the lambda architecture is that it eliminates the risk of data inconsistency, which is often seen in distributed systems. In a distributed database, where data cannot be delivered to all replicas due to node or network failures, there is a possibility of data inconsistency. In other words, one copy of the data may reflect the current value, but another copy may have a previous value. In a lambda architecture, because the data is processed sequentially (rather than in parallel with overlap, which can occur with operations in a distributed database), the indexing process can ensure that the data reflects the latest state at both the batch and speed levels. The lambda architecture does not specify the exact technologies to use, but is based on distributed, scalable technologies that can be expanded by adding more nodes.

This can be done at the data source, at the batch level, at the service level, and at the speed level. This allows users to use the lambda architecture regardless of how much data we need to process.

If a hardware failure occurs, other nodes are available to continue the workload. Since all data is stored at the batch tier, any failures during indexing at the service tier or at the velocity tier can be overcome by rerunning the indexing jobs at the batch/service tiers and allowing the velocity tier to continue indexing the latest data.

Kappa architecture is an alternative approach to big data analytics that combines streaming and batch processing in a single thread (a thread). The basic idea is to process all data as a stream of events, even for batch jobs. One of the main advantages of Kappa architecture over Lambda architecture is that it allows users to build streaming system and batch processing system on the same technology. This means that a developer can create a stream processing application to process data in real time, and if he needs to change the original data, he updates his code and then runs it again on the data in the messaging system in batch mode (Fig. 4.4).

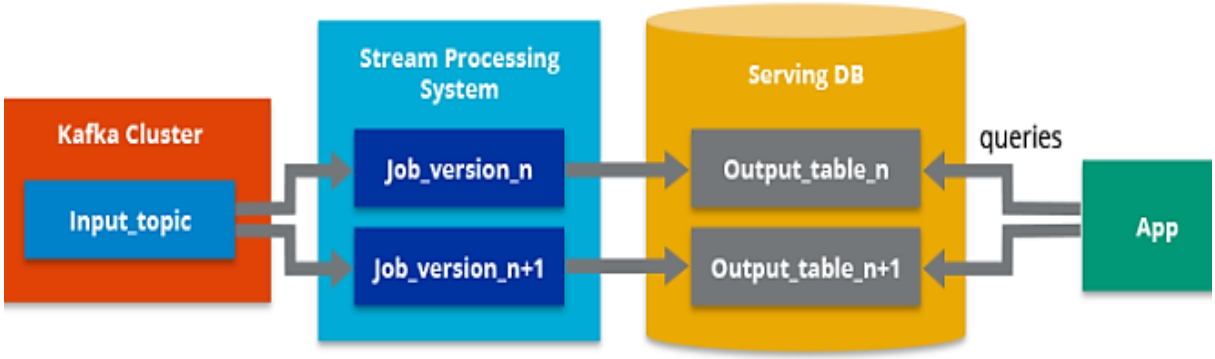


Fig. 4.4. Example of Kappa architecture

In this case, data is fed as a stream of events in real time (Stream Input), data is processed immediately upon receipt, rather than in batch mode (Stream Processing), batch processing can be used to improve data quality, resolve conflicts, or other tasks that require batch processing. Cloud service models such as SaaS, PaaS, and IaaS provide users and developers with different levels of access to computing resources and infrastructure – from full control to complete abstraction and ready-made solutions from cloud service providers. Knowledge and use of containers, effective big data engineering,

and the right choice of cloud service models are essential for modern companies and developers to successfully work with software products and data in an online environment. Lambda architecture and Kappa architecture are two different approaches to processing and analyzing large amounts of data. The choice between these two architectures will depend on the specific needs of a particular project, performance requirements, available resources, and other factors. The Kappa architecture can be useful for tasks where low latency and the ability to respond in real time are important without the need for large computational resources, while the Lambda architecture can be useful for tasks where detailed batch processing and merging of the results of both paths are required.

Control questions and tasks for topic 4

List of theoretical questions

1. What are the advantages of using containers over traditional application deployment methods?
2. How do containers contribute to scalability and high availability of applications on servers?
3. How does big data engineering help in business analytics and strategic decision-making?
4. What are the main differences between SaaS, PaaS, and IaaS?
5. What stages does the big data engineering process include?
6. For what types of tasks is each type of cloud service model best suited?
7. How do containers provide isolation between applications and their dependencies?
8. What are the benefits and risks associated with using SaaS?
9. What does IaaS include and how does it allow companies to manage infrastructure in the cloud?
10. What are the features of using Lambda architecture and Kappa architecture, what are their differences?
11. How does the orchestration of containers (e.g., with Kubernetes) enhance the management of distributed applications?
12. What are the key security considerations when using cloud-based services for big data processing and storage?

List of tasks for independent work

Practical task. Implementing microservices with containers.

Step 1. Create two microservices. Service 1. A user management service that handles user registration, authentication, and profile management. Service 2. A product catalog service that allows users to browse and manage product listings. Each microservice should expose a REST API to interact with clients.

Step 2. Choose a programming language and framework for each microservice (e.g., Node.js for Service 1 and Python with Flask for Service 2). Write a Dockerfile for each microservice to define its environment and dependencies.

Step 3. Create a docker-compose.yml file to define separate containers for Service 1, Service 2, and their respective databases (e.g., MongoDB for Service 1 and MySQL for Service 2). Configure the containers to communicate over a shared virtual network. Ensure proper environment variables for database connections and API settings.

Step 4. Start the entire environment using docker-compose up. Test API endpoints of both microservices to ensure they interact correctly with their databases. Simulate inter-service communication by having Service 1 make requests to Service 2 (e.g., retrieve a product listing for a user).

Step 5. Write detailed instructions for deploying the microservices on any server using Docker and Docker Compose. Include testing steps, example API requests, and troubleshooting tips.

List of used and recommended literature

4.1. What is containerization. [Electronic resource]. – Access mode: https://www.ibm.com/topics/containerization

4.2. What is SaaS. [Electronic resource]. – Access mode: https://www.salesforce.com/saas/

4.3. What is Platform as a Service. [Electronic resource]. – Access mode : https://cloud.google.com/learn/what-is-paas

4.4. What is infrastructure as a service. [Electronic resource]. – Access mode: https://www.ibm.com/topics/iaas

4.5. What is Stream Processing. [Electronic resource]. – Access mode: https://hazelcast.com/foundations/event-driven-architecture/stream-processing/

Topic 5. Big data infrastructure. Distributed data and analytics

Big data infrastructure is a set of technologies, resources, platforms, software tools, and architectures used to collect, store, process, and analyze large amounts of data. This infrastructure allows for efficient processing of a variety of big data that cannot be processed by traditional methods. Distributed data and its analytics are key components for solving problems related to big data and allow for efficient use of resources to obtain valuable information. Software solutions for distributed big data computing are platforms and tools that allow to efficiently process large amounts of data by distributing tasks across different servers or nodes. They use parallel processing to quickly work with data that cannot be processed on a single server. The most common of them are Apache Hadoop, Apache Spark, Google Cloud Dataproc, and Microsoft Azure HDInsight, which help analyze large amounts of data and extract useful information from it.

5.1. Big data infrastructure components

Consider the main components of a big data infrastructure. Data acquisition systems include sensors, IoT devices, and other means of collecting real-time data. Data storage systems are databases, file systems, and distributed systems that store large amounts of data. Powerful computing resources include servers, clusters, compute nodes, graphics processors, and other resources. Data processing frameworks and tools are software tools that allow to process and analyze data, including Hadoop, Spark, Apache Flink, etc. Streaming systems are used for real-time data analysis, they process data "on the fly" as it arrives. Examples of streaming systems are Apache Kafka (a distributed platform for streaming data processing and messaging with low latency), Apache Flink (a framework for streaming and batch data processing with high performance), Apache Storm (a framework for real-time streaming data processing with a wide range of applications), Spark Streaming (a component of Apache Spark for streaming data analysis and processing), Amazon Kinesis (a series of AWS cloud services for real-time streaming data processing and analysis).

Visualization and analysis systems are software tools for displaying and analyzing data that help discover new connections and conclusions. And the infrastructure must ensure that data is protected from unauthorized access and confidentiality is maintained. As with any IT initiative, it is critical for an

organization to align business and technology needs when deciding to invest in analytics and big data. We are seeing an increase in the use cases of big data in financial services around customer data to explore new markets, build products, expand cross-sell opportunities, deliver targeted offers, and improve ongoing customer support.

Hadoop, an open-source distributed file system, is considered the cornerstone of big data processing. Open-source Hadoop limits upfront licensing costs for managing and processing large amounts of data and is data-agnostic, providing flexibility for banks and financial services companies that want to extract information from unstructured data sources such as blogs and social media. When implementing a cloud-based big data platform, organizations will benefit from leveraging the Investigative Data Lab (IDL).

IDL provides business users with access to data for signal mining in social media, product data, and unstructured data. Once signals are found, they can be aggregated across business lines, queried in the cloud (no enterprise backup and restore required), moved on-premises, hosted in a data warehouse, and shared in a cache model or through query tools (Fig. 5.1). Over time, acquiring large data sets becomes very expensive. By focusing on finding the signal in the data, banks can load only the valuable parts of the data into the enterprise data warehouse.

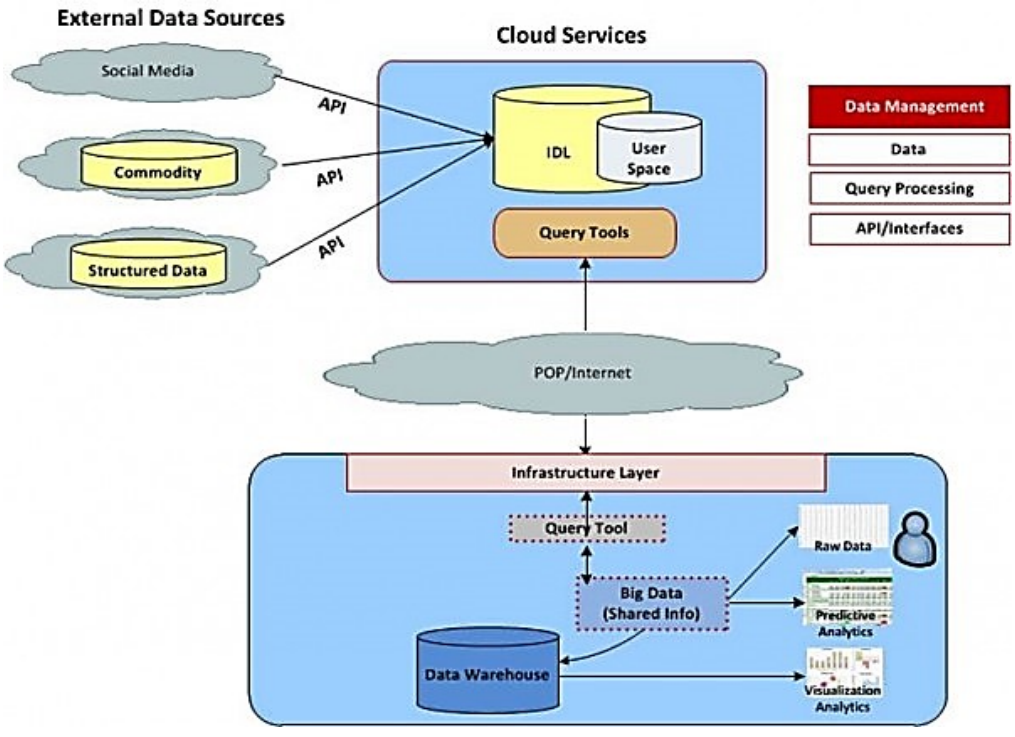


Fig. 5.1. Example of a cloud architecture for big data analytics

Cloud service providers can deploy real-time analytics, perform ad-hoc and batch analytics through big data infrastructure in the cloud. Big data infrastructure allows organizations to use large volumes of data to derive new information, optimize business processes, improve decision-making and many other tasks that require processing large amounts of information.

5.2. Distributed data and its analytics

Distributed data and analytics is a concept that refers to the processing and analysis of large amounts of data by distributing it across different nodes or servers in a network. This approach allows for efficient use of resources and speeds up data processing by dividing tasks into smaller parts that can be processed in parallel.

Distributed data means that data is physically distributed across multiple machines or nodes on a network. Instead of storing all the data in one place, it is broken into smaller blocks and distributed across different servers or clusters. This reduces the load on individual computing resources and allows for more efficient data processing.

One of the main advantages of distributed data is the ability to perform calculations on different parts of the data at the same time (parallel processing), which speeds up processing. Each server or node in the cluster can process its own part of the data, and the results are later combined. Distributed data analytics involves using various tools and methods to discover patterns in large amounts of information. This involves techniques such as averaging, histograms, data fusion from different sources, machine learning, etc.

MapReduce is a programming approach used to process and analyze large amounts of data in a distributed environment. MapReduce breaks the analysis task into two stages: the first, "Map", where the data is divided into smaller pieces and processed independently, and then the results are combined; the second, "Reduce", where the results of the processing are collected and aggregated.

Apache Hadoop is a framework for distributed data processing that includes the HDFS (Hadoop Distributed File System) file storage system and the MapReduce task execution platform.

Spark is a popular framework for distributed data analysis that provides greater performance through an optimized data processing engine, support for

interactive queries, and the ability to process streaming data.

The process of physically implementing distributed data and its analytics could look like this.

Step 1. First, the data is broken into smaller blocks or pieces. This can be done at the file system or database level. Each block of data can be stored on different servers or disks.

Step 2. If the data is stored in a database, the system can distribute the data to different servers using some distribution algorithm, such as hashing or range-based distribution. Each server will be responsible for processing its own portion of the data.

Step 3. Data analysis tasks are broken down into smaller tasks that can be processed in parallel on different servers or nodes. Each server or node processes its own portion of the data, and the analysis results can then be combined.

Step 4. Systems such as Apache Hadoop or Apache Spark are used to perform analytical operations and calculations. These systems divide the analysis tasks into smaller parts and execute them in parallel on different nodes.

Step 5. Once the analysis is complete, the results from the different nodes are combined, aggregated, and interpreted. This may include combining statistics, generating reports, or visualizing the results. Distributed data and its analytics are key components for solving problems related to large volumes of data and allow for efficient use of resources to obtain valuable information.

5.3. Architectures of parallel and distributed systems

The architecture of parallel computing systems is classified according to Flynn's Taxonomy, which defines four main categories.

SISD (Single Instruction, Single Data) is sequential processing where each piece of hardware executes a single instruction on a single set of data. It is typical of traditional processors.

SIMD (Single Instruction, Multiple Data) – a single processor executes a single instruction on different sets of data in parallel. This type of architecture is used in graphics processing units (GPUs), where a single set of instructions is executed on different data at the same time, which is efficient for tasks with a large amount of independent data, such as in

scientific computing.

MISD (Multiple Instruction, Single Data) – in this architecture, different instructions are executed on the same set of data simultaneously. It is not common in computing systems, but is sometimes used in data stream processing, for example, for complex filters.

MIMD (Multiple Instruction, Multiple Data) is the most popular architecture for distributed and clustered systems, where each processor executes different instructions on different data sets. This model is used in most modern multiprocessor systems and supercomputers.

Modern processors can have multiple cores, allowing for parallel computing within a single chip. Multi-core processors allow tasks to be distributed across cores, speeding up data processing. Each core has its own level one (L1) cache, and multiple cores can share a level two or level three (L2 or L3) cache, which improves performance when sharing data.

Processors, or central processing units (CPUs), are an integral part of computers and other computing devices. They perform key operations, processing data and ensuring the execution of programs.

An example is Intel and AMD processors, which have from two to dozens of cores, which allows for simultaneous processing of a large number of threads. This reduces the execution time of multitasking processes, such as graphics visualization, scientific calculations, etc. In the world of computing, two large companies – Intel and AMD stand at the sources of processor production.

AMD continued to strengthen its position in the computing industry with products such as the Ryzen 3000 and 5000 desktop processors and EPYC server processors. Infinity Fabric and Zen architecture technologies brought significant improvements in efficiency and performance. AMD Ryzen is known for its Zen architecture, which represents a significant leap in performance. Precision Boost and Precision Boost Overdrive technologies provide improved performance depending on the workload.

Intel processors, including the latest Tiger Lake models, deliver high performance with Turbo Boost Max and Hyper-Threading technologies. Their Sunny Cove and Willow Cove microarchitectures are designed to ensure efficient instruction execution.

AMD typically offers more cores and threads than comparable Intel processors in the same price range, making their processors more attractive

to those who do parallel computing and multitasking.

Intel also offers multi-core processors, but the issue of core and thread counts is becoming less of an issue with recent architectures. They are actively developing multi-tasking solutions for their processors. Intel offers a wide range of price points, including both affordable and expensive models. They also face competition in the budget segment from AMD.

Some AMD processor models, such as the Ryzen G-series, come with integrated Radeon graphics, making them the optimal choice for gaming and basic graphics tasks.

Intel also offers processors with integrated Intel UHD or Iris Xe graphics, providing basic graphics performance for everyday tasks.

AMD Ryzen processors successfully compete with Intel in gaming performance, and their multitasking capabilities make them better for creative professionals (Fig. 5.2).

Fig. 5.2. AMD Ryzen 5 PRO 4650G processor on the Renoir core

Intel remains a significant player in the gaming industry, with their Core i9 processor line ranking high in performance for professional applications.

Clusters are groups of computing nodes (servers) connected by a network that work as a single system. They are used to process large amounts of data and perform high-performance computing, for example, in scientific research and commercial applications.

Examples of data processing clusters are Apache Hadoop, Apache Spark, and Big Data Platforms from leading companies such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure. The main types

include High-Performance Computing (HPC) clusters, where nodes work on shared tasks, and clustered data processing systems like Hadoop, which are used to analyze large amounts of data.

Clusters typically include a master node and several worker nodes. The master node distributes tasks and coordinates computations, while the worker nodes perform parts of the tasks.

ShARC (Sheffield Advanced Research Computer) is a supercomputer cluster at the University of Sheffield designed to support high-performance computing (HPC) research. Its architecture consists of a variety of computing nodes, including nodes with multi-core processors and accelerators such as GPUs, allowing it to process large amounts of data and perform complex simulations. ShARC supports research in areas that require significant computing resources, such as climate change modeling, materials science research and genomic data analysis. The system is available to researchers at the university and other academic institutions through a resource management system that allows for optimal allocation of cluster capacity.

Consider the main characteristics and total capacity of ShARC computing resources.

- Nodes: 121 working nodes.
- Processor cores: 2024.
- Total RAM: 12160 GiB.
- Graphics processing units (GPUs): 40 units.
- Storage system based on the Lustre file system: 669 TiB.
- Opportunities and benefits for the ShARC topology.
- High-level parallel computing.

With 2024 processor cores and 121 worker nodes, ShARC provides the ability to perform large-scale calculations in parallel, ideal for scientific simulations, modeling, and big data analysis.

The presence of 40 GPUs allows ShARC to be effectively used for tasks requiring GPU-based computing, such as machine learning, deep learning, and image processing. The amount of RAM of 12160 GiB allows to process large data sets and perform tasks that require a significant amount of memory, such as working with large databases and complex algorithms.

The 669 TiB Lustre file system provides fast access to large amounts of data, which is important for tasks related to data analysis, simulations, and high-performance computing. Part of the resources is reserved for research

groups, which allows for exclusive access for important projects and research, increasing the efficiency of the system.

Grid computing uses the resources of geographically distributed computing systems to solve large problems. It is based on the principle of collective resource use, where nodes are combined into a network, providing their resources to process part of the problem. Grid computing is used in data processing projects, scientific research, such as genetic information processing, meteorological research, simulations of physical processes, etc.

One practical example of Grid computing is the SETI@home (Search for Extraterrestrial Intelligence at Home) project, which uses a distributed computing network to analyze radio signals from space. In this project, millions of users around the world installed a special client application on their computers that downloaded small pieces of data from the project's servers, performed complex calculations locally, and returned the results. Thanks to Grid computing, SETI@home received a powerful computing infrastructure created from thousands of individual computers, without the need to maintain expensive supercomputers. This made it possible to analyze huge amounts of data, searching for patterns or potential signals of extraterrestrial origin, with maximum efficiency and minimal costs.

Grid is a computing architecture that allows a large number of distributed resources to be coordinated to achieve a common goal. In our case, these resources are the computers whose processors are used to perform this computation. These resources can be distributed, that is, physically remote and connected by a computer network.

Each of these N nodes (i.e. computers) will perform a small part of the computation on its processors, allowing to produce the result (in an ideal world) N times faster than with a single computer. A real-world example of Grid computing is the LHC Computing Grid project, created to process the huge amounts of data generated by the Large Hadron Collider (LHC) at CERN (french Conseil Européen pour la Recherche Nucléaire) – an international research center of the European community, the world's largest high-energy physics laboratory.

LHC Computing Grid (LCG) is a distributed computing infrastructure designed to process the vast amounts of data generated by the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN). The LCG consists of over 170 computing centers in 42 countries around the

world and is capable of processing over 30 petabytes of data. per year, which are formed as a result of physical experiments at the LHC.

This infrastructure is divided into several levels (Tiers), where each level has its own functions.

Tier-0 (CERN) is the main computing center where data is initially stored and basic calculations are performed.

Tier-1 centers are located in different countries and are engaged in storing copies of data and processing it.

Tier-2 centers are smaller computing centers where data analysis is performed by researchers.

LCG allows thousands of scientists around the world to analyze the results of experiments at the LHC in real time, providing fast access to data, high-performance processing, and collaboration between researchers regardless of their location.

European Organization for Nuclear Research (CERN) is one of the world's largest and most renowned particle physics research centers, founded in 1954 and located on the Swiss-French border near Geneva. CERN's mission is to study the fundamental constituents of matter and the forces that act between them by conducting experiments at high energies.

CERN's centerpiece is the Large Hadron Collider (LHC), the world's largest particle accelerator, which allows researchers to accelerate protons to speeds close to the speed of light and collide them to study the properties of matter at the subatomic level. CERN is also a pioneer in computing, having invented the World Wide Web as a means of exchanging scientific information between researchers.

To process the large amounts of data generated during experiments, CERN uses the LHC Computing Grid, a powerful distributed computing network that allows scientists around the world to collaborate on research.

CERN is an international organization with over 20 member countries and involves thousands of scientists from around the world, creating a global platform for research that helps unlock the mysteries of the Universe.

Every second, the collider produces about 1 petabyte of data about particles interacting at high energies. To process such volumes of information, CERN combines the computing power of several hundred institutes around the world, creating a global Grid system. Researchers from different countries can connect to the network to store and analyze data, which

allows them to work on large amounts of information in parallel and accelerates scientific discoveries in the field of high-energy physics.

1 petabyte (PB) is a unit of digital information equal to 1,024 terabytes (TB) or 1,048,576 gigabytes (GB). To put it in a more understandable sense, 1 PB is roughly equivalent to the amount of information needed to store about 500 billion pages of text or about 13 years of high-definition (HD) video. Petabytes are often used to measure data volumes in large storage facilities such as data centers, scientific institutions, and cloud services, where huge amounts of information are accumulated for analytics, modeling, and processing. An example of the use of SIMD in graphics processors: when rendering images, the GPU performs operations on many pixels simultaneously, which significantly speeds up the process.

Multi-core processors are used in personal computers. For example, the 4-core Intel i5 processor allows to distribute tasks between cores, reducing processing time, which is noticeable when running resource-intensive programs.

In banking, a bank can use a cluster to process large amounts of data to detect fraudulent transactions in real time.

5.4. Programming models for parallel computing

Parallel programming is the process of writing programs that run on multiprocessor or multicore systems, allowing multiple computational operations to be performed simultaneously.

OpenMP (Open Multi-Processing) is a standard for parallel programming on multi-core processors using compiler directives.

- Supports distribution of computations between processor cores.
- Suitable for programming in C, C++, Fortran.
- The main concepts are cycle parallelization, thread synchronization, resource management (sections, barriers, locking).

OpenMP is typically used in C/C++ or Fortran . Let's look at an example of using OpenMP to compute the sum of an array in parallel in the C programming language:

```c
#include <stdio.h>
#include <omp.h>
int main() {
int n = 1000; // Array size
int arr[n];
int sum = 0;
// Array initialization
for (int i = 0; i < n; i++) {
arr[i] = 1; // Example: all elements are equal to 1
}
// Using OpenMP to parallelize computations
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < n; i++) {
sum += arr[i];
}
printf("Sum of array elements: %d\n", sum);
return 0;
}
```

In this example, `#include <omp.h>` is the inclusion of the OpenMP header file.

`#pragma omp parallel for` is an OpenMP directive that tells the compiler to parallelize the execution of a loop. All iterations of the loop will be automatically distributed among the available threads.

`reduction(+:sum)` is a directive that indicates summation. This means that OpenMP will create a copy of the sum variable for each thread and at the end will combine (reduce) all the results into a single variable by adding them.

The array `arr` is filled with values, and we calculate the sum of all its elements in parallel.

OpenMP automatically distributes loop iterations between multiple threads, which allows for faster computations on multi-core systems. To compile this program with OpenMP, we need to use the `-fopenmp` option in the GCC compiler:

```
gcc -fopenmp example.c -o example
```

This allows users to use OpenMP to automatically process tasks across multiple threads, which is especially effective for large-scale computations on multi-core systems.

MPI (Message Passing Interface) is a standard for passing messages between processes that may be running on separate nodes in a cluster.

- Used for programming in C, C++, Fortran.
- Suitable for execution on distributed systems where

communication between processes is important.

- The main concepts are message transmission, synchronization, communicators, and data distribution.

Consider an example of using MPI for broadcasting data to all processes in the C programming language. This task demonstrates how to use MPI to broadcast data from one process (root) to all other processes in the communicator:

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(&argc, &argv);
    // Get the rank of the current process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    // Get the total number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    // The root process (rank 0) initializes the data
to be broadcasted
    int data = 0;
    if (world_rank == 0) {
        data = 100; // Initialize data
        printf("Process 0 broadcasting data: %d\n",
data);
    }
    // Broadcast the data from process 0 to all
processes
    MPI_Bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
    // Each process outputs the received data
    printf("Process %d received data: %d\n",
world_rank, data);
    // Terminate the MPI environment
    MPI_Finalize();
    return 0;
}
```

`MPI_Init` initializes the MPI environment for communication between processes.

`MPI_Comm_rank` determines the rank (identifier) of the process in the communicator.

`MPI_Comm_size` retrieves the total number of processes in the communicator.

`MPI_Bcast:` broadcasts data from one process (root) to all other processes.

In this case, process 0 is the root process, and it broadcasts the value of data to all other processes.

`MPI_Finalize` cleans up the MPI environment when the program finishes.

This broadcast mechanism is commonly used in distributed systems to share configuration data or global parameters efficiently with all processes. It ensures synchronized execution across processes in parallel computing environments. MPI allows users to create scalable distributed programs that can run on clusters and supercomputers.

Thread is an independent sequence of instructions that can execute simultaneously with other threads within the same process. Using threads allows for efficient allocation of computing resources. Programming languages such as C++, Python, Java have multithreading support.

Multithreading is a technology that allows programs to execute multiple threads (threads) simultaneously, using multi-core processors or multiprocessor systems to improve performance and efficiency. Threads (or threads) are separate sequences of instruction execution within a single process that can run in parallel and share process resources such as memory.

In the context of an operating system, threads execute within a single process, using its resources.

Process – it is an executable program with a block of resources (e.g. memory) allocated to it. Each process has at least one thread.

Multithreaded program is a program that can create and manage multiple threads simultaneously.

Multithreaded programs can perform multiple tasks simultaneously, reducing program execution time on multi-core processors. Interfaces, such as graphical ones, can remain responsive to the user even when the application is performing background tasks in separate threads. The use of multi-core

systems allows users to distribute tasks between cores and effectively use computing power.

Threads can be used to handle various aspects of the game, such as physics calculations, sound processing, game logic, and graphics rendering.

Multithreading allows users to process multiple requests from different clients simultaneously. Individual threads can perform tasks in the background so as not to block the main interface.

Writing multithreaded code is more difficult because we need to consider the possibility of data races, deadlocks, and other concurrency issues.

Sharing resources requires the use of synchronization mechanisms (mutexes, semaphores, etc.) to avoid data corruption or incorrect program execution.

Programming language has built-in support for multithreading through the Thread class and synchronization mechanisms.

C++ supports multithreading through the <thread> library and other mechanisms such as mutexes and lambda functions for creating threads.

Python supports multithreading through the threading module, although due to the GIL (Global Interpreter Lock), computational threads can operate inefficiently.

OpenMP provides simple directive multithreading support for the C, C++, and Fortran languages.

Pthreads (POSIX Threads) is a standard for implementing multithreading on many platforms.

Multithreading is used in many areas of programming to provide efficiency and flexibility to programs. It is especially useful for creating high-performance systems that require parallel execution of tasks.

Multithreading support allows users to create programs that perform multiple tasks simultaneously. Synchronization mechanisms, such as mutexes, semaphores, and events, are used to ensure correct access to shared resources. Large arrays can be distributed across multiple cores for simultaneous processing.

Some libraries (such as NumPy for Python) have optimized methods for working with large data sets.

Threads can process large amounts of data simultaneously, which significantly speeds up calculations.

Technologies such as CUDA (for GPUs) allow parallel computing to be used to work with arrays on GPUs.

Consider the main tools for analyzing and debugging parallel programs.

Intel VTune, GNU gprof used for profiling programs to identify bottlenecks.

Valgrind is a tool for detecting memory errors and analyzing performance.

TotalView, GDB are debugging software tools that support multithreading; *OpenMP Debugging Tools* are specialized tools for debugging OpenMP programs.

Measuring the performance of parallel programs is critically important because it allows users to assess the efficiency of computing resource use, identify bottlenecks and points of performance loss, and optimize the code for better load distribution between processes or threads.

Through profiling and performance analysis, we can determine whether the application meets the expected performance and scalability characteristics, which is especially important for ensuring efficient operation on multi-core processors and in large cluster systems.

5.5. Software solutions for distributed big data computing

Distributed Big Data computing software solutions are comprehensive software platforms and tools that help efficiently process, analyze, and use large amounts of data by distributing computing tasks across different servers or nodes in a network. These solutions enable parallel processing to quickly and efficiently work with data that exceeds the processing limits of a single server.

Processing solutions allow to break analysis tasks into smaller parts that are processed in parallel on different servers or nodes, ensuring efficient use of resources. Some solutions offer a declarative approach to defining analytical operations, where users can describe what needs to be done with the data, and the program decides how best to do it across different nodes. Many software solutions work with distributed storage systems, which allow for efficient storage and access to large amounts of data.

Developers of distributed computing solutions take into account the peculiarities of working in a distributed environment and optimize algorithms for efficient work with distributed data.

Software solutions for distributed big data computing can be scaled to provide greater computing resources. Many available solutions have integration capabilities with other tools, databases, and other technologies to provide complete data processing.

Examples of software solutions for distributed big data computing include the Apache Hadoop framework and the Apache Spark framework, which provides a wide set of data analysis tools, including distributed processing, an optimized computation engine, and machine learning support.

Google Cloud Dataproc is a cloud service from Google Cloud Platform (GCP) that provides the ability to use and manage distributed computing based on popular open-source big data frameworks such as Apache Spark and Apache Hadoop. This service allows developers to focus on data analysis while having the ability to automatically scale computing resources based on demand and ensure high availability of computing.

Dataproc allows users to quickly create clusters for data processing and destroy them when they are no longer needed, allowing for efficient resource utilization and reducing costs.

Dataproc supports frameworks such as Apache Spark, Apache Hadoop, Apache Hive, Apache Pig and others, allowing to choose the best tool for big data analysis. The service automatically scales computing resources depending on the volume of data and tasks and allows users to define the minimum and maximum number of nodes in the cluster.

Dataproc integrates with other GCP cloud services, such as Google Cloud Storage, Google BigQuery, Google Cloud Pub/Sub, and others, allowing to move data and analysis results between services.

Dataproc can work with various data storage systems, including HDFS (Hadoop Distributed File System), Google Cloud Storage, and others. Dataproc also leverages the ability to use GCP monitoring and management tools to track cluster performance, as well as excellent capabilities for tuning and optimizing performance.

Dataproc allows users to create and execute tasks using the Python programming language (Fig. 5.3).

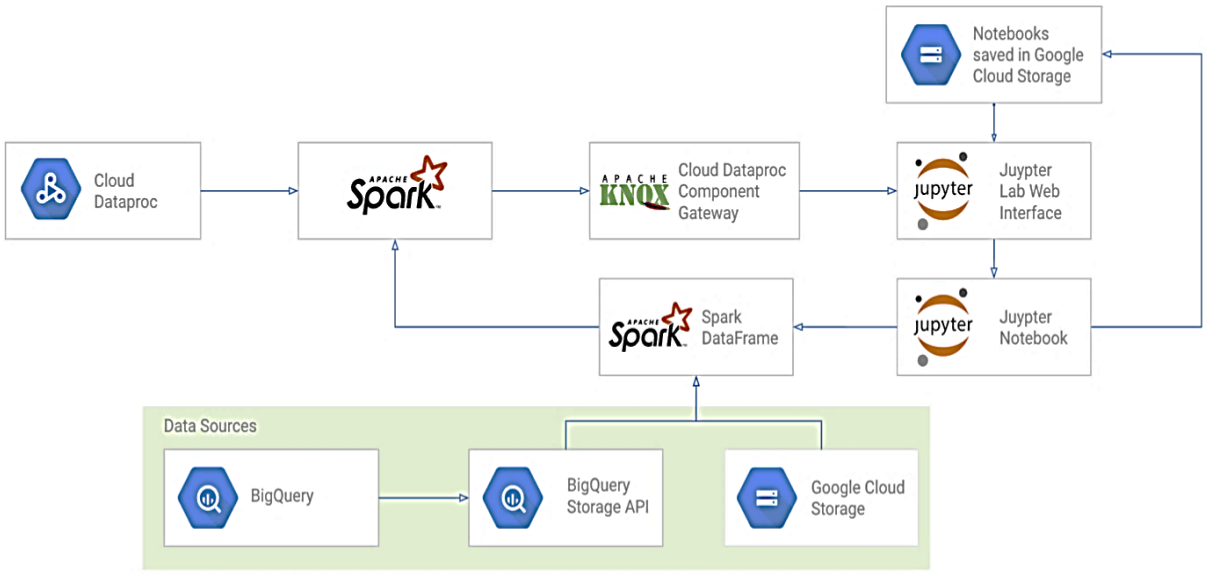


Fig. 5.3. Apache Spark and Jupyter on Cloud Dataproc

Microsoft Azure HDInsight is a cloud service from Microsoft Azure that enables to create, manage, and scale clusters for big data processing using various open frameworks such as Apache Hadoop, Apache Spark, Apache Hive, Apache HBase, and others. This service allows users to analyze large amounts of data using the power of distributed computing and integrate these computations with other Azure services.

HDInsight supports various frameworks and tools for big data processing, including Apache Hadoop, Apache Spark, Apache Hive, Apache HBase, Apache Storm, and others (Fig. 5.4). The developer can choose the best tool for the analytical tasks.

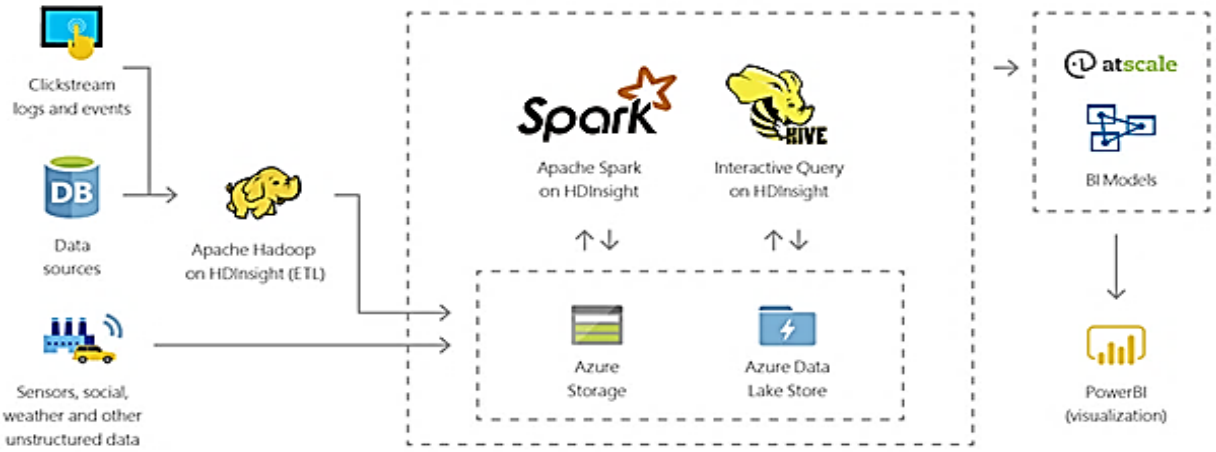


Fig. 5.4. Transformed data for processing in HDInsight

HDInsight integrates with other Azure services such as Azure Data Lake Storage, Azure SQL Database, Azure Active Directory, Azure Machine Learning, and many others.

HDInsight allows users to quickly create HDInsight clusters and destroy them when they are no longer needed, which allows for efficient use of resources and reduces costs. The service automatically scales computing resources depending on the volume of data and workloads. HDInsight allows users to use monitoring and management tools to track cluster performance, as well as tune and optimize operations.

HDInsight enables to work with a variety of data sources, including relational databases, non-relational storage systems, files, and Azure data. The service provides data, network, and identity layer protection to help keep infrastructure and data secure. HDInsight allows perform model and machine learning using integrated tools and libraries.

5.6. Distributed database management systems

Distributed database management systems (DDBMSs) are systems that store and manage data across multiple physical locations, providing a single view of the database to users. They allow data to be stored across multiple servers or data centers, which increases availability, reliability, and scalability. Key features of a DDBMS include transaction support, data consistency, load balancing, and fault tolerance. These systems enable organizations to efficiently process large amounts of data and provide fast access to it even under high loads or when individual network nodes fail.

Examples of distributed database management systems include *Google Spanner*, a globally distributed transactional database with high consistency; *Apache Cassandra*, which is known for its fault tolerance and scalability in large distributed environments; *Amazon DynamoDB*, which offers a serverless database with auto-scaling and high availability; and *CockroachDB*, which provides a distributed SQL database with auto-recovery and horizontal scaling capabilities. These systems allow for processing large amounts of data distributed across different geographical locations and maintain application uptime even under high loads and component failures.

Cassandra is a distributed database management system (DBMS) that was designed to provide high availability and scalability in big data processing systems. Cassandra has become especially popular where reliability and speed are important (network applications, real-time analytics, online advertising, etc.).

Cassandra was developed by Facebook employees in 2007. The idea for Cassandra was based on the Amazon Dynamo model, which Amazon uses to store data in distributed systems. This model is designed to provide high availability and scalability of data.

In 2008, Facebook donated the Cassandra source code to the Apache Software Foundation. Over time, Cassandra has become popular with a large number of companies that need a reliable and scalable DBMS for large amounts of data. Cassandra has found applications in various areas, including social networks, real-time analytics, and finance. The Apache community actively develops and supports Cassandra. It constantly improves the system, adding new features and improving performance. Overall, Cassandra has become the standard for many applications that require a distributed data store with high availability and scalability. Cassandra has gained recognition due to its unique capabilities and successful applications in large companies and various industries.

Cassandra uses a distributed architecture where data is distributed across multiple nodes in a cluster. This allows for high availability, scalability, and reliability. Nodes in a cluster are called "data nodes," and each node typically has the same set of data. Cassandra can scale its data and processing capacity by adding new nodes to the cluster. This allows it to handle large amounts of data and high workloads. Cassandra provides "failover" by allowing to configure data replication across multiple nodes. If one node fails, the data is still available on other nodes, ensuring reliability.

Cassandra supports various types of queries, including data modification queries (e.g., insert, update, delete) and data selection queries, and supports secondary indexes for complex queries. Cassandra uses a flexible data model where data can be organized into tables with arbitrary columns. This allows developers to store and manipulate data in a format that is convenient for them.

Cassandra supports ACID (Atomicity, Consistency, Isolation, Durability) properties to ensure data reliability and integrity, and supports

working with multiple data clusters across different data centers. This is useful for multi-regional applications and applications that require high availability. The reference architecture in Fig. 5.5 shows how to deploy virtual machines (VMs) and a virtual network configured for an N-tier application using Apache Cassandra on Linux for the data tier.

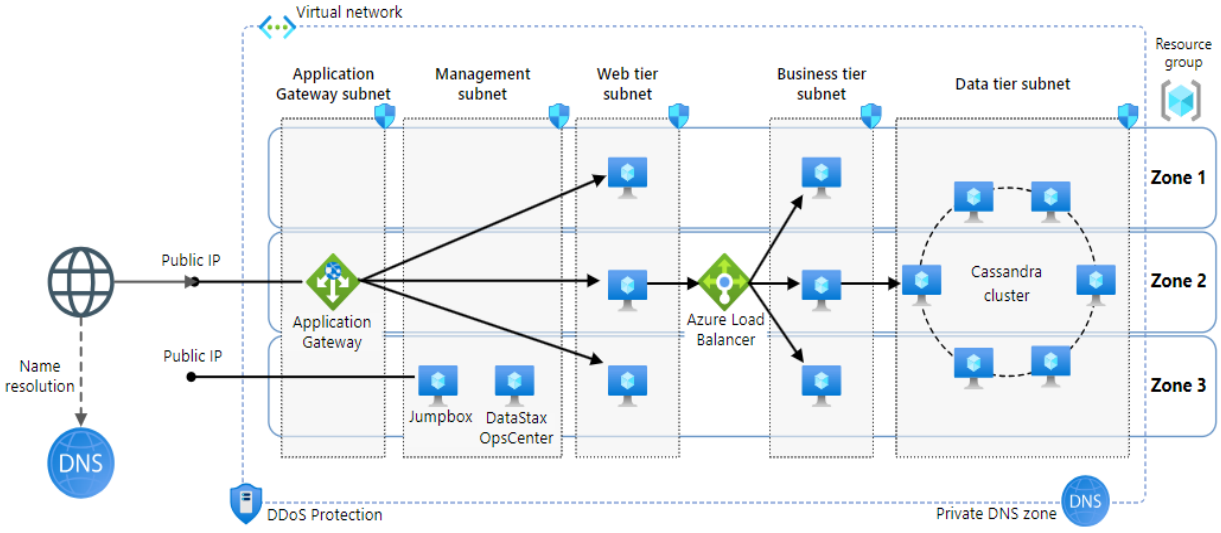


Fig. 5.5. N-tier application with Apache Cassandra

Fig. 5.5 shows the architecture of the cloud infrastructure in Azure, consisting of several subnets: Application Gateway subnet, Management subnet, Web tier subnet, Business tier subnet and Data tier subnet, where the Cassandra cluster is located in three availability zones (Zone 1, Zone 2, Zone 3). The Application Gateway receives traffic via Public IP and redirects it to the Azure Load Balancer, which distributes the load between servers of different tiers. The architecture supports DNS, DDoS Protection, as well as Private DNS zone to ensure reliability, scalability and uninterrupted operation of applications.

Resource groups are used to group Azure resources so that they can be managed by lifetime, owner, or other criteria. Availability zones are physical locations within an Azure region. Each zone consists of one or more data centers with independent power, cooling, and networking. By placing virtual machines across zones, application becomes resilient to zone failures.

Each Azure virtual machine is deployed in a virtual network, which can be segmented into subnets.

Application Gateway is a load balancer. In this architecture, it routes HTTP requests to the web interface. Application Gateway also provides a web application firewall (WAF) that protects the application from common exploits and vulnerabilities. Azure Standard Load Balancer is used to distribute network traffic from the web tier to the business tier.

NSG (network security groups) are used to restrict network traffic in a virtual network. For example, in the three-tier architecture considered, the database tier does not accept traffic from the web interface, only from the business tier and the management subnet. Although the Azure platform provides basic protection against distributed denial of service (DDoS) attacks, it is recommended to use Azure DDoS Network Protection.

Azure DNS is a hosting service for DNS domains that provides name resolution using Microsoft Azure infrastructure. Hosting domains in Azure allows users to manage DNS records using the same credentials, APIs, tools, and billing as other Azure services.

The Apache Cassandra database provides high availability at the data level by enabling replication and failover. A secure Jumpbox virtual machine on the network is used by administrators to connect to other virtual machines. The Jumpbox has an NSG that allows remote traffic only from public IP addresses on a safe list. The NSG must allow remote desktop traffic.

Parallel and distributed systems architectures facilitate the solution of complex problems by distributing the load across multiple nodes, while programming models such as MPI or MapReduce provide tools for creating high-performance software.

Distributed database management systems allow to efficiently work with data located on different geographically distributed servers, ensuring data consistency and availability. The combination of such technologies creates a basis for solving complex problems in science, business and industry, such as big data analysis, forecasting, modeling and development of new intelligent systems.

Control questions and tasks for topic 5

List of theoretical questions

1. What is big data infrastructure and what role does it play in modern data analysis?
2. What components make up a big data infrastructure and how do

they work together?

3. What is distributed data and why is it key to processing large amounts of information?

4. What are the advantages of distributed analytics over traditional data processing methods?

5. What are the main challenges associated with processing and analytics of distributed data?

6. What opportunities do software solutions provide for distributed big data computing?

7. What big data frameworks are supported in Microsoft Azure HDInsight?

8. How do software solutions help to process data distributedly on clusters?

9. What are the benefits of integrating software solutions with cloud computing platforms such as Google Cloud and Microsoft Azure?

10. What are some examples of practical scenarios for using distributed computing to analyze large amounts of data?

11. What are the possible problems and limitations associated with using Cassandra, and how can they be overcome in specific projects?

12. Give examples of distributed database management systems .

13. How does Cassandra provide high levels of availability and reliability for data?

14. What use cases can the Cassandra distributed database be particularly useful for large data storage systems?

15. What is the main difference between SISD and SIMD architectures?

16. What are the main advantages of MIMD architecture?

17. What are the differences between the OpenMP and MPI models for parallel programming?

18. What types of directives does OpenMP support, and how are they used?

19. How is communication between processes carried out in the MPI model?

20. How is parallel processing of arrays implemented and what libraries are used for this?

List of tasks for independent work

Practical task 1. Deploying a scalable web application on AWS.

Build a web application. Develop a simple web application using a framework of your choice, such as Flask (Python) or Express (JavaScript).

Test it locally to ensure it works as expected. Launch an EC2 Instance. Log in to the AWS Management Console and navigate to the EC2 Dashboard.

Launch a new EC2 instance, choosing an Amazon Machine Image (AMI) such as Ubuntu or Amazon Linux. Configure security groups to allow HTTP/HTTPS traffic. Set Up the Environment. Connect to the EC2 instance via SSH. Install required software (e.g., Python, Node.js) and dependencies for your web application. Deploy the Application. Copy your web application to the EC2 instance. Start the application and configure a reverse proxy (e.g., Nginx) to serve the application. Scale with Load Balancers. Navigate to the AWS Elastic Load Balancing service and create a load balancer.

Attach multiple EC2 instances to the load balancer to distribute incoming traffic. Test the Deployment. Access the web application via the load balancer's public URL. Verify its performance and scalability.

Practical task 2. Deploying a multi-tier application using Docker.

Develop the Application. Create a basic multi-tier web application with a frontend (e.g., React), backend (e.g., Flask), and database (e.g., MySQL).

Create Dockerfiles for each tier. Write a separate Dockerfile for the frontend, backend, and database tiers.

Example:

```
# Dockerfile for backend (Flask)
FROM python:3.9
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

Set up Docker Compose. Create a docker-compose.yml file to define the services (frontend, backend, database). Configure networking and dependencies between services.

Run the Containers. Use Docker Compose to start all containers:

```
docker-compose up
```

Test the Application. Verify that the frontend communicates with the backend and that the backend connects to the database.

Practical task 3. Setting up a Kubernetes cluster on Google Kubernetes Engine (GKE).

Containerize your application. Create a Docker container for your web application. Create a Kubernetes Cluster. Log in to the GCP Console and navigate to the Kubernetes Engine section. Create a new cluster with the desired number of nodes and configurations.

Deploy the Application. Write a Kubernetes Deployment YAML file to describe your application's pods and replicas. Example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web-app
  template:
    metadata:
      labels:
        app: web-app
    spec:
      containers:
      - name: web-app
        image: your-docker-image
        ports:
        - containerPort: 80
```

Expose the Application. Create a Service to expose the Deployment to external traffic. Test the Deployment. Access the web application using the external IP address provided by the Service.

Practical task 4. Configuring autoscaling for Kubernetes on AWS EKS.

Set Up EKS Cluster. Use the AWS Management Console or eksctl to create an EKS cluster. Deploy a Web Application. Create and deploy your containerized web application to the EKS cluster. Install the Kubernetes Metrics Server. Deploy the Metrics Server to collect resource usage metrics:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

Configure an HPA for your Deployment to scale based on CPU or memory usage. Example:

```
kubectl autoscale deployment web-app --cpu-percent=50 --min=1 --max=10
```

Create artificial load on the application using tools like Apache JMeter or Siege. Observe the scaling behavior using:

```
kubectl get hpa
```

Use AWS CloudWatch to monitor cluster performance and adjust autoscaling thresholds as needed.

List of used and recommended literature

5.1. Y. Demchenko, C. de Laat and P. Membrey. Defining architecture components of the Big Data Ecosystem, 2014 International Conference on Collaboration Technologies and Systems (CTS), Minneapolis, MN, USA, 2014. R. 104-112, doi: 10.1109/CTS.2014.6867550.

5.2. Understanding Big Data Infrastructure – A Complete Guide. [Electronic resource]. – Access mode: https://www.businesstechweekly.com/operational-efficiency/data-management/big-data-infrastructure/

5.3. Big Data Infrastructure Solutions and Services. [Electronic resource]. – Access mode : https://www.xenonstack.com/solutions/big-data-infrastructure/

5.4. What is Dataproc. [Electronic resource]. – Access mode: https://cloud.google.com/dataproc/docs/concepts/overview

5.5. Azure HDInsight. [Electronic resource]. – Access mode: https://learn.microsoft.com/en-us/azure/hdinsight/hdinsight-overview

5.6. Apache Cassandra Documentation. [Electronic resource]. – Access mode: https://cassandra.apache.org/_/index.html

5.7. OpenMP. [Electronic resource]. – Access mode: https://www.openmp.org/

5.8. Open Source High Performance Computing. [Electronic resource]. – Access mode: https://www.open-mpi.org/

Topic 6. Hadoop technology for big data analytics

A distributed system for processing and storing large amounts of data, Hadoop consists of the HDFS distributed file system for storage and MapReduce for data processing, allowing for efficient processing of big data on a cluster of servers. HDFS distributes data into blocks across the cluster, providing high availability and fast processing, while Hadoop YARN manages resources and computations, allowing for the addition of new components for data processing. The MapReduce paradigm divides tasks into "Map" phases for parallel processing and "Reduce" phases for combining results, which contributes to the efficient processing of large amounts of data.

6.1. Hadoop architecture

Hadoop is open source software that provides distributed processing of large amounts of data on clusters of servers.

The Hadoop architecture includes the following key components: Hadoop Distributed File System, MapReduce, YARN, Hadoop Common, and Hadoop Ecosystem.

Hadoop Distributed File System (HDFS) is a distributed file system designed to store large amounts of data across cluster nodes. HDFS divides large files into blocks and distributes them across nodes for parallel processing.

MapReduce is a programming and data processing model used in Hadoop. This model consists of processing data in two stages: a "map" stage, where the data is broken down into parts " key-value " , and a "reduce" stage where the results of "map" are processed and aggregated. MapReduce enables large-scale computation in a distributed environment.

YARN (Yet Another Resource Negotiator) is a resource manager that manages the allocation of resources (CPU time, memory, etc.) between different applications in a Hadoop cluster. YARN provides a more flexible and efficient way of managing resources than the previous version of MapReduce.

Hadoop Common is a set of tools and utilities that provide common services and libraries required for other Hadoop components to work. Hadoop Common includes command-line tools, networking libraries, and other common services.

In addition to the core components, there are many other projects and

solutions that are part of the Hadoop ecosystem. For example, Apache Hive for SQL queries, Apache Pig for data assembly, Apache HBase for real-time data processing, and many others. The Hadoop architecture is designed to process large amounts of data and scales by adding new nodes to the cluster. (Fig. 6.1). Hadoop allows enterprises to store, process, and analyze big data, which opens up opportunities for large-scale analytical projects and solutions.

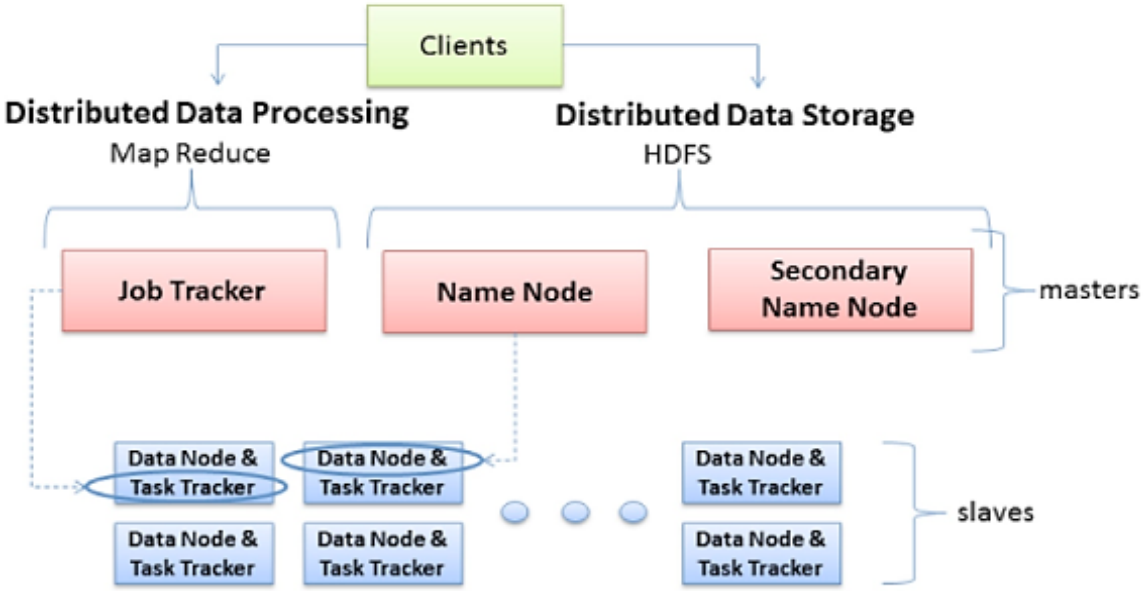


Fig. 6.1. An example of a Hadoop architecture representing the main roles of machines

The Hadoop architecture is based on two main roles: master/slave. Other roles related to the file system and distributed task execution are associated with each machine. Within the main machines , three main roles are related.

JobTracker is the master's responsibility for executing distributed tasks by coordinating slaves. JobTracker schedules execution, manages the state of slaves, and aggregates the results of MapReduce calculations.

NameNode participates in the operation of the HDFS distributed file system. The master machine associated with this role is responsible for distributing data to slave machines and managing the cluster namespace. The NameNode contains metadata reading, which allows users to know on which machine each file is located.

SecondaryNameNode – this role arises as part of the NameNode redundancy. Typically, this is assumed to be a physical machine other than the NameNode, ensuring continuous operation of the Hadoop cluster in the

event of its failure.

Each of the slave machines has two roles associated with them.

TaskTracker is a role that allows a slave to execute MapReduce jobs on the data it holds. The TaskTracker is controlled by the JobTracker of the master machine, which submits the jobs for execution.

DataNode is a node in the architecture that contains a portion of the cluster's data. Data nodes are typically replicated as part of the Hadoop architecture to ensure high availability of data.

When a client needs to access data or perform a distributed task, it goes through the master machine, which plays the roles of JobTracker and NameNode.

6.2. Hadoop distributed file system (HDFS)

The Hadoop Distributed Filesystem (HDFS) allows users to store large amounts of data on distributed server clusters, is used to process large amounts of data in the Hadoop ecosystem, and is a popular solution for big data analysis (Fig. 6.2).

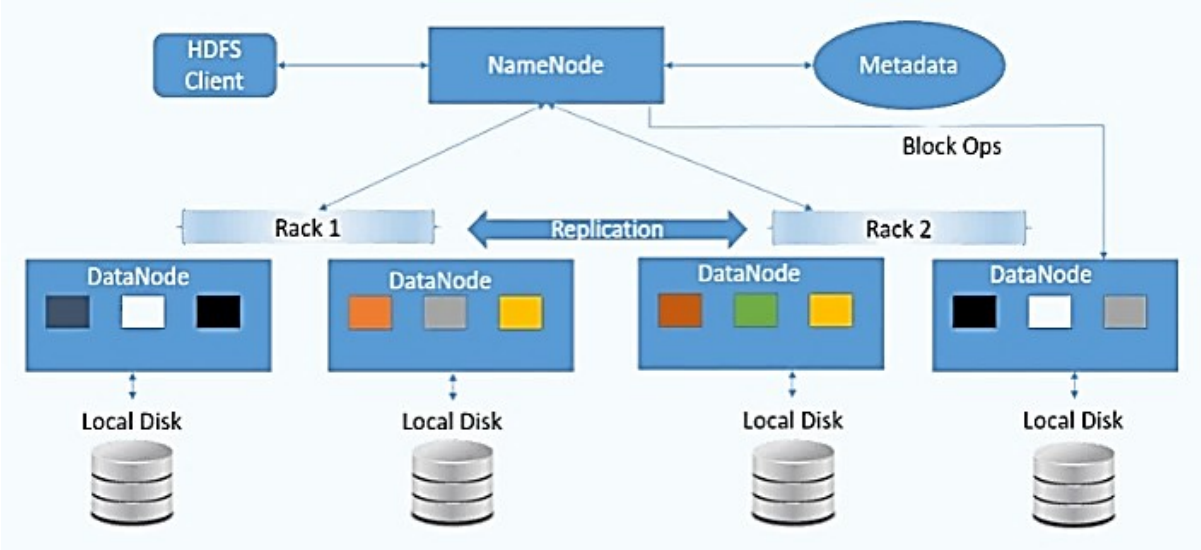


Fig. 6.2. HDFS architecture

HDFS breaks files into fixed-size blocks (typically 128 megabytes or 256 megabytes). Each block is typically replicated across different nodes in the cluster for reliability. This allows files to be distributed across multiple servers and processed in parallel.

All metadata associated with files is stored on a centralized server called

the NameNode. All other servers, known as DataNodes, contain the data itself, but not the metadata.

HDFS is a distributed file system built for scalability. We can add new nodes to the cluster as needed to increase the amount of data. If a node fails or loses data, other replicas of that data from DataNodes are used to recover the information. This provides high reliability and fault tolerance.

Data access in HDFS is done through distributed read and write operations. Data is distributed across nodes for parallel processing. Write operations are performed by creating new blocks or updating existing ones.

HDFS replicates data to ensure reliability. By default, each block is replicated on three different nodes. HDFS also has a load balancing mechanism that distributes data across nodes to maintain balance and optimize resource utilization.

HDFS is used in conjunction with Apache MapReduce to perform computations on large amounts of data. Since blocks are distributed, MapReduce can use the principle of "shuffling" to transfer data between nodes for in-place computation. HDFS supports data encryption and authentication methods to ensure data security.

6.3. Hadoop YARN

Hadoop YARN (Yet Another Resource Negotiator) is a Hadoop subsystem responsible for resource management and task scheduling on a cluster. The main purpose of YARN is to ensure efficient resource utilization and support big data computing on Hadoop clusters.

YARN can be considered as an operating system for a cluster.

Cluster is a collection of loosely or tightly coupled computers that work together as a single system. A cluster represents a set of resources, such as compute, memory, disk space, and network bandwidth, that YARN must distribute among the tasks running in the cluster. Similar to how an operating system manages machine resources and allocates them among competing processes, YARN distributes cluster resources among competing tasks. The two components operate in a master-slave relationship, where the resource manager (RM) is the master and the node manager is the slave. One resource manager operates in a cluster with one node manager per machine. Together, these two components form the data computing framework (Fig. 6.3).

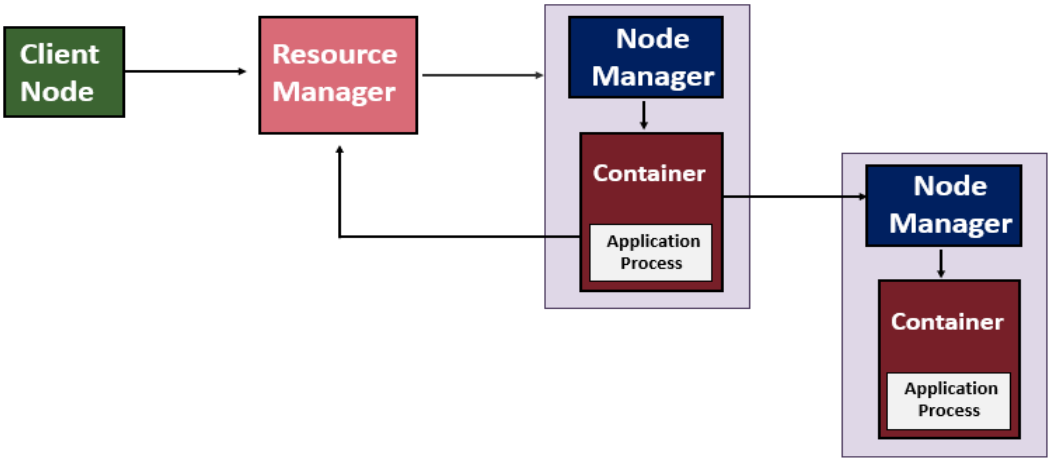


Fig. 6.3. HDFS architecture

The resource manager allocates resources among all programs in the system. The resource manager consists of two parts: the program manager and the scheduler.

The Application Manager is responsible for accepting task notifications and starting the container for the ApplicationMaster entity. It also restarts the ApplicationMaster container if the container crashes.

The scheduler is responsible for allocating resources such as disk, CPU, and running applications on the network, subject to constraints imposed by queues and capacity. The scheduler does not monitor applications or initiate restarts in the event of application or hardware failures.

In Unix, a container is a process, and in Linux, a control group. There is a build and teardown of tasks running inside a container. One machine in a cluster can have multiple containers.

The Resource Manager acts as a single point of failure. If the machine hosting the RM fails, jobs cannot be scheduled.

NodeManager is an agent that runs on each machine in the cluster, responsible for launching containers on that machine and managing the containers' resource usage. NodeManager reports usage to the resource manager's scheduler component.

A single node contains a set of CPU, RAM, and other CPU-intensive resources, such as containers . The lifecycle of YARN containers is managed by container launch contexts, and the resources are made available to applications on a given host.

The ApplicationMaster keeps a record of running applications and monitors their execution. Each time a job is submitted to the framework, an

application master is elected for it, which is responsible for distributing resources from the resource manager to the node manager, which then monitors and executes the job.

YARN manages resources on the cluster, taking into account the distributed nature of computing in a cluster environment, determines how much resources (CPU time, memory, etc.) are allocated to each task and node on the cluster.

YARN is responsible for scheduling and distributing computing tasks on the cluster nodes, and also takes into account the resource requirements of each task and assigns them to free nodes. YARN allows the use of additional components that can run on the cluster together with Hadoop MapReduce, such as Apache Tez, Apache Spark, etc. YARN contributes to increased performance because resources can be allocated only where they are needed, and tasks can be computed in parallel.

YARN also supports running a variety of tasks and computing frameworks, allowing to work with different types of data analysis applications.

YARN allows users to integrate various components and tools, such as Apache HBase, Apache Hive, Apache Pig, and many others, into the Hadoop ecosystem. YARN supports scalability, allowing to add new nodes to the cluster to increase performance, and also has fault tolerance, ensuring that the system continues to operate in the event of node failures.

Hadoop YARN plays an important role in resource management and computing scheduling on a cluster, allowing for efficient large-scale data computing on the Hadoop platform.

6.4. MapReduce paradigm

MapReduce paradigm is a computing model used to process and generate large amounts of data on distributed computing clusters. This paradigm was developed by Google and served as the basis for the Apache Hadoop data processing framework. The main components of the MapReduce paradigm are Map, Shuffle, Sort, and Reduce (Fig. 6.4). The diagram illustrates the MapReduce framework's workflow for processing large datasets. The process begins with "Input Data," which is divided into smaller chunks called splits. During the "Map phase," each split is processed independently by map tasks, producing intermediate key-value pairs. These

pairs are then shuffled and sorted to group data with the same keys. In the "Reduce phase," reducers aggregate and process the grouped data to generate the final "Output Data," which is stored as results. This approach enables parallel and distributed computation, making it efficient for handling big data.

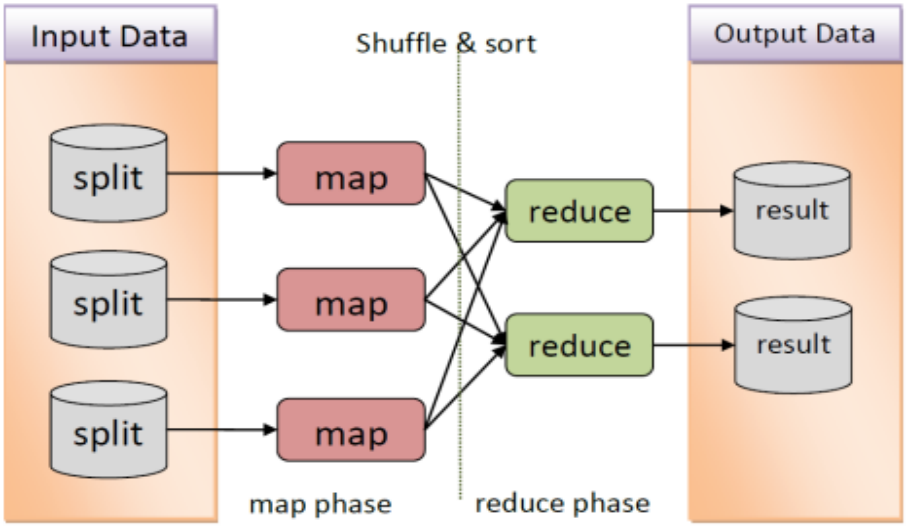


Fig. 6.4. MapReduce calculation

The first stage of Map involves executing the "Map" function. This function is given a large data set as input and splits it into small pieces for further processing. Each piece is processed separately, and the Map function is applied to each piece of data. It performs transformation or filtering of the data and generates key-value pairs.

Key-value pairs are collected and grouped by key. This allows data with the same key to be combined, which creates the opportunity for efficient data processing in the next step.

The final stage of Reduce involves executing the "Reduce" function. The Reduce function takes groups of data that match the same keys and processes them, and may also perform aggregation, calculations, or any other operations on the data. The results of the Reduce function are typically written to a large distributed data storage system.

The MapReduce paradigm has several important advantages:

1. scalability for processing large amounts of data by adding new nodes to the cluster;

2. fault tolerance for operation in distributed environments, which allows data processing to continue even if certain nodes fail.

Developing applications using this paradigm is usually quite simple. MapReduce is well suited for processing large amounts of structured data, such as server logs, hardware logs, etc. It is used in Apache Hadoop and many other data analysis systems.

6.5. Deploying Percona XtraDB cluster

Percona XtraDB Cluster is a tool for creating highly available databases with distributed storage and replication. It is an open solution for MySQL that integrates Percona Server and Percona XtraBackup with the Galera library to provide synchronous replication from multiple sources.

Configuring and maintaining a cluster requires a level of expertise in database administration. Deploying a Percona XtraDB Cluster is the process of creating a highly available database cluster based on Percona XtraDB, an open source alternative to MySQL and MariaDB. Percona XtraDB Cluster provides the ability to deploy horizontally scaled database clusters and provides high data availability.

Let's look at the basic steps for deploying Percona XtraDB Cluster.

Step 1. After selecting physical or virtual servers to create a cluster, we need to install an operating system that is supported by Percona XtraDB Cluster, for example, Ubuntu or CentOS. A cluster consists of nodes, where each node contains the same set of data synchronized between nodes.

The recommended configuration is at least three nodes, but we can also use use two nodes (Fig. 6.5).

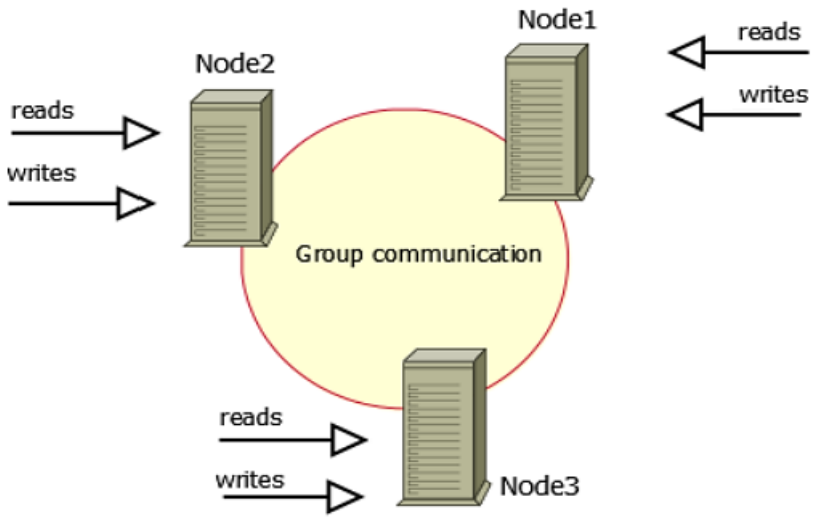


Fig. 6.5. Example of using three Percona XtraDB Cluster nodes

Each node is a regular MySQL Server instance (e.g., Percona Server). We can convert an existing MySQL Server instance to a node and run a cluster using that node as the base. We can also detach any node from the cluster and use it as a regular MySQL Server instance.

A Percona XtraDB cluster is based on a Percona server running XtraDB storage and using the Galera library, which is an implementation of the WSREP (Wsrep) API. The standard and recommended data transfer method is Percona XtraBackup.

Consider the advantages of Percona XtraDB.

- When we execute a query, it is executed locally on the node. All data is available locally, no remote access is required.
- There is no central management. If we lose any node at any point in time, the cluster will continue to function without data loss.
- Percona XtraDB is a good solution for scaling read workloads, meaning we can execute read requests to any of the nodes.

The downside of using Percona XtraDB is the overhead for each new node. When a new node is added, it must copy the entire data set from one of the existing nodes. If it is 100 GB, 100 GB is copied. Replication only works with the InnoDB storage engine. Any writes to other table types, including system (mysql.*) tables, are not replicated. The write throughput of the entire cluster is limited by the weakest node. If one node becomes slow, the entire cluster slows down.

Step 2. We need to install Percona XtraDB Cluster on each server. To do this, we can use the Percona repository or download the packages from the official website.

We must use the instructions to set configuration parameters, such as unique identifiers for each server and other parameters required for the cluster. The version number identifies the product release.

The product contains the latest generally available (GA) features at the time of release. The Percona XtraDB cluster assigns a set of unique numbers to each version in ascending order. The numbers are a combination of the Percona Server for MySQL version and internal version numbers that identify the build.

Percona uses semantic version numbering that follows the pattern of base version, minor build, and optional special build. Percona assigns unique non-negative integers in ascending order to each minor build release. The

version number combines the base version number of Percona Server for MySQL, the minor build version, and the special build version, if applicable.

The Percona XtraDB Cluster 5.7.40-31.63 version numbers define the following information.

Basic version – the left set of numbers indicates the version of Percona Server for MySQL that is used as the base. Incrementing the base version resets the minor build version and the special build version to zero.

Minor build version is an internal number that indicates the corresponding version. When this number is incremented by one, the special build fix is reset to 0.

Special build version is an optional number assigned to custom builds used to fix bugs. Software functionality, unless included in a bug fix, is not changed.

Step 3. To set up XtraDB Cluster, we need to install and configure `wsrep_cluster_name` – a unique name for the cluster, as well as `wsrep_node_address` and `wsrep_node_name` for each server.

We need to start Percona XtraDB Cluster on each server and make sure they can communicate with each other (Fig. 6.6).

pxc1	192.168.70.61
pxc2	192.168.70.62
pxc3	192.168.70.63

Fig. 6.6. Configuring three Percona XtraDB Cluster nodes

If we are using Debian or Ubuntu, we need to add the following configuration variables to /etc/percona-xtradb-cluster.conf.d/wsrep.cnf on the first node:

```
[mysqld]
wsrep_provider=/usr/lib/libgalera_smm.so
wsrep_cluster_name=pxc-cluster
wsrep_cluster_address=gcomm://192.168.70.61,192.168.70.62,192.168.70.63
wsrep_node_name=pxc1
wsrep_node_address=192.168.70.61
wsrep_sst_method=xtrabackup-v2
wsrep_sst_auth=sstuser:passw0rd
pxc_strict_mode=ENFORCING
binlog_format=ROW
default_storage_engine=InnoDB
innodb_autoinc_lock_mode=2
```

In the future, we need to use the same configuration for the second and third nodes, except for the `wsrep_node_name` and `wsrep_node_address variables`. For the second node:

```
wsrep_node_name=pxc2
wsrep_node_address=192.168.70.62
```

For the third node:

```
wsrep_node_name=pxc3
wsrep_node_address=192.168.70.63
```

The logical name for the cluster must be the same for all nodes in that cluster (`wsrep_cluster_name`). We must specify the IP addresses of the nodes in the cluster. It is recommended that specify the addresses of all nodes for a node to join the cluster. This way, if the first node in the list is unavailable, the joining node can use other addresses (`wsrep_cluster_address`). By default, a Percona XtraDB cluster uses Percona XtraBackup for State Snapshot Transfer (SST). It is recommended to set `wsrep_sst_method=xtrabackup-v2`. This method requires that an SST user be configured on the originating node.

Step 4. To synchronize data, we need to upload backups of database or copy data from one of the servers, start data synchronization between the cluster servers. After configuring all PXC nodes, we need to initialize the cluster by booting the first node. The initial node should contain all the data that we want to replicate to the other nodes. Booting involves starting the first node without any known cluster addresses: if the `wsrep_cluster_address variable` is empty, Percona XtraDB Cluster assumes that this is the first node and initializes the cluster.

The node is started using the `bootstrap.server command` in bootstrap mode with `wsrep_cluster_address=gcomm://`. This parameter tells the node to initialize the cluster with the `wsrep_cluster_conf_id variable` set to 1.

After adding other nodes to the cluster, we can restart this node as usual and it will use the default configuration again. To verify that the cluster has been initialized, we need to run the command:

```
mysql@pxc1> show status like 'wsrep%';
```

The output shows that the cluster size is 1 node, this is the primary component, the node is in the Synced state, it is fully connected and ready to replicate with a record set.

```
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_local_state_uuid     | c2883338-834d-11e2-0800-03c9c68e41ec |
| ...                        | ...                                  |
| wsrep_local_state          | 4                                    |
| wsrep_local_state_comment  | Synced                               |
| ...                        | ...                                  |
| wsrep_cluster_size         | 1                                    |
| wsrep_cluster_status       | Primary                              |
| wsrep_connected            | ON                                   |
| ...                        | ...                                  |
| wsrep_ready                | ON                                   |
+----------------------------+--------------------------------------+
40 rows in set (0.01 sec)
```

New nodes `<Node>` that are properly configured are provisioned automatically. When a node starts with the address of at least one other running node in the `wsrep_cluster_address` variable, it automatically joins the cluster and synchronizes with it.

By default, a Percona XtraDB cluster uses Percona XtraBackup to transfer snapshots (SST). To do this, we need to set `wsrep_sst_method` for variable value `xtrabackup-v2` and provide the SST user credentials along with the `wsrep_sst_auth` variable.

The second node is started using one of the following commands:

```
[root@pxc2 ~]# /etc/init.d/mysql start
```

or

```
[root@pxc2 ~]# systemctl start mysql
```

After startup, the server should receive SST automatically.

To check the status of the second node, we need to run the command:

```
mysql@pxc2> show status like 'wsrep%';
```

The result shows that the new node has been successfully added to the cluster. The cluster size is now 2 nodes, it is the primary component, it is fully connected and ready to accept replication with a record set.

```
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_local_state_uuid     | c2883338-834d-11e2-0800-03c9c68e41ec |
| ...                        | ...                                  |
| wsrep_local_state          | 4                                    |
| wsrep_local_state_comment  | Synced                               |
| ...                        | ...                                  |
| wsrep_cluster_size         | 2                                    |
| wsrep_cluster_status       | Primary                              |
| wsrep_connected            | ON                                   |
| ...                        | ...                                  |
| wsrep_ready                | ON                                   |
+----------------------------+--------------------------------------+
40 rows in set (0.01 sec)
```

To add a third node, we need to start the node using one of the commands:

```
[root@pxc3 ~]# /etc/init.d/mysql start
```

or

```
[root@pxc3 ~]# systemctl start mysql
```

To check the status of the third node, we need to run the command :

```
mysql@pxc3> show status like 'wsrep%';
```

The result shows that the new node has been successfully added to the cluster. The cluster size is now 3 nodes, it is the primary component, it is fully connected and ready for replication with a record set.

```
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_local_state_uuid     | c2883338-834d-11e2-0800-03c9c68e41ec |
| ...                        | ...                                  |
| wsrep_local_state          | 4                                    |
| wsrep_local_state_comment  | Synced                               |
| ...                        | ...                                  |
| wsrep_cluster_size         | 3                                    |
| wsrep_cluster_status       | Primary                              |
| wsrep_connected            | ON                                   |
| ...                        | ...                                  |
| wsrep_ready                | ON                                   |
+----------------------------+--------------------------------------+
40 rows in set (0.01 sec)
```

If the second node's status is Synced, then the node has received a full SST, is synchronized with the cluster, and can proceed to add the next node. After all nodes have been added to the cluster, we can test replication by executing queries and manipulating data on the nodes to verify that these changes are synchronized across the cluster.

To test replication, we need to create a new database on the second node:

```
mysql@pxc2> CREATE DATABASE percona;
```

The following results confirm that the new database has been created:

```
Query OK, 1 row affected (0.01 sec)
```

We need to go to the newly created database:

```
mysql@pxc3> USE percona;
```

The following results confirm that the database has been modified:

```
Database changed
```

We need to create a table on the third node:

```
mysql@pxc3> CREATE TABLE example (node_id INT PRIMARY KEY, node_name VARCHAR(30));
```

The following output confirms that the table was created:

```
Query OK, 0 rows affected (0.05 sec)
```

Consider an example of inserting records into the first node:

```
mysql@pxc1> INSERT INTO percona.example VALUES (1, 'percona1');
```

The following output confirms that the records were inserted:

```
Query OK, 1 row affected (0.02 sec)
```

We get the rows from this table on the second node:

```
mysql@pxc2> SELECT * FROM percona.example;
```

The following output confirms that all rows were received:

```
+---------+-----------+
| node_id | node_name |
+---------+-----------+
|       1 | percona1  |
+---------+-----------+
1 row in set (0.00 sec)
```

A recommended high availability solution for a Percona XtraDB cluster is to consider installing ProxySQL on client nodes to efficiently manage workloads across the cluster without any changes to the applications that generate queries.

Step 5. For monitoring and management, we need to install and configure the monitoring tools provided by Percona XtraDB Cluster, such as Percona Monitoring and Management (PMM) to track performance, health, and issues in the cluster.

Anyone with network access can connect to any Percona XtraDB Cluster node, either as a client or as another node joining the cluster.

Therefore, we should restrict access using VPN and filter traffic on the ports used by Percona XtraDB Cluster.

Unencrypted traffic can potentially be viewed by anyone monitoring the network. To prevent this, we should enable encryption on all nodes in the cluster. The Percona XtraDB cluster supports tablespace encryption to provide encryption at rest for physical space data files.

The firewall can allow Percona XtraDB Cluster traffic to be filtered based on clients and nodes. By default, Percona XtraDB cluster nodes use the following ports:

3306 – used for MySQL client connections and SST (State Snapshot Transfer) via mysqldump;

4444 – used for SST via rsyncPercona XtraBackup;

4567 – used for recordset replication traffic (over TCP) and multicast replication (over TCP and UDP);

4568 – used for IST (Incremental State Transfer).

We need to ensure that these ports on each node are only accessed from trusted IP addresses. We can do this by using packet filtering.

There are two types of traffic in a Percona XtraDB cluster:

1. client-server traffic (between client applications and cluster nodes);
2. replication traffic, which includes SST, IST, write set replication, and various service messages.

Percona XtraDB Cluster supports encryption for all types of traffic. Encryption of replication traffic can be configured in both automatic and manual modes. Percona XtraDB Cluster uses the underlying MySQL encryption mechanism to protect communication between client applications and cluster nodes.

Replication traffic refers to the traffic between nodes, which includes SST traffic, IST traffic, and replication traffic. Each type of traffic is transmitted over different channels, so it is important to configure secure channels for all variants to fully secure replication traffic. Starting with version 5.7, PXC supports a single configuration parameter that helps ensure full replication traffic, and is often referred to as auto-configuration. The user can also override this and configure the security of each channel by specifying independent parameters.

The Percona XtraDB cluster includes a variable `pxc-encrypt-cluster-traffic` that allows to automatically configure encryption for SST, IST, and replication traffic there. This variable is not dynamic, so it cannot be changed at runtime.

Key and certificate files are required to automatically configure SSL encryption. Starting with version 5.7, MySQL generates key and certificate files by default and places them in the data directory. Backup and Restore is used when we need to recover data after an accidental deletion, update, or critical error. Incremental Backup is used to efficiently save and restore changed data. After a full backup, we should create incremental backups that contain only the changed data. This helps reduce restore time and the amount of backup storage.

Point-in-time recovery is used when we need to restore data to a specific point in time. Sometimes we need to restore a database to a specific state that existed at a specific point in time, such as before a critical error.

Hot node replacement is used when a node in a cluster fails. If one of the cluster nodes fails, it can be replaced by another without interrupting the cluster. This is an important scenario for ensuring high availability.

Replica reassignment is used to change the distribution of replicas in a cluster, replicas are moved between nodes to optimize resource utilization and node load.

Replication is used to provide data redundancy and high availability. Replication allows users to create multiple copies of data on different nodes for fault tolerance. As data volume and workload grow, we need to scale cluster by adding new servers.

Percona XtraDB Cluster is a high-performance solution for ensuring fault tolerance and scalability of MySQL databases, supporting synchronous replication, ensuring data is up-to-date on all cluster nodes, even in the event of a failure of one of them.

With automatic load balancing mechanisms, the cluster efficiently distributes requests across nodes, improving performance. The tool also provides protection against data loss through automatic recovery mechanisms and flexible backup capabilities. The combination of these advantages makes Percona XtraDB Cluster an ideal choice for systems that require reliable storage and real-time data processing, such as Big Data systems or online transactional platforms.

Combining Percona XtraDB Cluster with Big Data platforms such as Hadoop enables efficient management of both structured and unstructured data. These technologies create a powerful foundation for building a big data infrastructure that is flexible, scalable, and reliable. Their integration allows organizations to process and analyze huge amounts of data in real time, supporting complex business processes and strategic decisions.

Control questions and tasks for topic 6

List of theoretical questions

1. What is the architectural structure of Hadoop and what components are included in it?
2. How does the Hadoop Distributed File System (HDFS) enable reliable and distributed data storage?
3. How does Hadoop YARN manage resource allocation on a cluster?
4. What capabilities does Hadoop YARN provide for working with big data resources?
5. How does the MapReduce paradigm work in Hadoop, and what are the "Map" and "Reduce" phases?
6. What are the advantages of using the MapReduce paradigm for processing large amounts of data?
7. How can Hadoop be used for real-time computing?
8. What tools and programming languages are supported in the Hadoop environment?
9. How are data security and access issues addressed in Hadoop?
10. What data backup and recovery capabilities are present in Hadoop?
11. What are the benefits of deploying the Hadoop distributed computing system for big data?
12. What are the benefits and features of Percona XtraDB Cluster, and what makes it important for deploying a database system?
13. What are the steps involved in deploying Percona XtraDB Cluster, and what should be considered during this process?
14. How can you achieve high availability and horizontal scalability with Percona XtraDB Cluster?

List of tasks for independent work

Practical task 1. Web scraping with Selenium to automate interaction with a web form.

Objective: use Selenium to fill out a web form, submit it, and scrape the results.

Step 1. Install Selenium using pip if it is not already installed:

```
! pip install selenium
```

Download the appropriate web driver for your browser (e.g., ChromeDriver for Chrome). Ensure the driver is in your system PATH or specify its location in the script.

Step 2. Write the Python code.

Create a script to automate filling out a form and scraping the results:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
# Set up the web driver
driver = webdriver.Chrome()
# Navigate to the web page
url = "https://example.com/form-page"
driver.get(url)
# Fill out the form
name_field = driver.find_element(By.ID, "name")
name_field.send_keys("John Doe")
email_field = driver.find_element(By.ID, "email")
email_field.send_keys("john.doe@example.com")
comment_field = driver.find_element(By.ID,
"comment")
comment_field.send_keys("This is a test comment.")
# Submit the form
submit_button = driver.find_element(By.ID, "submit")
submit_button.click()
# Wait for the results to load
time.sleep(3)
# Scrape the results
result = driver.find_element(By.ID, "result").text
```

```
print("Form submission result:", result)
# Close the browser
driver.quit()
```

Step 3. Execute the script. Run the Python script and observe the output. Ensure the website is accessible and the element IDs match the form fields on the target web page. Verify that automated interaction with the website complies with its terms of service. Use explicit waits (e.g., Selenium's WebDriverWait) for more reliable element interaction.

Practical task 2. Real-time Twitter sentiment analysis using Tweepy and TextBlob.

Objective: stream tweets using the Twitter API, analyze their sentiment, and classify them as positive, neutral, or negative.

Step 1. Install Required Libraries. Install the required Python libraries:

```
! pip install tweepy textblob
```

Step 2. Authenticate with the Twitter API. Create a Twitter Developer account, set up an app, and obtain API keys and tokens. Use the following Python code to authenticate:

```
import tweepy
from textblob import TextBlob
# Twitter API credentials
consumer_key = "your_consumer_key"
consumer_secret = "your_consumer_secret"
access_token = "your_access_token"
access_token_secret = "your_access_token_secret"
# Authenticate
auth = tweepy.OAuth1UserHandler(consumer_key,
consumer_secret, access_token, access_token_secret)
api = tweepy.API(auth)
```

Step 3. Write a script to stream tweets and analyze their sentiment:

```
class MyStreamListener(tweepy.StreamListener):
    def on_status(self, status):
        # Get the tweet text
        tweet_text = status.text
        print("Tweet:", tweet_text)
         # Perform sentiment analysis
        analysis = TextBlob(tweet_text)
```

```
                sentiment = analysis.sentiment.polarity
                if sentiment > 0:
                    print("Sentiment: Positive")
                elif sentiment == 0:
                    print("Sentiment: Neutral")
                else:
                    print("Sentiment: Negative")
                print("-" * 50)
                def on_error(self, status_code):
                if status_code == 420:  # Rate limit exceeded
                    return False
    # Stream tweets containing a specific keyword
    stream_listener = MyStreamListener()
    stream = tweepy.Stream(auth=api.auth,
listener=stream_listener)
    stream.filter(track=["Python"], languages=["en"])
```

Step 4. Execute the script. Run the script to start streaming tweets containing the keyword "Python" and classify their sentiment in real-time.

Adhere to Twitter's API usage policy, including rate limits and acceptable use guidelines. Ensure sensitive credentials like API keys are stored securely, e.g., in environment variables.

List of used and recommended literature

6.1. Apache Hadoop [Electronic resource]. – Access mode: https://hadoop.apache.org/

6.2. Download - Apache Hadoop [Electronic resource]. – Access mode: https://hadoop.apache.org/releases.html

6.3. HDFS Architecture [Electronic resource]. – Access mode: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

6.4. Apache Hadoop YARN [Electronic resource]. – Access mode : https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html

6.5. MapReduce [Electronic resource]. – Access mode : https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

6.6. Percona XtraDB Cluster [Electronic resource]. – Access mode : https://docs.percona.com/

Topic 7. Streaming data analytics technologies

Stream analytics technologies include a set of tools and platforms that allow to collect, process, and analyze data in real time. Among them, Apache Kafka, which provides scalable and reliable data transfer between systems, Apache Flink and Apache Storm, which process data with low latency and high throughput, and Apache Samza, which integrates with Kafka for efficient stream processing. These technologies allow businesses to obtain operational analytics and make decisions based on up-to-date information, which is especially important for areas such as financial markets, IoT, and network monitoring.

Big data pipeline is a sequence of processing and transferring large amounts of data from one stage of analysis or processing to the next in order to obtain a meaningful conclusion or result. This aspect of big data analytics allows users to organize the data processing and analysis processes, reducing the cost of time and resources. The main components of a big data pipeline include data collection, data transfer, data processing, data analysis, and result storage.

The first step is to collect data from various sources such as sensors, server logs, data streams, etc. The data is transferred to a processing environment such as a computing cluster. Data processing includes cleaning, transformation, aggregation, and other operations to prepare the data for analysis. Then, data analysis algorithms are incorporated, including machine learning, statistical analysis, and other techniques to derive meaningful conclusions. The analysis results are stored for later use and further analysis.

Data ingestion and processing in a big data pipeline may encounter the following challenges.

1. Large volumes of data can lead to the use of large computing resources and the need for their efficient use.

2. Data comes in in real time and need to be processed instantly.

3. Data can be of different quality and in different formats, which requires processing and unification.

4. If data processing is distributed across multiple nodes or servers, it is important to ensure their synchronization and coordination.

5. Ensuring data security, confidentiality, and integrity at all stages of the big data pipeline is an important task.

6. Large volumes of data can include errors and other problems, and handling these situations is an important part of the big data pipeline .

If one of the problems is the speed of data transmission over the network, we can set a higher network bandwidth to ensure more efficient data reception. It is important to have buffers that can temporarily store data that arrives faster than it can be processed and process it later. If we need to process data in real time, we can use specialized technologies for processing streaming data, such as Apache Kafka or Apache Flink. Large amounts of data can be compressed, which allows users to reduce the amount of data that is transmitted over the network.

Data duplication and backup can ensure data is preserved in case of loss during transmission. When processing data, optimized algorithms and data structures can be used to reduce processing time. If multiple data processing nodes are unevenly loaded, load balancing helps distribute the workload evenly.

Monitoring and error analysis systems allow to detect and correct data ingestion problems in a timely manner. The use of high-performance data warehouses and databases can improve data access. Breaking a large data stream into smaller segments can make it easier to ingest and process. In general, solving data ingestion problems in a Pipeline requires a comprehensive approach that takes into account the specifics of the specific system and data processing requirements.

7.1. Kafka distributed streaming platform

Apache Kafka distributed streaming platform is a streaming data exchange system that was designed to process large streams of data in real time. Kafka is a scalable system that allows users to expand its storage and processing capabilities for streaming data, as well as add new brokers (servers) to the cluster to handle large amounts of data and increase throughput. Kafka provides reliable data storage and replication. Messages that arrive in Kafka can be stored for a certain period of time to ensure their availability. In addition, data can be replicated to different brokers to provide reliability and recovery in the event of a failure.

Kafka provides many additional features, such as distributed topics, event processing, data streaming, integration with other technologies for processing and analyzing streaming data. The Kafka platform is designed for

real-time data processing at high speed and can process a large number of messages in a short time. A Kafka cluster can include a large number of brokers and clients. Kafka supports real-time processing of streaming data and integration with other systems for data processing and analysis. Kafka has an active developer community and a large number of resources for training and support.

Apache Kafka is a distributed event store and stream processing platform, an open source system written in Java and Scala , developed by LinkedIn and a project of the Apache Software Foundation. The project aims to provide a unified, high-performance, low-latency platform for processing real-time data streams.

Kafka can connect to external systems (for data import/export) via Kafka Connect and provides the Kafka Streams libraries for stream processing applications. Kafka uses a binary TCP-based protocol that relies on a "message set" abstraction that naturally groups messages together to reduce network transmission overhead.

Kafka was developed and introduced in early 2011. Apache Kafka is based on a commit log, allowing users to subscribe to Kafka and publish data to any number of systems or applications in real time. Examples of applications using the Kafka platform include managing ridership and driver matching at Uber, providing real-time analytics and predictive maintenance for British Gas' smart home , and performing real-time services at LinkedIn.

Kafka stores key-value messages that come from any number of processes (producers) . Data can be divided into different “partitions ” within different “topics”. Within a partition, all messages are strictly ordered by their offset (the position of the message in the partition), indexed, and stored with a timestamp. Other processes, called “consumers”, can read messages from partitions (Fig. 7.1). Producers create and send data to a Kafka cluster, which is organized around topics. Each topic is divided into partitions, which allows for parallel processing of data to achieve high throughput.

Consumers receive data from parts of topics in the cluster, providing asynchronous and scalable information processing. This architecture is used for real-time streaming data processing systems, including event logs, analytics, and distributed systems integration.

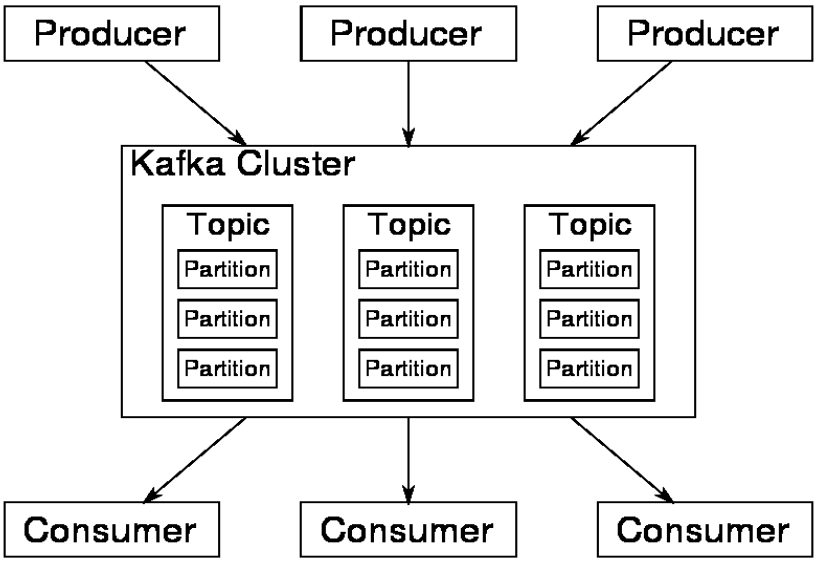


Fig.7.1. Apache Kafka Architecture: interaction of Producers, Topics, and Consumers

For stream processing, Kafka offers the Streams API, which allows users to create Java applications that consume data from Kafka and write results to Kafka. Apache Kafka also works with external stream processing systems such as Apache Apex, Apache Beam, Apache Flink, Apache Spark, Apache Storm, and Apache NiFi.

Kafka runs on a cluster of one or more servers (called brokers), and partitions of all topics are distributed across the nodes of the cluster. In addition, partitions are replicated across multiple brokers. This architecture allows Kafka to deliver large streams of messages in a fault-tolerant manner and replace some common messaging systems such as Java Message Service (JMS) and Advanced Message Queuing Protocol (AMQP). Starting with 0.11. In the 0.0 release, Kafka offers transactional records that provide one-time processing of a stream using the Streams API.

Kafka supports two types of topics: regular and compressed. Regular topics can be configured with a retention time or space limit. If there are records older than the specified retention time, or if the space limit for a partition is exceeded, Kafka can delete old data to free up storage space. By default, topics are configured with a retention time of 7 days, but data can also be stored indefinitely. The expiration of records in compressed topics is not affected by time or space constraints. Instead, Kafka treats later messages as updates to previous messages with the same key and guarantees that it will never delete the last message for a key. Users can delete messages entirely by writing a so-called tombstone message with a null value for a given key.

Kafka has five main APIs: Producer API, Consumer API, Connector API, Streams API, and Admin API. The Producer API allows an application to publish streams of records. The consumer API allows an application to subscribe to topics and process streams of records. The Connector API implements reusable producer and consumer APIs that can link topics to existing applications . The Streams API converts input streams into output and creates a result. The Admin API is used to manage Kafka topics, brokers, and other Kafka objects. The consumer and producer APIs are decoupled from the core Kafka functionality using a basic messaging protocol. This allows for the creation of compatible API layers in any programming language that are as efficient as the Java APIs shipped with Kafka. The Apache Kafka project maintains a list of such third-party APIs.

Kafka Connect (or Connect API) is a platform for importing/exporting data from/to other systems, using an internal producer and consumer API. The Connect framework itself implements connections that implement the actual logic for reading/writing data from other systems. The Connect API defines the programming interface that must be implemented to create a custom connector. Kafka Streams (or Streams API) is a stream processing library written in Java. It was added in the Kafka 0.10.0.0 release. The library allows users to develop stateful stream processing applications that are scalable and fully fault-tolerant. The core API is a domain-specific stream processing language (DSL) that offers high-level operators such as filter, map, group, window, aggregation, union, and table concepts. In addition, the stream API can be used to implement special operators for a lower-level development approach. DSLs and stream APIs can also be mixed. For stateful stream processing, Kafka Streams uses RocksDB to maintain local operator state. Since RocksDB can write to disk, the state maintained can be larger than the available main memory. To provide fault tolerance, all updates to local stores are also written to a topic in the Kafka cluster. This allows users to recreate the state by reading from these topics and sending all data to RocksDB.

End-to-end performance monitoring requires tracking metrics from brokers, consumers, and producers in addition to monitoring ZooKeeper, which Kafka uses to coordinate between consumers. The Kafka platform is used to build powerful big data analytics applications where high throughput and low latency are required, such as in financial systems, social networks, logistics, and telecommunications systems.

7.2. Apache Spark platform

Apache Spark is an open technology and platform for processing and analyzing large amounts of data that provides a distributed computing model that simplifies the development of data analysis applications.

The main features of Spark technology include high data processing speed, support for various data sources, real-time operation, and ease of use.

Spark was launched in 2009 at the University of California, Berkeley as a research project. It was founded by Matej Zacharia. In 2013, Spark joined the Apache Incubator, a project under the Apache Software Foundation. This marked its position as an open source project and gave it the opportunity to expand its developer community. In 2014, the project received Top-Level Apache status, which increased its importance in the open source world. This was an important event because it meant more support and trust from users.

In 2014, Spark introduced the MLlib libraries for machine learning and GraphX for graph processing. Over the following years, Spark technology quickly became very popular. Large companies and organizations began to use Spark for processing large amounts of data and machine learning.

The project regularly releases new versions, including improvements and new features. Spark continues to develop and attract new users.

Apache Spark is one of the leading tools for data processing and machine learning, which is popular all over the world and has an active global community and support from large IT companies.

Spark provides a distributed computing model that allows users to distribute calculations across a large number of cluster nodes. This allows users to process large amounts of data quickly and efficiently (Fig. 7.2) .

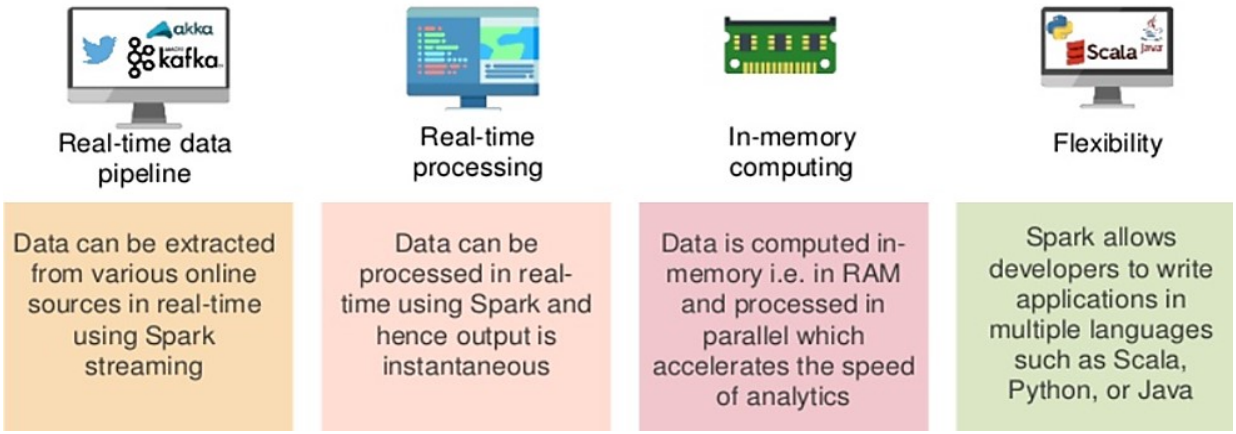


Fig. 7.2. Features of Apache Spark

Resilient Distributed Dataset (RDD) is the core abstract data object in Spark that allows for distributed data operations. RDD automatically provides recovery in the event of data loss, making Spark a more robust system.

Spark has APIs for application development, including Scala, Java, Python, and R. This allows developers to use the programming language of their choice. Spark has the MLlib library for developing machine learning models and data analysis. This allows users to perform analytical tasks such as classification, clustering, and regression. With the introduction of the Spark Deep Learning library, Spark can be used to develop and train deep neural networks. Spark Streaming allows users to process data in real time, which is important for applications that require rapid response to events.

Spark integrates with various data sources such as Apache Hadoop, Apache HBase, Apache Hive, Apache Cassandra, and many others. Spark provides an interactive interface for developing and debugging applications and provides a wide set of data processing operations, including filtering, sorting, joining, and many others.

Spark applications run as independent sets of processes in a cluster, coordinated by a SparkContext object in the main application (called the driver application). In particular, to run on a cluster, SparkContext can connect to several types of cluster managers (either Spark's own standalone cluster manager or Mesos/YARN), which allocate resources between applications (Fig. 7.3).

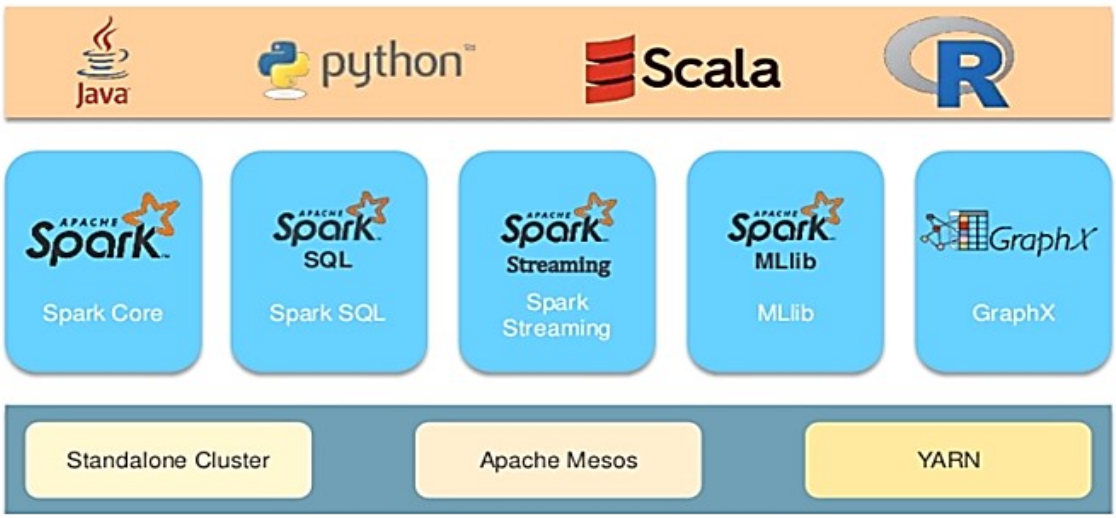


Fig.7.3. The Apache Spark ecosystem

Once connected, Spark gets executors on nodes in the cluster, which are processes that perform computations and store data for the application. Spark then sends the application code (defined by JAR or Python files passed to SparkContext) to the executors. SparkContext sends the tasks to the executors for execution (Fig. 7.4).

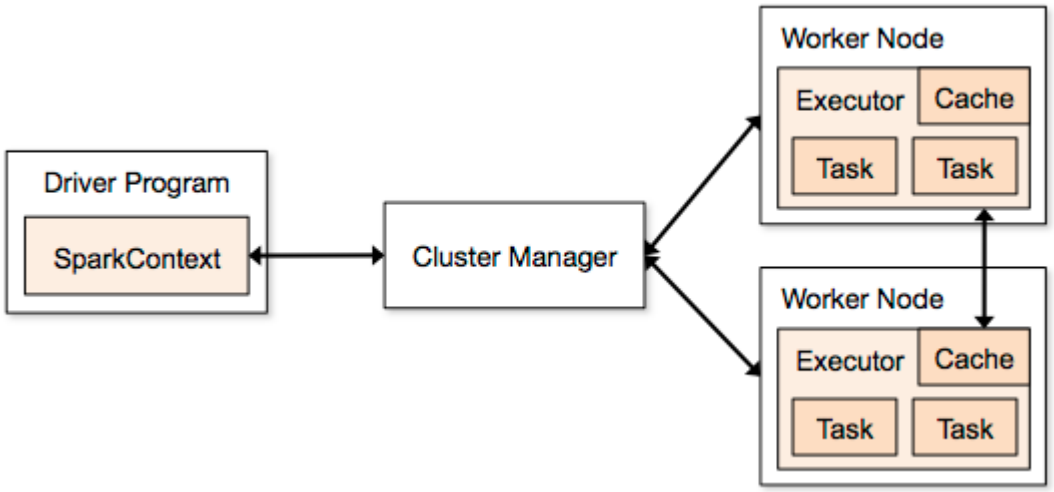


Fig.7.4. Executing tasks in Apache Spark

Each application gets its own executor processes, which remain active throughout the application and execute tasks in multiple threads. This has the advantage of isolating the applications from each other on both the scheduling side (each driver schedules its own tasks) and the executor side. It also means that data cannot be exchanged between different Spark applications (SparkContext instances) without writing it to external storage.

Spark does not depend on the underlying cluster manager, which also supports other applications (e.g. Mesos/YARN).

RPC (Remote Procedure Call) is a mechanism that allows programs to call functions or procedures on a remote server as if these functions were called locally, on the same server. In the case of a cluster, RPC can be used to communicate between different nodes of the cluster.

Because the driver schedules tasks in the cluster, it should be run near the worker nodes, preferably on the same local network. If we need to send requests to the cluster remotely, it is better to open an RPC for the driver to send the operation nearby than to run the driver far from the worker nodes. Users can call functions or methods on cluster nodes that may be located in

different locations on the network. The user can pass parameters to the remote function if necessary for its execution.

After calling a remote function, the user receives the results of the function, which can be used on the local node or sent to other nodes in the cluster. Remote functions can call other remote functions, which allows users to create complex logical procedures on different nodes.

A cluster RPC system typically provides mechanisms for determining which cluster node to execute a remote function on and for distributing calls across nodes. Using cluster RPC can be useful in computing where tasks need to be distributed across different cluster nodes and for interoperability between them. This helps improve the performance and scalability of applications in a clustered environment.

Apache Spark MLlib (Machine Learning Library) is a machine learning library that is part of the Apache Spark project and is designed for developing analytical applications (Fig. 7.5). Spark MLlib provides a wide range of machine learning algorithms and tools for working with them.

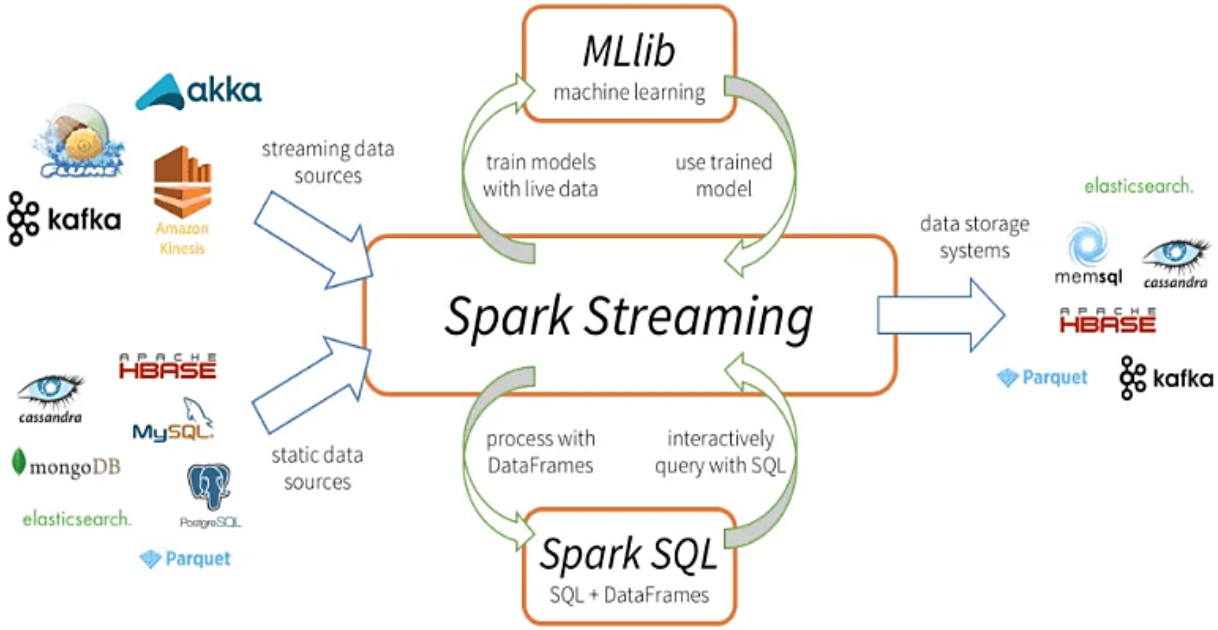


Fig. 7.5. Technologies for Apache Spark module interaction

MLlib contains many popular machine learning algorithms such as classification, regression, clustering, recommendation, text processing, as well as support for statistical methods.

MLlib is integrated with Apache Spark, which allows users to use distributed computing to process large amounts of data. This allows users to speed up model training and data processing.

MLlib has an easy-to-use API that allows developers to use machine learning algorithms without much effort.

MLlib supports various programming languages, including Scala, Java, Python, and R. This makes it accessible to developers with different preferences.

The MLlib library is designed with scalability in mind, allowing to work with datasets of various sizes, from small to large, making it suitable for many large projects.

MLlib supports saving and loading machine learning models, allowing to save trained models and use them later without the need for retraining.

MLlib can use graph computing for data processing and analysis, including for deep learning.

MLlib integrates with other components of the Apache Spark ecosystem and other data processing and analytics tools.

Apache Spark Streaming is a module of the Apache Spark library designed for processing and analyzing streaming data in real time. Spark Streaming allows users to implement streaming data analysis, process and aggregate data from various sources, and perform real- time calculations.

Spark Streaming supports input from a variety of sources, such as Apache Kafka, Apache Flume, Twitter, HDFS, and TCP/IP data streams. Spark Streaming also supports native solutions for real-time input processing.

Spark Streaming uses the concept of micro-batching, where input data is divided into small chunks that are processed at specific time intervals (e.g., every second). This allows for real-time processing and analysis of streaming data (Fig. 7.6).

Fig. 7.6. Splitting input data into micro-packets and their further processing

A discretized stream or DStream is the core abstraction provided by Spark Streaming and is a continuous stream of data received from a source, or a stream of processed data created by transforming an input stream. Internally, a DStream is represented by a continuous series of RDDs, which are Spark's abstraction of an immutable distributed dataset. Each RDD in a DStream contains data from a specific interval, as shown in Fig. 7.7.

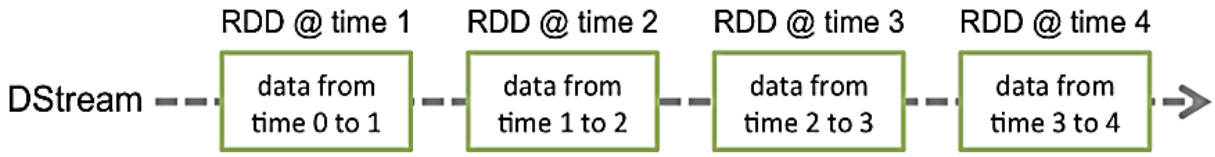


Fig. 7.7. DStream data stream

Any operation applied to a DStream translates to operations on the underlying RDDs. For example, in the previous example of converting a stream of strings to words, the flatMap operation is applied to each RDD in the DStream of strings to generate an RDD of DStream of words (Fig.7.8).

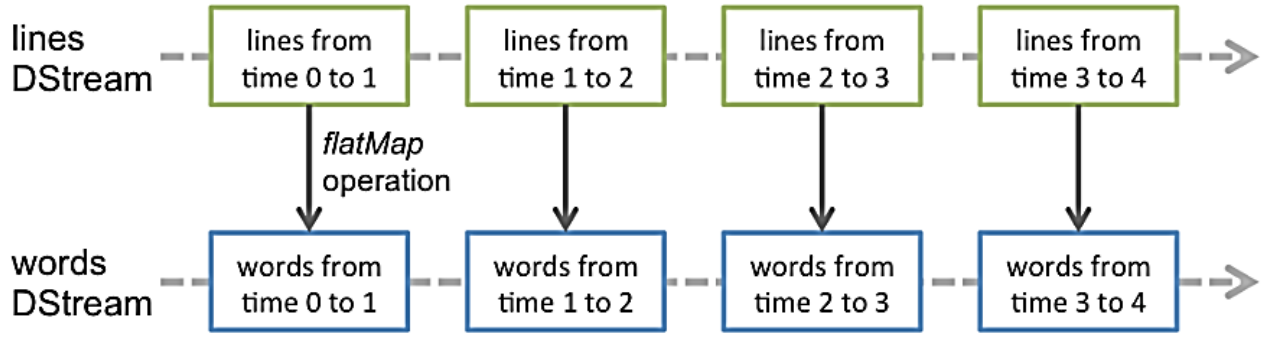


Fig. 7. 8. Example of converting a stream of strings into words

Spark Streaming is integrated with Apache Spark, allowing to use machine learning and data analysis features from Apache Spark MLlib.

Spark Streaming defines time windows within which data processing and analysis should be performed. This allows users to perform operations on data over a specific period of time. Spark Streaming allows users to store the results of streaming data processing in various sources such as HDFS, databases, Apache Kafka, etc. Scaling Spark Streaming occurs by adding more Spark cluster nodes to process large amounts of streaming data.

Apache Spark GraphX is a module of the Apache Spark library specifically designed for processing and analyzing graph structures in a distributed environment. GraphX allows users to develop and perform analytical tasks on graphs with a large number of nodes and edges.

Count is a mathematical structure consisting of a set of objects in which some pairs of objects are related to each other in some way. These relationships can be represented by edges and vertices that form a graph. The vertices represent the objects, and the edges show the various relationships between these objects. In computer science, a graph is is an abstract data type that is intended to implement the concepts of undirected and directed graphs in mathematics, particularly in graph theory. A graph data structure can also associate with each edge a specific edge value, such as a symbolic label or a numeric attribute (cost, capacity, length). Graphs can be used to detect disasters such as hurricanes, earthquakes, tsunamis, forest fires, and volcanoes to warn people.

PageRank is an algorithm designed to assess the importance of web pages based on the analysis of the graph of hyperlinks between them. It is performed by iteratively calculating the "rating" of each page, which depends on the number and quality of links to it, taking into account the weight of the pages being linked. The PageRank algorithm is closely related to stream processing and graphs, since the web graph is a dynamic, constantly changing stream of data. To update the page ranking in real time, it is necessary to use algorithms for stream processing and large graphs, for example, using Apache Spark or Apache Flink. These tools allow for efficient distribution of computations for large graphs, where each vertex (web page) and edge (link) is processed as part of the data stream, which provides scalability and high performance.

Graph analysis can be used to monitor financial transactions and identify people involved in financial fraud. Graphs, when used in conjunction with machine learning, help to understand customer purchasing trends. For example, these technologies are used by Uber and McDonald's. Graphs are also used to develop geographic information systems functionality for weather forecasting.

GraphX extends Spark RDD with Resilient Distributed Property Graph. A property graph is a directed multigraph that can have multiple parallel edges. Each edge and vertex have user-defined properties. Spark GraphX

combines ETL (Extract, Transform & Load), exploratory analysis, and iterative graph computation in a single system, allowing to view the same data as graphs and collections, efficiently transform and merge graphs with RDDs, and create own iterative graph algorithms using the Pregel API. Consider a simple graph of user connections on the social network Twitter (Fig. 7.9).

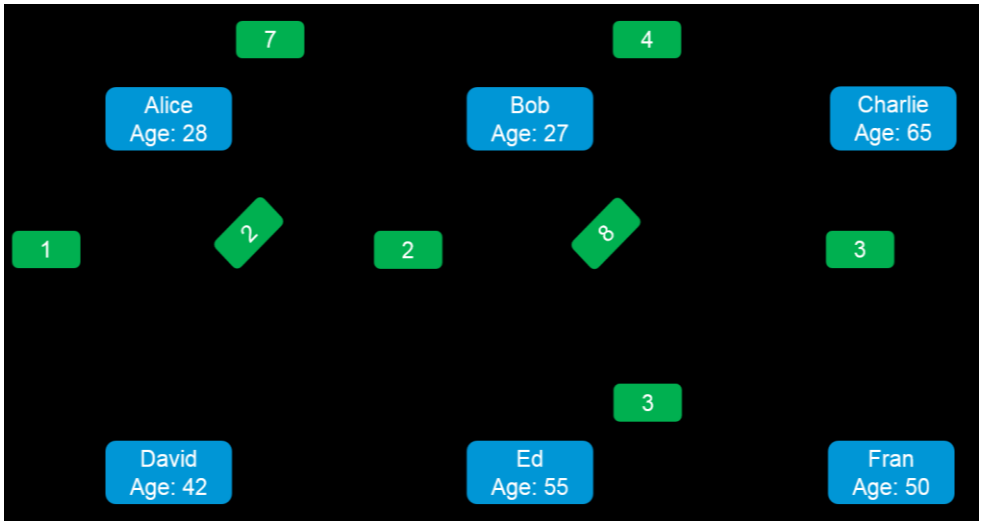


Fig. 7.9. Example of a graph Twitter user connections

By looking at a graph, we can get information about people (vertices) and the connections between them (edges). The graph here represents Twitter users and those they follow on Twitter. For example, Bob follows David and Alice. To programmatically implement this example using Apache Spark, we first import the necessary classes for GraphX:

```
import org.apache.spark._
import org.apache.spark.rdd.RDD
import org.apache.spark.util.IntParam
import org.apache.spark.graphx._
import org.apache.spark.graphx.util.GraphGenerators
```

Next, we will display all the names and ages of the users (vertices):

```
val vertexRDD: RDD[(Long, (String, Int))] = sc.parallelize(vertexArray)
val edgeRDD: RDD[Edge[Int]] = sc.parallelize(edgeArray)
val graph: Graph[(String, Int), Int] = Graph(vertexRDD, edgeRDD)
graph.vertices.filter { case (id, (name, age)) => age > 30 }
.collect.foreach { case (id, (name, age)) => println(s"$name is $age")}
```

Result The execution of the above code is as follows:

```
David is 42 years old,
Fran is 50 years old,
Ed is 55 years old,
```

```
Charlie is 65 years old .
```

Let's find out who likes who on Twitter:

```
for (triplet <- graph.triplets.collect)
{
println(s"${triplet.srcAttr._1} likes ${triplet.dstAttr._1}")
}
```

Result The execution of the above code is as follows:

```
Bob loves Alice.
Bob loves David.
Charlie loves Bob
Charlie loves Fran
David loves Alice.
Ed loves Bob
Ed loves Charlie
Ed loves Fran
```

GraphX provides the ability to perform graph computations, such as graph traversal, shortest path finding, node search, subgraph finding, etc.

The module has a convenient and high-level API for working with graphs, which facilitates the development and implementation of analytical tasks on graphs.

GraphX supports working with various types of graphs, including directed and undirected graphs. The module is designed to process large graphs and allows users to scale calculations on Apache Spark clusters.

GraphX contains a set of built-in graph algorithms, such as PageRank, breadth-first and depth-first traversal, shortest path finding, and more. GraphX is integrated with other Spark modules, such as Spark SQL, Spark Streaming, and Spark MLlib, allowing users to use graphs in various analytical tasks. The module allows users to perform various operations on graphs, such as merging graphs, filtering nodes and edges, etc.

GraphX benefits include the ability to integrate with other Spark components for data processing, high performance thanks to optimized graph processing algorithms, support for complex graph operations and analytics, as well as convenient APIs for working with graphs in Scala, Java, and Python, which allows users to extend capabilities and integrate them into existing workflows.

7.3. Capabilities of the Apache Flink computing platform for streaming and batch data processing

Apache Flink is a distributed streaming data processing and computing framework designed for real-time analysis and processing of large amounts of data. Apache Flink divides the data processing module into two main submodules: the DataStream API for streaming analysis and the DataSet API for batch analysis. Both APIs provide the means to perform operations on data. Flink supports many data sources, including Apache Kafka, Apache Hadoop HDFS, Apache Cassandra, ElasticSearch, file systems, and native integrations.

Data can be stored in memory or on disk, depending on speed and reliability requirements. Apache Flink uses the "state" abstraction to store intermediate results of computations.

Flink provides the ability to perform operations on stream and batch data, such as filtering, splitting, merging, grouping, etc., providing a user-friendly interface for data processing.

The system has a query optimizer to improve computational and memory efficiency, which helps improve performance. Flink can scale data processing by adding additional nodes to the cluster. Large amounts of data can be processed in parallel.

State preservation and restoration allows processors to recover state upon failure and continue processing data without loss.

Flink integrates with other projects in the Apache ecosystem, such as Apache Kafka, Apache Hadoop, and Apache ZooKeeper.

Apache Flink supports the FlinkML machine learning library, which provides capabilities for solving classification, regression, and other machine learning tasks. Apache Flink can integrate with many other tools and programming languages, such as Python, R, other data processing systems, and databases.

Apache Flink's architecture makes it a powerful tool for processing streaming and batch data in real time and for performing various data analysis and machine learning tasks.

Apache Flink is a distributed processing platform and engine for stateful computing over unbounded and bounded data streams.

All data is created as a stream of events. Credit card transactions, sensor measurements, machine logs, or user interactions on a website or mobile app – all of this data is generated as a stream.

Data can be processed as unbounded or bounded streams (Fig. 7.10).

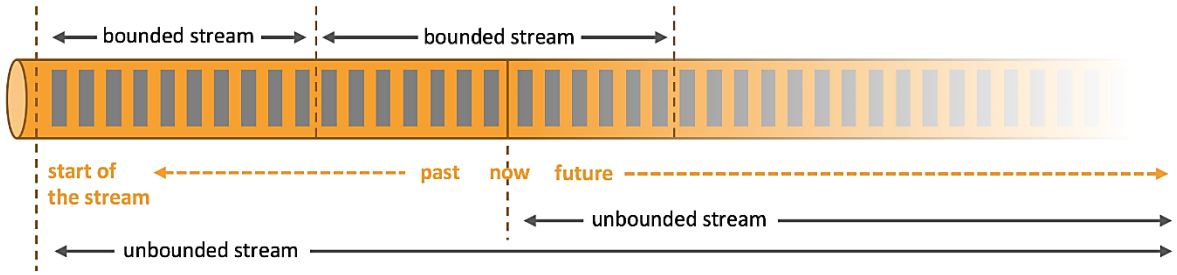


Fig. 7.10. Data as unbounded or bounded streams

Unbounded streams have a beginning but no definite end. They do not stop and provide data as they are created. Unbounded streams must be processed continuously, that is, events must be processed immediately after they are received. It is impossible to wait for all input data to arrive, because the input is unbounded and will not be completed at any point in time. Processing unbounded data often requires that events arrive in a specific order, such as the order in which the events occurred, so that the completeness of the result can be justified.

Bounded streams have a definite beginning and end. Bounded streams can be processed by accepting all data before performing any computations. Ordered reception is not required for bounded stream processing, as a bounded set of data can always be sorted. Bounded stream processing is also known as batch processing.

Apache Flink handles both unbounded and bounded datasets. Precise timing and state control allows the Flink runtime to run any application in unbounded threads. Bounded threads are internally handled using algorithms and data structures that are specifically designed for fixed-size datasets, which provides high performance. Apache Flink is a distributed system that requires compute resources to execute applications. Flink integrates with all popular cluster resource managers, such as Hadoop YARN and Kubernetes, but can also be configured to run as a standalone cluster.

Flink is designed to work with all of the resource managers listed above. This is achieved through special resource manager deployment modes that allow Flink to interact with each resource manager in its own way.

When deploying an application, Flink automatically determines the required resources based on the configured parallelism of the application and requests them from the resource manager. In the event of a failure, Flink replaces the failed container by requesting new resources. All communication to submit or manage the request is done through REST calls. This makes it easy to integrate Flink into many environments.

Flink is designed to run stateful streaming applications at any scale. Applications are parallelized into thousands of tasks that are distributed and executed concurrently across the cluster. This allows the application to leverage large amounts of CPU, main memory, disk, and network I/O. The asynchronous and incremental checkpointing algorithm ensures minimal impact on processing latency, ensuring state consistency exactly once.

Flink users have recorded impressive scalability metrics for Flink applications running in their environments, such as applications processing trillions of events per day, applications supporting multiple terabytes of state, and applications running on thousands of cores.

Flink provides three-tier APIs. Each API offers a different trade-off between conciseness and expressiveness and targets different use cases (Fig. 7.11).

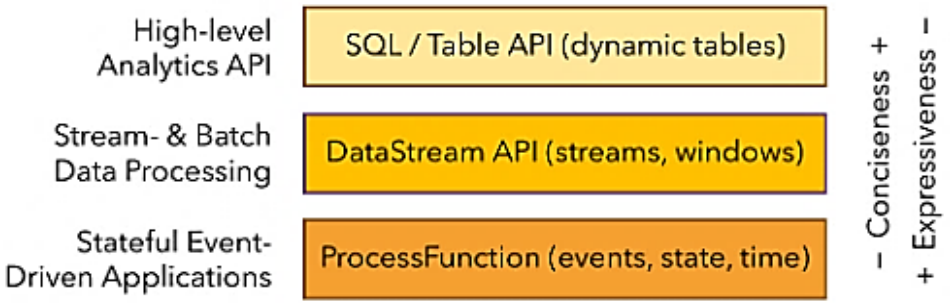


Fig. 7.11. Three-tier Flink API

ProcessFunctions are functional interfaces that Flink provides for processing individual events from one or two input streams or events grouped in a window. ProcessFunction functions provide fine-grained control over

time and state. A ProcessFunction function can arbitrarily change its state and register timers that will trigger a callback function in the future. Thus, ProcessFunctions can implement complex business logic for each event, which is required for many event-driven applications.

Flink includes libraries for common data processing cases. Libraries are usually built into the API and are not completely self-contained. They can take advantage of all the features of the API and be integrated with other libraries.

Complex Event Processing (CEP) – pattern detection is a common use case for event stream processing. The Flink CEP library provides an API for event pattern detection. The CEP library is integrated with the Flink DataStream API, and patterns are evaluated on DataStreams. Applications for the CEP library include network intrusion detection, business process monitoring, and fraud detection.

DataSet API is the core Flink API for batch processing applications. The DataSet API primitives include map, reduce, (outer) join, co-group, and iterate. All operations are supported by algorithms and data structures that operate on serialized data in memory and are moved to disk if the data size exceeds the memory size. The Flink DataSet API data processing algorithms are implemented in a manner analogous to traditional database operators, such as hybrid hash join or outer merge sort.

Gelly is a library for processing and analyzing scalable graphs. Gelly is implemented on top of and integrated with the DataSet API. Gelly benefits from its scalable and robust operators, includes built-in algorithms such as label propagation, page ranking, and also provides a Graph API that makes it easy to implement own graph algorithms.

Stateful Flink programs are optimized for accessing local state. The state of a job is always maintained in memory or, if the size of the state exceeds the available memory, in data structures on disk with efficient access. Therefore, jobs perform all computations by accessing local state, which ensures low processing latencies. Flink guarantees state consistency in the event of failures by periodically and asynchronously sending local state to long-term storage (Fig. 7.12).

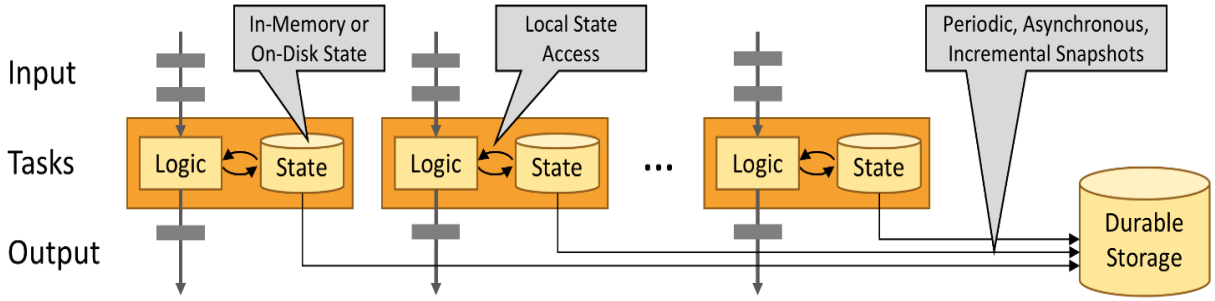


Fig. 7.12. State retention and access to local state of Flink programs

Every streaming program has state, meaning that programs that apply transformations to individual events do not need state. Any program that performs basic business logic must remember events or intermediate results so that they can be accessed later, for example, when the next event arrives after a certain amount of time (Fig. 7.13).

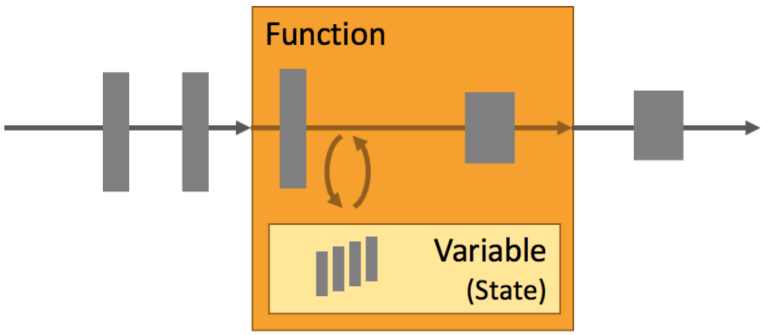


Fig. 7.13. Function with program state tracking

Apache Flink provides the ability to use stateful functions to preserve and manage state during streaming data processing. This is especially useful in streaming data processing where previous data and the results of computations need to be taken into account.

Stateful functions allow to store state that may be important for later calculations. As new data is processed, functions can update the state to reflect the new information. For example, they can add values to a sum, change counters, and so on. Functions can read the state to use in later calculations. This can be useful for making decisions based on previous results.

Apache Flink provides the ability to manage the state lifecycle, including initialization, preservation, and recovery of state upon failures.

Stateful functions can set timeouts for state or react to certain events, allowing them to manage state in real time.

Stateful functions are especially useful in streaming data processing, where previous data and the results of calculations need to be taken into account. They allow build complex data processing logic using state. Stateful functions allow to perform windowing and aggregation operations in streaming analysis, storing intermediate results for each window.

Apache Flink works primarily with the Java and Scala programming languages, and supports streaming data processing and state tracking. Flink has a Python API called PyFlink, but state tracking support can be limited.

Below is an example demonstrating windowed aggregations and state management using PyFlink. This example showcases how to group data into time windows and apply a custom aggregation function with state tracking.

```
from pyflink.common.serialization import
SimpleStringSchema
from pyflink.datastream import
StreamExecutionEnvironment
from pyflink.datastream import TimeCharacteristic
from pyflink.datastream.functions import
AggregateFunction
from pyflink.datastream.window import Time
from pyflink.datastream.window import
TumblingEventTimeWindows
from pyflink.common.watermark_strategy import
WatermarkStrategy
from pyflink.datastream.functions import RuntimeContext
# Set up the StreamExecutionEnvironment
env =
StreamExecutionEnvironment.get_execution_environment()
env.set_stream_time_characteristic(TimeCharacteristic.E
ventTime)
env.enable_checkpointing(10000)  # Checkpoint every 10
seconds
# Define a custom AggregateFunction with state
class StatefulSumFunction(AggregateFunction):
    def open(self, runtime_context: RuntimeContext):
```

```
        # Initialize state to keep a running sum
        self.state = runtime_context.get_state(
StateTtlConfig.new_builder().ttl(600).build()  # State
TTL of 10 minutes
        )
        def create_accumulator(self):
        # Create an initial accumulator (sum, count)
        return [0, 0]
    def add(self, value, accumulator):
        # Update the sum and count
        current_sum = self.state.get() or 0
        updated_sum = current_sum + value
        self.state.put(updated_sum)  # Update state
        return [updated_sum, accumulator[1] + 1]
    def get_result(self, accumulator):
        # Calculate the average
        return accumulator[0] / accumulator[1] if
accumulator[1] > 0 else 0
    def merge(self, acc_a, acc_b):
        # Merge two accumulators
        return [acc_a[0] + acc_b[0], acc_a[1] +
acc_b[1]]
# Simulate an input data stream
stream = env.from_collection(
    [(1, 100), (2, 200), (3, 300), (1, 150), (2, 250),
(3, 350)],
    type_info=Types.ROW([Types.INT(), Types.INT()])
)
# Assign timestamps and watermarks
stream = stream.assign_timestamps_and_watermarks(
WatermarkStrategy.for_monotonous_timestamps().with_time
stamp_assigner(lambda row, timestamp: row[0])
)
# Group by key and apply a tumbling window
aggregated_stream = (
    stream
```

```
    .key_by(lambda x: x[0])  # Group by the first field
.window(TumblingEventTimeWindows.of(Time.seconds(60)))
# 60-second tumbling window
    .aggregate(StatefulSumFunction())  # Apply the
custom aggregate function
)
# Print the results
aggregated_stream.print()
# Execute the pipeline
env.execute("Windowed Aggregation with Stateful
Tracking")
```

The `StreamExecutionEnvironment` is configured for event time processing with checkpoints enabled every 10 seconds. Event time allows the system to process data based on event timestamps, critical for delayed events in real-time applications.

The `StatefulSumFunction` class extends the `AggregateFunction`. It maintains a state that tracks the running sum of values for each key, using state with a time-to-live (TTL) of 10 minutes.

In the `open` method, the state is initialized with a TTL configuration to manage resource usage.

These maintain intermediate results, such as the running sum and count.

The `add` method updates the state with the latest sum for the current key, ensuring stateful tracking across multiple elements in a window.

A tumbling window groups the data into fixed 60-second intervals. Each window is processed independently, and the custom aggregate function is applied to calculate the average for each key within the window.

Watermarks are used to handle out-of-order events. This ensures that late data is processed correctly within the window boundaries.

The `execute` method starts the Flink job. Processed results are printed to the console for verification.

Uses state to track and maintain data across multiple records, with TTL to prevent resource exhaustion. Groups data into time windows for efficient batch-like processing. Checkpoints provide resiliency to system failures. Keyed state allows distributed processing, ensuring scalability.

This example highlights real-world use cases such as streaming analytics, where windowed aggregations with stateful logic enable efficient, scalable, and fault-tolerant data processing.

FlinkML is a machine learning library that allows developers and researchers to perform machine learning tasks on the Apache Flink platform, providing tools for training models and processing data in streaming and batch modes.

Flink ML supports various machine learning algorithms, including classification, regression, clustering, recommendation, and many others. We can choose the appropriate algorithm for the appropriate task. Flink ML allows users to select the most important features for training models, supports graph algorithms for working with graph structures, and provides tools for processing text data, which is useful for natural language processing tasks.

Flink ML integrates with machine learning libraries such as Scikit-Learn or TensorFlow. Using Apache Flink allows users to scale model training on clusters, allowing to process large amounts of data. Flink ML allows users to deploy trained models for prediction and analysis of streaming data.

Using stateful functions in Apache Flink, we can save and update model parameters during training.

Because many streaming applications are designed to run continuously with minimal downtime, the stream processor must provide excellent disaster recovery, as well as tools to monitor and support the applications as they run. A distributed stream processor must be disaster-resilient to be able to run streaming applications 24/7. Obviously, this means not only restarting the application after a failure, but also ensuring that its internal state remains consistent so that the application can continue processing as if the failure had never occurred.

Flink provides several features that ensure smooth operation of applications and their stability.

Consecutive checkpoints are Flink recovery mechanism based on consistent checkpoints of the program state. In the event of a crash, the program is restarted and its state is loaded from the last checkpoint. Combined with resettable stream sources, this feature can guarantee state consistency exactly once.

Efficient checkpointing – checkpointing application state can be quite expensive if the application maintains terabytes of state. Flink can perform asynchronous and incremental checkpoints so that the impact of checkpoints on the application SLA latency is very small.

End-to-end exactly-once – Flink has transactional sinks for storage systems that ensure that data is only written once, even in the event of a failure.

Integration with cluster managers – Flink is integrated with cluster managers such as Hadoop YARN or Kubernetes. In case of a process failure, a new process is automatically started to take over its work.

High availability configuration – Flink has a high availability mode that eliminates all single points of failure. The high availability mode is based on Apache ZooKeeper, a proven service for reliable distributed coordination.

Flink savepoints are a unique and powerful feature that solves the problem of updating stateful applications and many other related problems. A savepoint is a consistent snapshot of the state of an application, so it is very similar to a checkpoint. Unlike checkpoints, savepoints must be triggered manually and are not automatically removed when the application stops. A savepoint can be used to start a stateful application and initialize its state.

Savepoints allow to perform the following functions.

Program evolution – savepoints can be used for program development. A corrected or improved version of the program can be restarted from a savepoint taken from a previous version of the program. It is also possible to run the program from an earlier point in time (if such a savepoint exists) to correct incorrect results produced by a faulty version.

Cluster migration – using savepoints, applications can be migrated (or cloned) to different clusters.

Upgrading the Flink version – an application can be migrated to run on a new Flink version using a savepoint.

Scaling a program – savepoints can be used to increase or decrease the parallelism of a program.

A/B tests and what-if scenarios – the performance or quality of two (or more) different versions of an application can be compared by running all versions from a single save point.

We can pause a program by taking a savepoint and stopping it. At any later point, we can resume the program from the savepoint. Savepoints can be archived to allow to reset the program to an earlier point in time.

Like any other service, continuously running streaming applications require monitoring and integration into operational infrastructure and monitoring services. Monitoring helps to anticipate problems and respond to them in advance.

An important feature is the available interfaces for managing running applications. Flink integrates with many common logging and monitoring services and provides a REST API for managing applications and querying information, as well as a web interface for inspecting, monitoring, and debugging running applications. Flink implements the popular slf4j logging interface and integrates with the log4j or logback logging frameworks.

Flink has a sophisticated metrics system for collecting and reporting system and user-defined metrics. Metrics can be exported to several reports, including JMX, Ganglia, Graphite, Prometheus, StatsD, Datadog, and Slf4j.

The REST API is used to submit a new application, retrieve a savepoint of a running application, or cancel an application. The REST API also provides metadata and aggregated metrics for running or completed applications.

Apache Flink is a great choice for developing and running many types of applications due to its extensive feature set. Flink features include support for streaming and batch processing, sophisticated state management, event handling semantics, and state consistency guarantees. Additionally, Flink can be deployed on a variety of resource providers, such as YARN and Kubernetes. Configured for high availability, Flink has no single point of failure and scales to thousands of cores and terabytes of application state, providing high throughput and low latency.

An event-driven application is a stateful application that accepts events from one or more event streams and responds to incoming events by triggering computations, state updates, or external actions.

Event-driven applications are an evolution of traditional application design with separate compute and storage tiers. In this architecture, applications read data from and store data in a remote transactional database.

In contrast, event-driven applications are based on stateful threading applications. In this design, data and computation are co-located, providing

local access to the data (in memory or on disk). Fault tolerance is achieved by periodically writing checkpoints to remote persistent storage (Fig. 7.14).

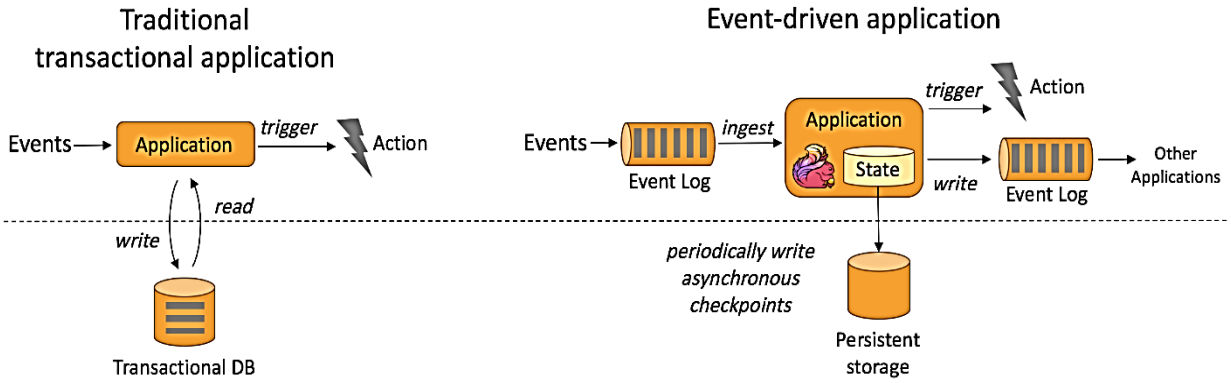


Fig. 7.14. Comparison of traditional application architecture and event-driven applications

Instead of querying a remote database, event-driven applications access their data locally, which provides better performance in terms of both bandwidth and latency. Periodic checkpoints to a remote persistent store can be performed asynchronously and incrementally. Therefore, the impact of checkpoints on regular event processing is very small.

The event-driven design of an application provides more benefits than simply accessing local data. In a multi-tier architecture, it is common for multiple applications to share a single database. Therefore, any change to the database, such as a change in data layout due to an application update or a service scaling, requires coordination. Because each event-driven application is responsible for its own data, changes to the data representation or application scaling require less coordination.

The limits of event-driven applications are determined by how well a stream processor can handle time and state. Many of Flink's features are centered around these concepts. Flink provides a set of state primitives that can handle large amounts of data (up to several terabytes) with consistency guarantees. In addition, Flink has a library for complex event processing (CEP) to detect patterns in data streams. An important Flink feature for event-driven applications is the savepoint. A savepoint is an image of a consistent state that can be used as a starting point for compatible applications. Given a savepoint, an application can be updated or scaled, or multiple versions of an application can be run for A/B testing.

Traditionally, analytics is performed as batch queries or programs on limited datasets of recorded events. To include the latest data in the analysis result, it must be added to the analyzed dataset, and the query or program is rerun. The results are written to storage or output as reports. With sophisticated stream processing, analytics can also be performed in real time. Instead of reading final datasets, stream queries or programs accept real-time streams of events and continuously create and update results as events are "consumed." The results are written to an external database or stored as internal state. The Dashboard application can read the latest results from an external database or directly query the internal state of the application.

Apache Flink supports streaming as well as batch analytics applications (Fig. 7.15, 7.16).

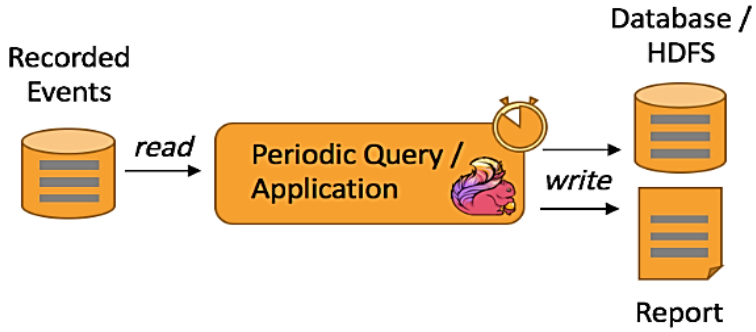


Fig. 7.15. Batch analytics

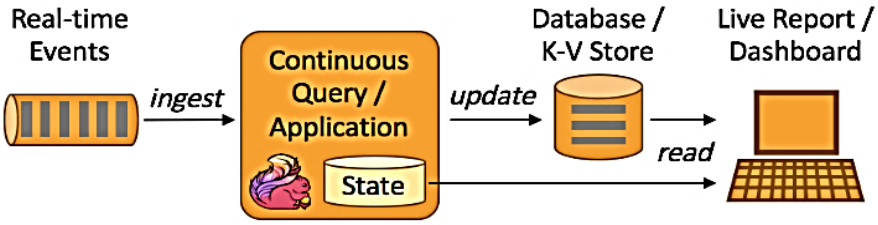


Fig. 7.16. Stream analytics

Flink supports continuous streaming as well as batch analytics, and includes an ANSI-compliant SQL interface with unified semantics for batch and streaming queries. SQL queries compute the same result whether they are executed on a static dataset of recorded events or on a real-time stream of events. Rich support for user-defined functions ensures that custom code can be executed in SQL queries. If even more custom logic is required, the Flink DataStream API or DataSet API provide lower-level control.

Many common data transformation or enrichment tasks can be solved using Flink's SQL interface and its support for user-defined functions. Data pipelines with more advanced requirements can be implemented using the DataStream API, which is more general.

Flink provides a rich set of connectors for various data storage systems, such as Kafka, Kinesis, Elasticsearch, and JDBC database systems, and includes continuous sources for file systems that track directories and sinks that write files with distributed timestamps.

Apache Flink supports business-critical applications in many companies and enterprises around the world (Fig. 7.17).

Fig. 7.17. Examples of famous Flink users

A multinational technology company from China specializing in e-commerce, cloud computing, digital payments, and online services, Alibaba uses a fork of Flink called Blink to optimize search rankings in real time.

Amazon Managed Service for Apache Flink is a fully managed Amazon service that allows users to use Apache Flink to process and analyze streaming data.

Huawei is a leading global provider of information and communication technology infrastructure and smart devices. Huawei Cloud provides a cloud service based on Flink.

Uber uses its internal SQL-based AthenaX streaming analytics platform with open source Apache Flink. AthenaX is Uber's internal streaming analytics platform, built on the open source Apache Flink and supporting SQL queries for real-time streaming data processing. The platform works by taking data from various sources (such as user events, GPS trackers, or monitoring systems) and processing it in a streaming manner using SQL

queries that users write through a user-friendly interface. Apache Flink provides distributed processing of this data, performing calculations in parallel on many nodes to achieve low latency and high performance. Thanks to this, AthenaX allows Uber to perform complex real-time analytics, such as calculating ETA (estimated time of arrival), optimizing routes, or detecting anomalies in processes.

ResearchGate is social network for scientists, uses Flink to analyze the network and discover similar articles. The principle of operation is that data about articles, their metadata (keywords, authors, citations) and user interactions (views, downloads, likes) are processed in streaming mode using Flink. The platform performs complex analytics using machine learning and text similarity algorithms to create connections between articles. The results of this processing allow users to offer relevant scientific materials, optimize recommendations and improve interaction in the network. Flink provides scalability and high performance, which allows ResearchGate to analyze data from millions of articles and interactions in real time.

Zalando, one of the largest e-commerce companies in Europe, uses Flink for real-time process monitoring and ETL.

7.4. Comparative analysis of computing platforms for streaming data analytics

Streaming data analytics involves processing and analyzing real-time data that comes in a continuous stream. In addition to Kafka, Spark, and Flink, there are other important technologies for streaming data analytics, each with its own unique features and benefits.

Apache Storm platform is designed to process large streams of data in real time with high throughput and low latency. It is used for social media analysis, log processing, and sensor data monitoring in IoT.

Kafka is used as a messaging platform, while Storm processes data. These technologies are often used together.

Spark is better suited for micro-batch processing, while Storm processes data at the individual record level. Flink provides better support for event processing, while Storm is more popular for low-latency scenarios.

Apache Samza specializes in low-latency data stream processing and integration with Apache Kafka. Used for log analysis, event processing, and building recommended systems.

Apache Samza integrates with Kafka, using Kafka as the primary data source and provides better support for low-latency event processing compared to micro-batch processing in Spark.

Flink technology is more versatile and supports both streaming and batch processing, while Samza focuses on streams.

Apache Pulsar is a messaging system with streaming capabilities that offers quotas and high availability. Used for financial transaction processing, log analytics, and IoT event monitoring.

Pulsar offers advanced support for real-time data processing, making it a more flexible alternative to Kafka. Pulsar is better suited for real-time event processing, while Spark focuses more on micro-batch processing.

Flink has more advanced data analysis tools, while Pulsar provides efficient message passing and stream processing.

Google Cloud Dataflow is a managed data processing service for stream and batch processing built on Apache Beam. Used for real-time data processing in various industries such as financial services, online advertising, and healthcare.

Google Cloud Dataflow can be used in healthcare for real-time processing and analysis of data streams from medical devices, such as patient monitors in hospitals.

For example, Dataflow can collect data on heart rate, blood oxygen levels, or blood pressure, process them in real time, detect anomalies (such as a sudden drop in heart rate), and immediately notify medical staff. This enables rapid response to critical situations, improving the accuracy and efficiency of medical care.

Fig. 7.18 illustrates a remote patient health monitoring system based on sensors and cloud technologies. The person is equipped with various biometric sensors that measure heart rate, blood oxygen level, blood pressure, motion, glucose level, body tilt, ECG, and respiration.

These data are transmitted via a network device to the Internet, where they are stored in databases and undergo analytical processing. Doctors and medical personnel gain access to the information to monitor the patient's condition and respond promptly in case of critical changes in health indicators. This system is essential for telemedicine, as it ensures continuous remote monitoring and rapid assistance.

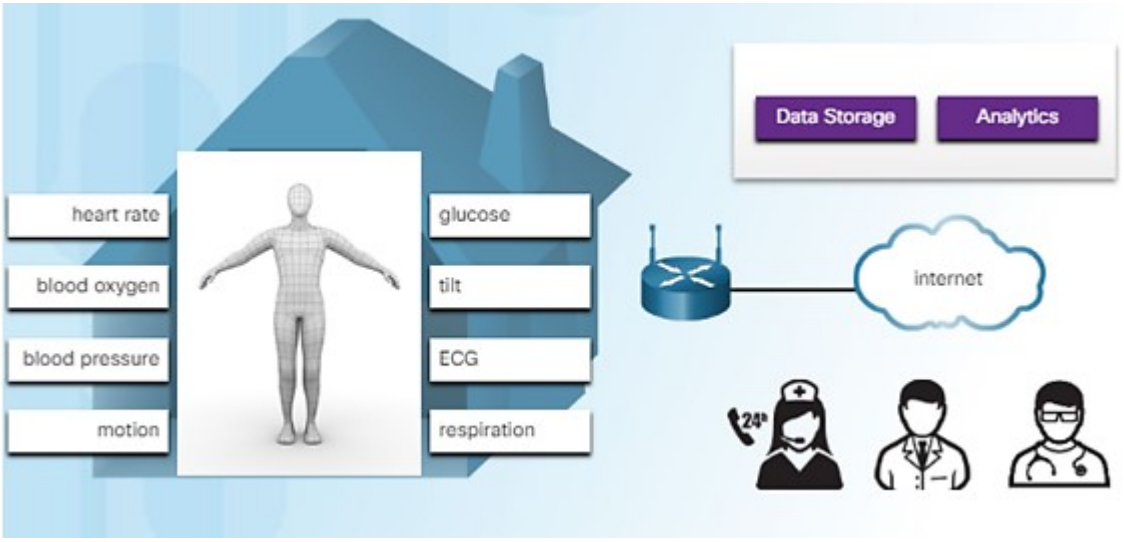


Fig. 7.17. Remote patient health monitoring system based on sensors and cloud technologies

Dataflow can be used together with Kafka to process streaming data. Dataflow provides better integration with other Google cloud services, while Spark is a more general data processing platform. Flink provides more flexible and versatile data stream processing, while Dataflow provides a managed solution with integration into Google Cloud.

Amazon Kinesis provides services for processing real-time data streams, including Kinesis Data Streams, Kinesis Data Firehose, and Kinesis Data Analytics. Used to collect, process, and analyze data from logs, IoT sensors, and social media.

Kinesis provides similar features for processing data streams, but integrates better with other AWS services. Kinesis provides simpler integration with AWS, while Spark can be used for more complex analytical tasks. Flink provides more advanced streaming data processing capabilities, while Kinesis provides a managed solution with integration into AWS.

Azure Stream Analytics is a managed, real-time data stream processing service that offers integration with other Azure services. Used for monitoring IoT events, analyzing log data, and financial transactions. For example, a smart city company uses Azure Stream Analytics to process data from thousands of real-time traffic sensors. The data stream comes from IoT devices installed on traffic lights and surveillance cameras, is analyzed to detect congestion, accidents, or speeding, and then the results are fed into Azure SQL Database and Power BI for visualization. This allows the city to

quickly change traffic lights, alert drivers through navigation systems, and improve the overall efficiency of the transportation infrastructure.

Azure Stream Analytics offers a robust, fully managed real-time analytics service designed to handle large-scale data processing needs with unparalleled ease and efficiency. One of its standout advantages is its ability to seamlessly integrate with various data sources in the Azure ecosystem, such as Event Hubs, IoT Hub, and Blob Storage. This enables organizations to process massive streams of real-time data with minimal setup.

The service is built for high scalability, allowing users to analyze millions of events per second and deliver actionable information with ultra-low latency. This makes it particularly valuable for applications in IoT analytics, fraud detection, and monitoring systems where timeliness is critical. Furthermore, Azure Stream Analytics provides a powerful SQL-like query language that simplifies the process of filtering, aggregating, and transforming data, making it accessible to a broad range of users, from data engineers to business analysts. Its ease of use reduces the complexity often associated with setting up and managing real-time analytics pipelines.

Another major advantage is the built-in reliability, flexibility, and cost-efficiency that Azure Stream Analytics offers. The platform guarantees high availability with built-in disaster recovery and auto-scaling capabilities, ensuring that analytics workloads can operate continuously even under changing demands or in the face of system failures.

Additionally, it supports advanced analytics, including machine learning integration and complex event processing, which allows businesses to extract data and make predictive decisions in real time.

Azure Stream Analytics also supports both cloud and edge processing, enabling users to deploy analytics close to where data is generated, reducing latency and optimizing resources. Its pay-as-you-go pricing model further ensures cost-efficiency, allowing organizations to scale their operations based on their needs without committing to large upfront investments. These features make Azure Stream

Table 7.1 provides a comparative analysis of the main streaming data analytics technologies.

Table 7.1. **Comparative analysis of streaming data analytics technologies**

Technology	Features	Advantages	Disadvantages
Apache Kafka	Messaging platform	High throughput, reliability	Does not support event handling "out of the box"
Apache Spark	Micro-batch data processing	Support for batch and streaming processing	Higher latency compared to Flink and Storm
Apache Flink	Stream and packet processing	Low latency, flexibility	Difficulty of setup
Apache Storm	Real-time event processing	Low latency	High complexity of setup
Apache Samza	Integration with Kafka	Low latency	Fewer features compared to Flink
Apache Pulsar	Multi-thematic quotas	High throughput	Relatively new project
Google Cloud Dataflow	Managed stream and packet processing	Integration with Google Cloud	Dependence on Google Cloud
Amazon Kinesis	Stream processing services	Integration with AWS	Cost on a large scale
Azure Stream Analytics	Managed real-time stream processing	Integration with Azure	Dependency on Azure

The table provides a comparative overview of several popular technologies used for data processing, highlighting their features, advantages, and disadvantages. These tools are widely used in handling large-scale data streams, offering different capabilities tailored to specific use cases.

Apache Kafka is a messaging platform known for its high throughput and reliability, making it an excellent choice for real-time data pipelines. It lacks built-in event-handling capabilities, requiring additional tools or custom configurations to manage events. Apache Spark excels in micro-batch data processing and supports both batch and streaming data workloads. While it is versatile and widely adopted, its latency is higher compared to other stream

processing tools such as Apache Flink or Apache Storm. Apache Flink and Apache Storm are both designed for low-latency data processing. Flink offers flexibility and strong support for stream and packet processing, though it has a steep learning curve and challenging setup process. Similarly, Storm provides low-latency real-time event processing but is known for its high complexity during setup.

Apache Samza integrates well with Kafka and provides low-latency processing, it lacks the extensive feature set of more advanced frameworks like Flink. Apache Pulsar, on the other hand, is a relatively new project offering high throughput and support for multi-thematic quotas, though its novelty may present some challenges in terms of community support and maturity. For cloud-managed solutions, Google Cloud Dataflow offers robust integration with Google Cloud services, making it an ideal choice for organizations already invested in the ecosystem, but it is dependent on Google Cloud. Similarly, Amazon Kinesis and Azure Stream Analytics integrate seamlessly with AWS and Azure, respectively. While Kinesis is efficient, its costs can scale quickly with increased data volume. Azure Stream Analytics is an excellent choice for real-time stream processing within the Azure environment, but it too is limited by its dependency on the Azure platform.

Control questions and tasks for topic 7

List of theoretical questions

1. What is the Kafka distributed streaming platform and what are the main advantages it provides when processing streaming data?
2. How does Kafka provide scalability and reliability for streaming data processing?
3. What tools and capabilities does Kafka provide for working with data in real time?
4. What scenarios and use cases can the Kafka platform be useful for enterprises and organizations?
5. What are the main features and benefits of the Apache Spark data processing platform?
6. How does Spark use distributed computing to efficiently process big data?
7. What are the capabilities and applications of the Spark MLlib module for machine learning?

8. What machine learning algorithms does Spark MLlib support and how to use them?

9. How does Spark Streaming help perform real-time streaming data analysis?

10. What are the challenges and applications for Spark Streaming?

11. What are the capabilities of the GraphX module for analyzing and processing graph structures of big data?

12. How does Spark integrate with other tools and components for complex data analysis and processing in a distributed environment?

13. What capabilities does Apache Flink provide for processing streaming data, and how do they differ from processing batch data?

14. What are stateful functions in Apache Flink, and how are they used for data processing?

15. What capabilities and tasks can be solved using stateful functions in Apache Flink?

16. How does Flink ML integrate with Apache Flink for data processing and model training?

17. How does Apache Flink provide scalability in data processing and model training?

18. What capabilities does Flink ML provide for deploying and using trained models in real systems?

List of tasks for independent work

Practical task 1. Real-time data processing and aggregation with Apache Kafka and Apache Flink.

Objective: use Apache Kafka to produce and consume real-time data streams and Apache Flink to process and analyze the data with stateful transformations.

Step 1. Set up the environment. Install Python 3.6 (or later), Apache Kafka, and Apache Flink. Install the required Python libraries:

```
pip install kafka-python pyflink
```

Download and unzip Apache Kafka and Apache Flink from their official websites.

Step 2. Start Kafka and Create a Topic.

Start ZooKeeper and the Kafka server:

```
bin/zookeeper-server-start.sh
```

```
config/zookeeper.properties
    bin/kafka-server-start.sh config/server.properties
```

Create a Kafka topic:

```
    bin/kafka-topics.sh --create --topic sensor_data --
bootstrap-server localhost:9092 --replication-factor 1 -
-partitions 1
```

Step 3. Write a Kafka Producer Script. Create a Python script (producer.py) to simulate sensor data:

```
    from kafka import KafkaProducer
    import json
    import random
    import time
    producer = KafkaProducer(
        bootstrap_servers='localhost:9092',
        value_serializer=lambda v:
json.dumps(v).encode('utf-8')
    )
    while True:
        data = {'sensor_id': random.randint(1, 10),
'temperature': random.uniform(20, 30)}
        producer.send('sensor_data', value=data)
        print(f"Sent: {data}")
        time.sleep(1)
```

Run the script to start sending data:

```
    python producer.py
```

Step 4. Configure Flink Streaming Job. Create a Python Flink script (flink_job.py) to process the Kafka stream:

```
    from pyflink.datastream import
StreamExecutionEnvironment
    from pyflink.datastream.connectors.kafka import
FlinkKafkaConsumer
    from pyflink.common.serialization import
SimpleStringSchema
    from pyflink.common.watermark_strategy import
WatermarkStrategy
    import json
```

```
    def process_stream():
        env =
StreamExecutionEnvironment.get_execution_environment()
        env.set_parallelism(1)
        kafka_consumer = FlinkKafkaConsumer(
            topics='sensor_data',
           deserialization_schema=SimpleStringSchema(),
            properties={'bootstrap.servers':
'localhost:9092'}
        )
        kafka_consumer.set_start_from_latest()
        stream = env.add_source(kafka_consumer)
        parsed_stream = stream.map(lambda x:
json.loads(x))
        avg_temp = parsed_stream \
            .key_by(lambda x: x['sensor_id']) \
            .map(lambda x: (x['sensor_id'],
x['temperature'])) \
            .key_by(lambda x: x[0]) \
            .reduce(lambda a, b: (a[0], a[1] + b[1] /
2))  # Calculate running average
        avg_temp.print()
        env.execute("Sensor Data Processing")
    if __name__ == "__main__":
        process_stream()
```

Run the Flink job:

```
python flink_job.py
```

Step 5. Extend the analysis. Modify the Flink job to include minimum and maximum temperature:

```
    min_max_temp = parsed_stream \
        .key_by(lambda x: x['sensor_id']) \
        .map(lambda x: (x['sensor_id'],
x['temperature'], x['temperature'])) \
        .key_by(lambda x: x[0]) \
        .reduce(lambda a, b: (a[0], min(a[1], b[1]),
max(a[2], b[2])))  # Min and Max
```

```
min_max_temp.print()
```

Test the updated Flink job.

Step 6. Document your results. Summarize the results, including the real-time averages, minimum, and maximum temperatures for each sensor.

Practical task 2. Multi-stream processing with Kafka and Spark Structured Streaming.

Objective: create two Kafka producers sending data to different topics, and use Spark Structured Streaming to consume and process the streams simultaneously.

Step 1. Set Up the Environment. Install Python 3.6 (or later), Apache Kafka, and Apache Spark.

Install required libraries:

```
pip install kafka-python pyspark
```

Step 2. Create Kafka topics. Create two Kafka topics:

```
bin/kafka-topics.sh --create --topic topic_a --
bootstrap-server localhost:9092 --replication-factor 1
--partitions 1
bin/kafka-topics.sh --create --topic topic_b --
bootstrap-server localhost:9092 --replication-factor 1
--partitions 1
```

Step 3. Write two Kafka producers. Producer for topic_a (producer_a.py):

```
from kafka import KafkaProducer
import json
import random
import time
producer = KafkaProducer(
    bootstrap_servers='localhost:9092',
    value_serializer=lambda v:
json.dumps(v).encode('utf-8')
)
while True:
    data = {'source': 'A', 'value':
random.randint(0, 50)}
    producer.send('topic_a', value=data)
    time.sleep(1)
```

Producer for topic_b (producer_b.py):

```
from kafka import KafkaProducer
import json
import random
import time
producer = KafkaProducer(
    bootstrap_servers='localhost:9092',
    value_serializer=lambda v:
json.dumps(v).encode('utf-8')
)
while True:
    data = {'source': 'B', 'value':
random.randint(51, 100)}
    producer.send('topic_b', value=data)
    time.sleep(1)
```

Step 4. Write a Spark Streaming Job. Create a Spark script (spark_job.py):

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, avg, min, max
spark = SparkSession.builder \
    .appName("MultiStreamProcessing") \
    .getOrCreate()
stream_a = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers",
"localhost:9092") \
    .option("subscribe", "topic_a") \
    .load()
stream_b = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers",
"localhost:9092") \
    .option("subscribe", "topic_b") \
    .load()
combined_stream = stream_a.union(stream_b)
aggregated = combined_stream \
```

```
        .groupBy(col("source")) \
        .agg(
            avg("value").alias("avg_value"),
            min("value").alias("min_value"),
            max("value").alias("max_value")
        )
    query = aggregated.writeStream \
        .outputMode("complete") \
        .format("console") \
        .start()
    query.awaitTermination()
```

Run the script to start processing:

```
python spark_job.py
```

Step 5. Analyze results, summarize the average, minimum, and maximum values for both streams.

List of used and recommended literature

7.1. Apache Hadoop [Electronic resource]. – Access mode: https://kafka.apache.org/

7.2. Spark Streaming Programming [Electronic resource]. – Access mode: https://spark.apache.org/docs/latest/streaming-programming-guide.html

7.3. Apache Flink Architecture [Electronic resource]. – Access mode: https://flink.apache.org/what-is-flink/flink-architecture/

7.4. Apache Storm [Electronic resource]. – Access mode: https://storm.apache.org/

7.5. Apache Samza [Electronic resource]. – Access mode: https://samza.apache.org/

7.6. Apache Pulsar [Electronic resource]. – Access mode: https://pulsar.apache.org/

7.7. Google Cloud Dataflow [Electronic resource]. – Access mode: https://cloud.google.com/dataflow?hl=en

7.8. Amazon Kinesis [Electronic resource]. – Access mode: https://aws.amazon.com/kinesis/

7.9. Azure Stream Analytics [Electronic resource]. – Access mode: https://azure.microsoft.com/en-us/products/stream-analytics

CHAPTER II.

SOFTWARE METHODS FOR BIG DATA ANALYTICS

Topic 8. Software methods for statistical analysis of big data

Data exploration and preparation is an important step in big data analytics in Python. Data collection can include a variety of sources such as databases, Excel files, CSV files, web services, APIs , etc.

8.1. Research and preparation of data for analysis

Before begin analyzing data, we need to spend time cleaning it. During this process, we may find outliers or anomalies in data. If outliers are found, they need to be investigated to understand the causes or correct the data.

Outlier is a value or data point that is significantly different from the others, either significantly smaller or significantly larger. Sometimes outliers are errors, and sometimes they contain important information. For example, in the graph in Fig. 8.1, the point in the far lower right corner is an outlier, while the rest of the data is grouped along a trend line.

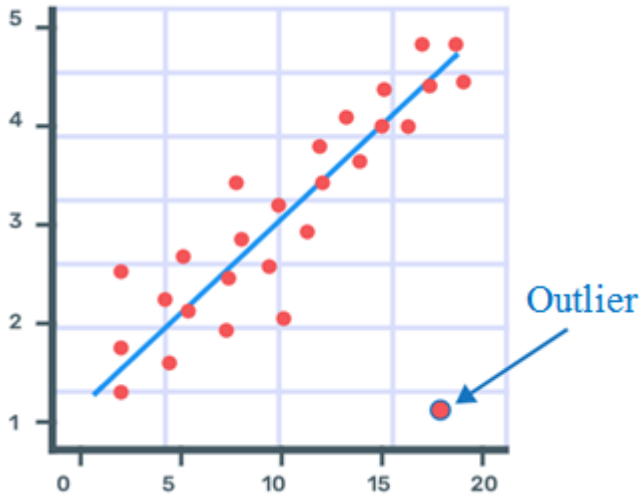
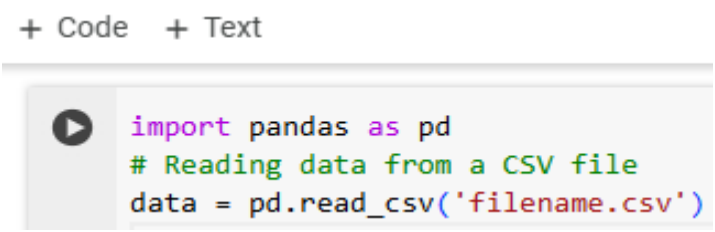


Fig. 8.1. Example of an outlier

During data analysis, outliers caused by errors can cause anomalies in the results. Outliers that are not errors can be very important to the analysis. That is why investigating such anomalies is an important part of the data cleaning process – it ensures accurate and reliable analysis.

In small data sets, outliers can be easily spotted by sorting or filtering the data. But for large data sets or large amounts of information, specialized tools are needed.

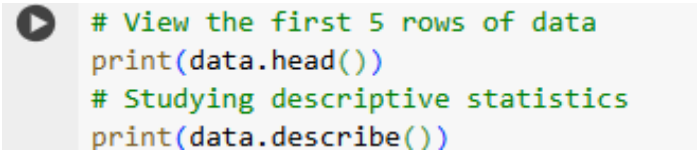
Python uses Pandas library to load and read data. The following code uses the Pandas library to read data from a CSV file into a variable called `data`.



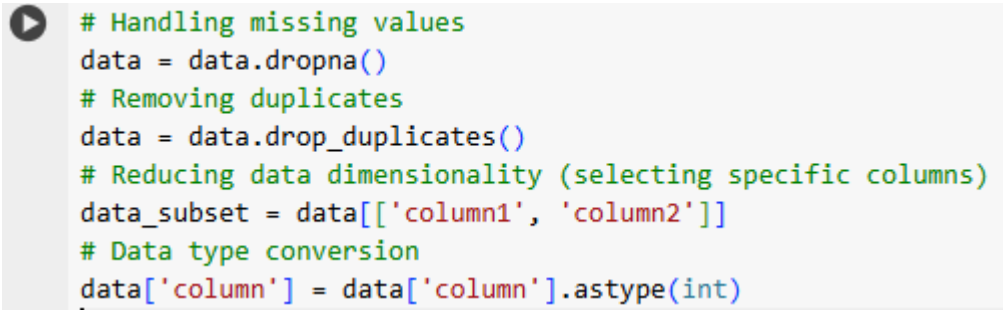
We import the `pandas` library and gives it the alias `pd`. Using the alias `pd` is a common approach in the Python community to shorten code.

Next, we use the `read_csv` function from the `pandas library` to read the data from the CSV file. The result of reading the data is stored in the variable `data`, which now contains a `DataFrame object` is the basic data structure in the `pandas library`.

To analyze and understand the data structure, view the first few lines, and study descriptive statistics, `the head()` and `describe()` methods are used:



When preliminary data exploration, it is important to identify and process missing values, remove duplicates, convert data types, and use functions to reduce data dimensionality:



Analysis of the distribution of variables, study of correlations, and detection of anomalies is carried out using the following code:

```
# Variable distribution analysis
import seaborn as sns
sns.histplot(data['variable'], kde=True)
# Correlation matrix
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True)
```

To better understand the relationships and structure of data, it is advisable to use graphs and charts:

```
# Distribution schedule
sns.boxplot(x='variable1', y='variable2', data=data)
# Line graph
sns.lineplot(x='time', y='variable', data=data)
```

These steps help prepare the data for further analysis and modeling in the Python environment. Consider an example of a preliminary study of data before its analysis. In the example, the data of indicators recorded by a temperature and humidity sensor was loaded. Since the file was received in CSV format, the appropriate function was used to load it into the data frame.

We verify the import was correct by displaying the first five records in the data frame:

```
weatherSensorDf.head()
```

The CSV file contents have been loaded into the data frame correctly. The table has seven columns: UNIXTIME, date, time, TEMP, HUMD, RSSI, VCC. The Date and Time columns are responsible for the date and time of the indicator acquisition and duplicate the information from the UNIXTIME column, which presents the date and time in UNIX timestamp format.

We add a new TIMESTAMP column to the data frame, into which we copy the values from the UNIXTIME column and convert them to the Python date and time type using the appropriate function:

```
weatherSensorDf['TIMESTAMP'] = pd.to_datetime(weatherSensorDf['UNIXTIME'],
unit='s', utc=True).dt.tz_convert('Europe/Kyiv')
weatherSensorDf
```

The new TIMESTAMP column in the data frame now contains the correct date and time information for recording sensor readings in a readable format. The UNIXTIME, Date, Time columns are no longer needed as they duplicate the existing information in the TIMESTAMP column.

Therefore, we are removing these UNIXTIME columns, Date, Time from the data frame (Fig. 8.2):

```
weatherSensorDf.drop(['UNIXTIME', 'Date ', 'Time'], axis=1, inplace=True)
weatherSensorDf.head()
```

	TEMP	HUMD	RSSI	VCC	TIMESTAMP
0	20.94	69.70	-41.00	3.1	2022-08-27 22:00:00+03:00
1	21.51	62.06	-42.17	3.1	2022-08-27 23:00:00+03:00
2	20.45	68.52	-43.42	3.1	2022-08-28 00:00:00+03:00
3	20.15	69.38	-44.75	3.1	2022-08-28 01:00:00+03:00
4	19.68	70.85	-46.45	3.1	2022-08-28 02:00:00+03:00

Fig. 8.2. Displaying a data frame using the method head()

We have five columns in the data frame. The TEMP column shows the outside temperature in degrees Celsius, the HUMD column shows the relative humidity in percent. The RSSI column is an indicator of the Wi-Fi signal strength level (through which the recorded data is received) reaching the sensor antenna, the VCC column displays the voltage of the batteries powering this sensor in volts, the TIMESTAMP column displays the date and time when the sensor recorded the indicators.

We find out how many sensor measurement records are available in the data frame:

```
len(weatherSensorDf)
```

The dataset contains 1392 records. That is, the size of the dataset is 1392 rows, and the number of columns is 5. We check the data types of each column in the dataset and make sure that there are no missing values in the rows (Fig. 8.3):

```
weatherSensorDf.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1392 entries, 0 to 1391
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TEMP       1392 non-null   float64
 1   HUMD       1392 non-null   float64
 2   RSSI       1392 non-null   float64
 3   VCC        1392 non-null   float64
 4   TIMESTAMP  1392 non-null   datetime64[ns, Europe/Kyiv]
dtypes: datetime64[ns, Europe/Kyiv](1), float64(4)
memory usage: 54.5 KB
```

Fig. 8.3. Display data types of each column in the dataset

As we can see, the data type for the TIMESTAMP column is date-time, meaning the conversion was done correctly.

We also study basic statistical information: minimum, maximum, average, etc. (Fig. 8.4):

```
weatherSensorDf.describe()
```

	TEMP	HUMD	RSSI	VCC
count	1392.000000	1392.000000	1392.000000	1392.000000
mean	13.461602	75.639325	-72.403247	3.097399
std	5.641691	17.412619	5.452205	0.004884
min	1.410000	23.540000	-87.830000	3.070000
25%	9.607500	64.250000	-75.250000	3.100000
50%	12.960000	79.600000	-72.500000	3.100000
75%	16.572500	90.172500	-70.080000	3.100000
max	37.370000	101.180000	-41.000000	3.100000

Fig. 8.4. Display of basic statistical information

We see that the maximum temperature value is 37.37°C and the minimum is 1.41°C. The average value is 13.462°C. The same information can be viewed for other columns. In addition, in the above statistical information, we can see the standard deviation value, percentiles of the 25%, 50%, 75% level.

8.2. Descriptive and predictive big data analytics

Descriptive big data analytics is the process of analyzing and interpreting large amounts of data to identify key characteristics, patterns, and trends, which helps to understand the structure and characteristics of a dataset without building predictive models. Descriptive analytics helps answer the questions "What happened?" and "What patterns are observed?"

Key aspects of descriptive big data analytics are statistics and summarization, data visualization, grouping and summarization, pattern exploration, and anomaly detection.

Data statistics involves calculating basic statistical parameters such as mean, median, mode, and standard deviation.

Summarization is the calculation of total indicators, for example, total quantity, average cost, etc. Example: calculating the average cost of purchases in a large dataset of an online store.

Data visualization is the use of graphs, charts, and other visual elements to represent data distributions, such as creating scatter plots, histograms, pie charts, etc. Example: creating a histogram to display the distribution of student grades in a large university dataset.

Grouping data by certain characteristics is done to obtain grouped results. Example: grouping users by activity level and calculating the average time spent on the platform.

An example of studying correlations and dependencies is analyzing the relationship between time spent on a website and the number of purchases made. An example of using analysis to detect unexpected or anomalous values, study outliers, and deviations from the expected is the detection of anomalous values in a dataset of temperature data for a specific region.

Predictive big data analytics is a branch of data analysis that uses various techniques and algorithms to predict future events or trends based on large amounts of data. The main goal is to identify possible outcomes and relationships for making business decisions or management actions. Key aspects of predictive big data analytics include building predictive models, optimizing processes and resources, marketing strategies and personalization, risk management and security measures, and assessing customer behavior.

Building predictive models is the process of training models on historical data and using them to predict future values, using machine learning

algorithms to create predictive models. An example is predicting product sales volume based on previous monthly sales.

Process and resource optimization is the use of forecasts to optimize production processes and resource allocation, minimizing costs and maximizing efficiency based on predictions. For example, in logistics, delivery routes can be optimized using forecasts of demand for goods in different regions, reducing fuel costs and delivery times, and ensuring timely replenishment of inventory. Marketing strategies and personalization is the use of predictive analytics to adapt marketing strategies and personalize offers to customers, predicting customer reactions to advertising campaigns and offers. Example: personalized offers of goods or services based on analysis of customer purchasing habits. Risk management and security measures are the use of predictive analytics to anticipate risks and take security measures, analyze anomalies and anticipate possible threats. An example is predicting the likelihood of cyberattacks and taking measures to protect the network. Customer behavior analysis is the analysis of customer interaction data to predict their future behavior, creating recommendation models based on previous choices and actions. An example is predicting which products a customer might be interested in based on their previous purchases and views.

Predictive big data analytics uses the power of machine learning algorithms to uncover patterns in data and provide useful predictions to support decision-making.

Descriptive big data analytics involves analyzing historical data to identify patterns and trends, such as studying customer behavior based on past purchases for market segmentation.

Descriptive statistics are used to describe or summarize the values and observations in a data set. For example, a fitness tracker recorded a person’s step count and heart rate over a period of 10 days. If the person achieved their fitness goals on 6 out of 10 days, that means they were successful 60% of the time. We might also note that their maximum heart rate during that period was 140 beats per minute and their average heart rate was 72 beats per minute. These observations are examples of descriptive statistics that help describe and simplify a data set.

Basic descriptive statistics can include the total number of data points in a set, the range of values for that data point, or the number of times a value occurs. Descriptive statistics can also help answer questions about whether a

given value is trending. The answers to these questions can be presented numerically or graphically. The results of descriptive statistics are often presented in the form of pie charts, bar graphs, or histograms.

It is important to note that although descriptive statistics describe the current or past state of the population under study, it does not allow for comparison of groups, inferences, or predictions about other data sets that are not part of the population. For example, in the case of a fitness tracker, we cannot conclude that a person is in poor health just because they achieved their goal 60% of the time. Nor can we use this data set to predict outcomes for other people with similar characteristics.

Predictive big data analytics uses machine learning models and statistical algorithms to predict future events, such as predicting customer churn based on analysis of their current interactions and transaction history.

8.3. Correlation analysis of big data

Big data correlation analysis determines the degree of relationship between two variables, allowing to measure and analyze how much one variable changes with another.

Pandas and Numpy libraries are used for correlation analysis of big data. Consider an example of correlation analysis in Python:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# Generate random data for the example
np.random.seed(42)
data = pd.DataFrame({
'Variable_1': np.random.rand(100),
'Variable_2': np.random.rand(100),
'Variable_3': np.random.rand(100)
})
# Output the first 5 rows of data
print(data.head())
# Correlation matrix
correlation_matrix = data.corr()
# Visualization of the correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation matrix')
plt.show()
```

The purpose of this code is analyze the relationship between variables in the generated dataset by calculating a correlation matrix and visualizing it as a heat map. This allows users to identify which variables have a strong positive or negative correlation, as well as better understand the structure of the data.

In this example, we use the `pandas library` to create `a DataFrame` with random data. Then, we use the `corr() method` to calculate the correlation matrix.

The code ends with a visualization of the correlation matrix using the method `heatmap` from the `seaborn library`. This graphical representation allows users to quickly assess the degree of correlation between different variables. The result of the program execution is shown in Fig. 8.5.

```
Variable_1 Variable_2 Variable_3
0 0.374540 0.031429 0.642032
1 0.950714 0.636410 0.084140
2 0.731994 0.314356 0.161629
3 0.598658 0.508571 0.898554
4 0.156019 0.907566 0.606429
```

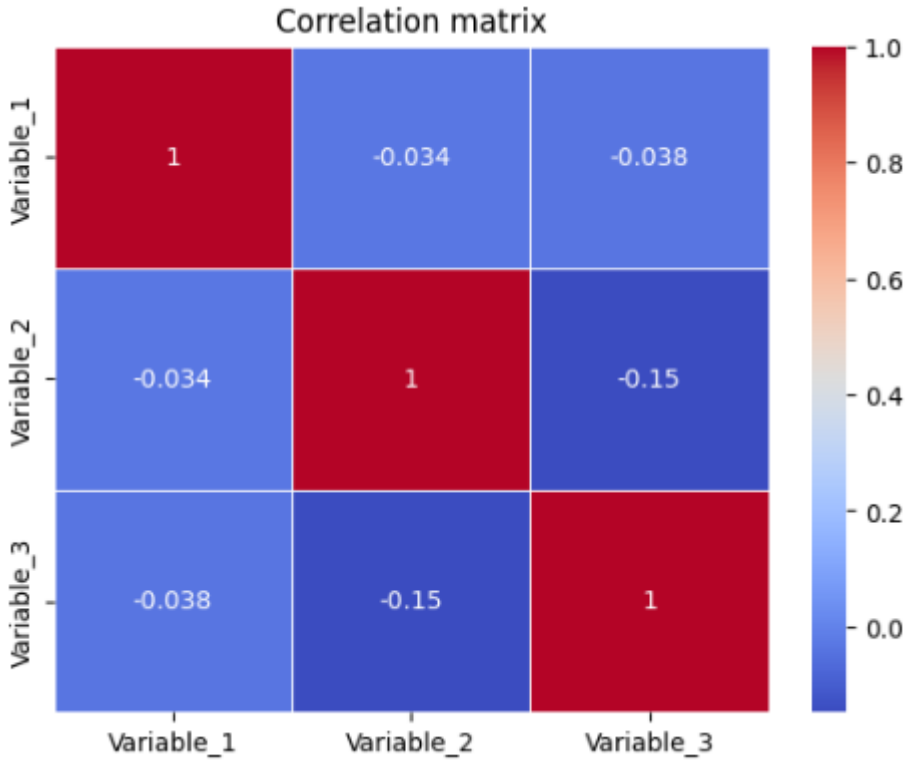


Fig. 8.5. Correlation matrix

For real big data, similar approaches can be applied, but using libraries that are optimized for processing large amounts of data, such as Dask, Vaex, or others, depending on the needs of the research.

8.4. Regression analysis of big data

Big data regression analysis is an analysis method that studies the relationship between one or more independent variables and a dependent variable.

The main steps of regression analysis are:

- data loading;
- data pre-processing;
- building a linear regression model;
- visualization of results.

In the context of big data, optimized libraries such as Dask or Vaex can be used for regression analysis to efficiently process large amounts of data.

Consider an example of code where linear regression is implemented to analyze the relationship between a dependent variable and an independent variable, using synthetic data for demonstration purposes.

The code starts by creating a synthetic dataset with 100 data points.

The independent variable X is a set of random numbers scaled to range between 0 and 10.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
# Generate synthetic data for the example
np.random.seed(42)
data = pd.DataFrame({
    'X': np.random.rand(100) * 10,  # Independent variable
    'Noise': np.random.randn(100)  # Random noise
})
data['Y'] = 3 * data['X'] + 5 + data['Noise']  # Dependent variable with noise
```

The dependent variable Y is calculated as a linear function of X (with a slope of 3 and an intercept of 5) while adding random noise to simulate real-world variability.

```
# Visualize the data
plt.scatter(data['X'], data['Y'], alpha=0.7)
plt.title('Scatter plot of X vs Y')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
# Splitting the data into training and testing sets
X = data[['X']]  # Independent variable
y = data['Y']  # Dependent variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
# Training the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Making predictions
y_pred = model.predict(X_test)
# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
# Output model coefficients and performance metrics
print(f"Model Coefficients: {model.coef_[0]:.2f}")
print(f"Model Intercept: {model.intercept_:.2f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
```

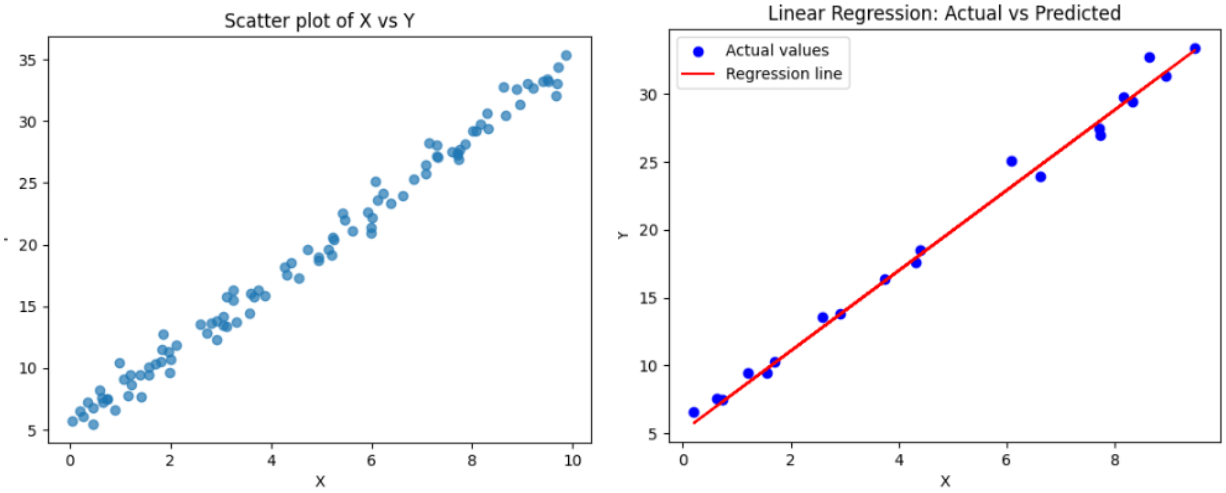
A scatter plot of X vs Y is displayed to visualize the relationship. This is an essential step to determine if linear regression is appropriate for the data. In this example, we can see a positive linear trend.

The dataset is divided into training (80%) and testing (20%) subsets. This ensures the model is trained and tested on separate data, preventing overfitting. A LinearRegression model from the sklearn library is trained using the training data. The model learns the relationship between X and Y. Predictions are then made on the test set.

The Mean Squared Error (MSE) measures the average squared difference between actual and predicted values, while the R-squared (R^2) value indicates how well the model explains the variability in the data.

```
# Visualizing the regression line
plt.scatter(X_test, y_test, color='blue', label='Actual values')
plt.plot(X_test, y_pred, color='red', label='Regression line')
plt.title('Linear Regression: Actual vs Predicted')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

The regression line is plotted over the test data. The actual values (blue points) and predicted trend (red line) help evaluate the model visually (Fig. 8.6).




```
Model Coefficients: 2.96
Model Intercept: 5.14
Mean Squared Error: 0.65
R-squared: 0.99
```

Fig. 8.6. Linear regression implemented to analyze the relationship between a dependent variable and an independent variable

Consider an example of regression analysis using Python. First, we'll download a dataset of house prices from an open source. Then, we'll remove rows with missing values to simplify the processing.

We'll choose an independent variable (median income) and a dependent variable (median house price).

```python
# Loading required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Loading data
url = "https://raw.githubusercontent.com/ageron/handson-ml2/master/datasets/housing/housing.csv"
data = pd.read_csv(url)
# Data preprocessing
data = data.dropna() # Remove rows with missing values
# Choosing the independent variable (number of rooms) and the dependent variable (housing price)
X = data[['median_income']]
y = data['median_house_value']
```

```
# Splitting the data into training and test samples
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Building a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Prediction on the test sample
y_pred = model.predict(X_test)
# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
# Visualization of results
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_test['median_income'], y=y_test, label='Actual')
sns.lineplot(x=X_test['median_income'], y=y_pred, color='red', label='Predicted')
plt.xlabel('Average Income')
plt.ylabel('Average house price')
plt.title('Linear Regression: Average Income vs. Average House Price')
plt.legend()
plt.show()
```

This code analyzes the relationship between median income and median home value using a linear regression model. First, it imports libraries for data manipulation, model building, and visualization. Then, it loads a housing dataset and preprocesses it by removing rows with missing values.

To build the model, `the median_income variable is chosen` as the independent (predictor) and `the median_house_value` is chosen as the dependent (target). The data is split into training and test sets in a ratio of 80% to 20%, after which a linear regression model is built and trained on the training set. Based on the test data, predictions are made, the results are evaluated using the mean square error (MSE) and the coefficient of determination (R^2), which are output to the console. For visualization, a graph is created where the points represent the actual values, and the line represents the predicted values, which allows users to assess the quality of the model in the context of the relationship between income and house prices.

Result of program execution:

```
Mean Squared Error: 7221011204.235033
R^2 Score: 0.47196228574894983
```

The resulting graph shows the relationship between median income and median home value, with the blue dots representing the actual data and the red line representing the predictions of a linear regression model. It can be used to analyze trends, such as confirming that home values increase with income, and to assess the quality of the model. If the dots are close to the line, this indicates a high level of accuracy in the model, while significant

deviations may indicate the need for refinement or more sophisticated modeling techniques (Fig. 8.7).

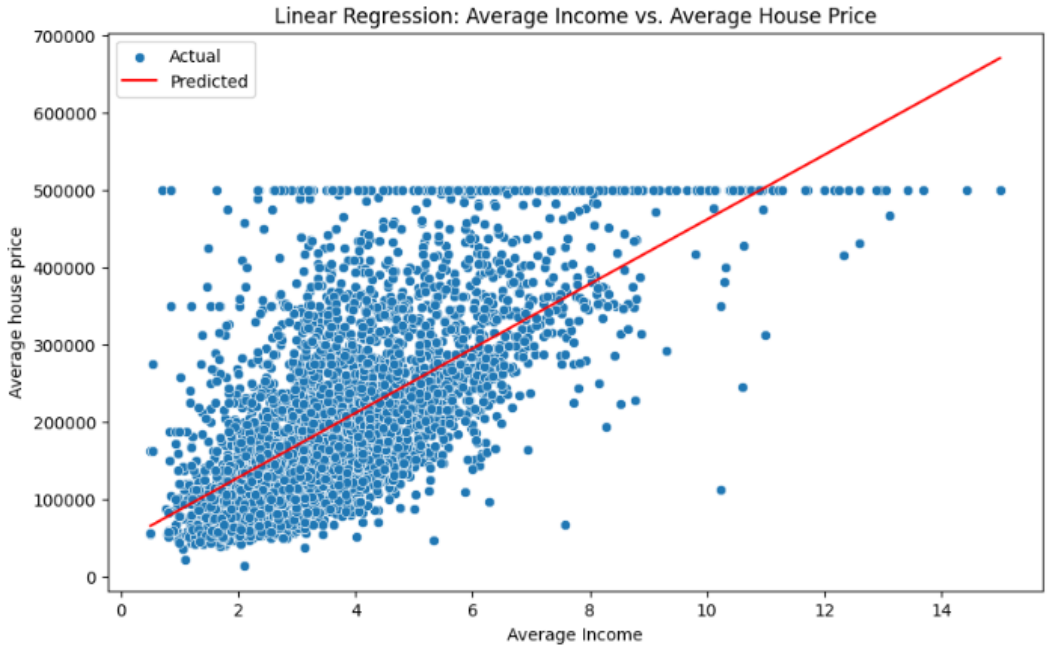


Fig. 8.7. Building a regression model

Consider building a multiple linear regression model to predict academic performance based on the above factors that influence students' academic performance. The dataset under study is "Student Performance (Multiple Linear Regression)" contains 10,000 student records, each of which includes information about various predictors and an index of success (Fig. 8.8).

Dataset to determine how various factors affect the Student's Performance

# Hours Studied Numeric	# Previous Scores Numeric	✓ Extracurricular A... Categorical	# Sleep Hours Numeric	# Sample Question ... Numeric	# Perfor Target Va
1 9	40 99	true 4948 49% false 5052 51%	4 9	0 9	10
7	99	Yes	9	1	91.0
4	82	No	4	2	65.0
8	51	Yes	7	2	45.0
5	52	Yes	5	2	36.0
7	75	No	8	5	66.0
3	78	No	9	6	61.0
7	73	Yes	5	6	63.0

Fig. 8.8. Dataset «Student Performance (Multiple Linear Regression)»

Possible predictors may include demographic data (age, gender), academic performance (grades, attendance), socioeconomic status, and extracurricular activities of students. To begin, we load all the necessary libraries and create a dataframe from the loaded dataset.

```
[ ]  %matplotlib inline
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import folium
     from sklearn.linear_model import LinearRegression
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import PolynomialFeatures
     from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
     from sklearn.inspection import permutation_importance
```

```
[ ]  dataset_path = '/content/Student_Performance.csv'
     DF = pd.read_csv(dataset_path)
```

Matrix correlations (Fig. 8.9) shows that the strongest positive correlation with the index (`Performance Index`) is observed for the previous grades (`Previous Scores, 0.951`) and learning time (`Hours Studied`) has moderate positive impact (0.374).

```
[ ] DF.corr(method='pearson')
```

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index
Hours Studied	1.000000	-0.012390	0.001245	0.017463	0.373730
Previous Scores	-0.012390	1.000000	0.005944	0.007888	0.915189
Sleep Hours	0.001245	0.005944	1.000000	0.003990	0.048106
Sample Question Papers Practiced	0.017463	0.007888	0.003990	1.000000	0.043268
Performance Index	0.373730	0.915189	0.048106	0.043268	1.000000

Fig. 8.9. Correlation between selected factors and productivity index

In the code below is carried out separation dataset for training and testing data for construction models regressions: variables X (independent : number of prepared questions, hours of sleep, previous grades, hours learning) and y (dependent : index success) are divided into training (`X_train, y_train`) and test (`X_test, y_test`) sets. So too implemented construction two regression models : linear regressions for prediction index success rate (`y_pred_linear`) using linear dependencies between variables and polynomial second- order regression (`y_pred_poly`), where the input variables are transformed into polynomials signs that allows to take into account nonlinear connections for

promotion accuracy predictions.

```
[ ]  X = DF[['Sample Question Papers Practiced','Sleep Hours','Previous Scores','Hours Studied']]
     y = DF['Performance Index']

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[ ]  X = DF[['Sample Question Papers Practiced','Sleep Hours','Previous Scores','Hours Studied']]
     y = DF['Performance Index']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
     linear_model = LinearRegression()
     linear_model.fit(X_train, y_train)
     y_pred_linear = linear_model.predict(X_test)
     poly = PolynomialFeatures(degree=2)
     X_train_poly = poly.fit_transform(X_train)
     X_test_poly = poly.transform(X_test)
     poly_model = LinearRegression()
     poly_model.fit(X_train_poly, y_train)
     y_pred_poly = poly_model.predict(X_test_poly)
```

The code below defines a function `print_metrics` that calculates and prints the model quality metrics (MAE, MSE, R^2) for predictions (`y_pred`) compared to true values (`y_true`) and specifies the model name (`model_name`). The function is then called to compare linear and polynomial regression. Next, the predictors are analyzed: a table of model coefficients is created and their absolute influence is determined, and the importance of the predictors is calculated using `permutation_importance` (a random permutation method), where variables are sorted by importance for building predictions.

```
def print_metrics(y_true, y_pred, model_name):
print(f"{model_name}:")
print(f"MAE: {mean_absolute_error(y_true, y_pred):.2f}")
print(f"MSE: {mean_squared_error(y_true, y_pred):.2f}")
print(f"R²: {r2_score(y_true, y_pred):.2f}\n")
print_metrics(y_test, y_pred_linear, "Linear Regression")
print_metrics(y_test, y_pred_poly, "Polynomial Regression")
coef_df = pd.DataFrame({
'Predictor': X.columns,
'Coefficient': linear_model.coef_,
'Absolute Impact': np.abs(linear_model.coef_)
}).sort_values('Absolute Impact', ascending=False)
print("Top predictors by influence:")
print(coef_df[['Predictor', 'Coefficient']])
perm_importance = permutation_importance(linear_model, X_test, y_test, n_repeats=10)
```

The graph in Fig. 8.10 shows a comparison of actual and predicted performance values, where the red line represents a perfect model fit.

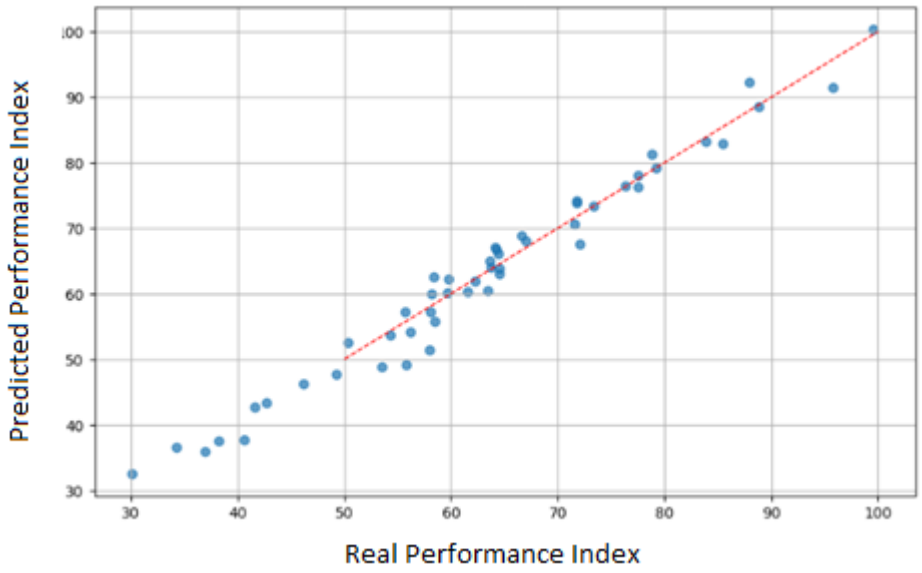


Fig. 8.10. Comparison of actual and predicted productivity values

Control questions and tasks for topic 8

List of theoretical questions

1. What are the steps involved in the process of researching and preparing data for analyzing large amounts of information?
2. What descriptive big data analytics methods do you find most effective for understanding the structure and characteristics of a dataset?
3. How can correlation analysis be used to identify relationships between different variables in big data?
4. How can regression analysis help in predicting future values based on relationships between variables?
5. What practical applications of big data correlation analysis can you point out for different fields such as business, medicine, or science?
6. What aspects of big data research are key to successfully implementing predictive analytics?
7. How can the results of descriptive analytics be used to optimize management strategies and decision-making in large companies?
8. What problems can arise in regression analysis of big data, and how can they be overcome?
9. How to use data visualization to effectively present the results of correlation and regression analysis?
10. How can knowledge gained from descriptive and predictive analytics be used to improve big data risk management strategies?

List of tasks for independent work

Practical task 1. Correlation analysis of data.

Using a dataset on demographic and economic indicators, find the correlation between the level of education of the population and average income. Link to the dataset: https://archive.ics.uci.edu/dataset/2/adult

Step 1. Load the dataset and perform preprocessing.

Step 2. Calculate the correlation matrix for the numerical variables.

Step 3. Visualize the correlation between education level and average income using a heat map and a pairwise graph.

Practical task 2. Regression analysis of data.

Using a dataset of car characteristics, build a regression model to predict a car's fuel consumption based on its characteristics. Link to the dataset: https://archive.ics.uci.edu/dataset/10/automobile

Step 1. Load the dataset and perform preprocessing.

Step 2. Split the data into training and test samples.

Step 3. Build a linear regression model to predict fuel consumption, evaluate the model, and visualize the results.

List of used and recommended literature

8.1. Data Science - Statistics Correlation [Electronic resource]. – Access mode: https://www.w3schools.com/datascience/ds_stat_correlation.asp

8.2. Data Science - Statistics Correlation Matrix [Electronic resource]. – Access mode: https://www.w3schools.com/datascience/ds_stat_correlation_matrix.asp

8.3. Data Science - Linear Regression [Electronic resource]. – Access mode: https://www.w3schools.com/datascience/ds_linear_regression.asp

8.4. Data Science - Regression Table - Coefficients [Electronic resource]. – Access mode: https://www.w3schools.com/datascience/ds_linear_regression_coeff.asp

8.5. Student Performance [Electronic resource]. – Access mode: https://www.kaggle.com/datasets/nikhil7280/student-performance-multiple-linear-regression

Topic 9. Software methods for classification and clustering of big data

Big data classification is the process of assigning labels or categories to new data based on patterns learned from previously labeled data. Big data clustering is an analysis method that involves automatically grouping similar data into clusters based on their shared characteristics without any predefined labels.

9.1. Big Data classification methods

Let's look at several popular big data classification methods with code examples in the Python programming language.

Random forest is a classification method that uses an ensemble of solutions (decision trees). Random Forest works on the principle of ensemble learning, creating a large number of independent decision trees based on random subsets of the input data and random subsets of features. Each tree is trained on its own data set, obtained by the bootstrap method (random sampling with replacement). To classify new data, each tree gives its own prediction, and the final decision is made by majority vote. In regression problems, the average value of the predictions of all trees is taken. This approach increases the accuracy of the model, reduces the risk of overfitting, and provides robustness to noise in the data.

Consider the main stages of the Random Forest method.

Stage 1. Data preparation.

Input data: training dataset with features (X) and labels (y).

Output: prepared data for building decision trees.

At this stage, the data is cleaned and prepared for use in the model.

Stage 2. Bootstrap Aggregation.

Input data: training dataset.

At this stage, random subsets of the data are created using bootstrap sampling. This allows each tree to be trained on different data sets. A subset of the data is randomly selected from the dataset with bootstrap sampling. This process is repeated for every tree in the forest.

Stage 3. Building Decision Trees.

Login or data: each bootstrap-aggregated subset of data.

Each decision tree is built on a subset of the data. Randomly selecting features at each node helps reduce correlation between trees. The decision tree is built until it is fully branched or reaches its maximum depth.

Stage 4. Prediction.

Login or data: new data (test sample).

Each decision tree predicts an outcome for the test sample. The results of each tree are combined. The majority voting method is used for classification. For regression, the average value of the predictions is taken.

Stage 5. Combining results.

Input no data: predictions from all trees.

Output: final prediction for the test sample.

For classification tasks, the majority voting method is used to determine the class, and for regression tasks, the mean value of the predictions is used. Consider an example of a software implementation of the Random Forest method in Python:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Reading data
X, y = load_data()
# Splitting into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Building a Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Predictions for the test set
y_pred = model.predict(X_test)
# Accuracy assessment
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

This code performs data classification using the Random Forest algorithm. First, the data is read using the `load_data() function`, then it is split into training and test sets using the `train_test_split function`, where 20% of the data is allocated for testing. Then, a Random Forest model (`RandomForestClassifier`) is created with 100 trees and a random state of 42 for reproducibility of the results. The model is trained on the training data (`X_train, y_train`) and used to predict on the test data (`X_test`).

The accuracy of the model is assessed using an accuracy metric (`accuracy_score`), which determines the proportion of correctly classified samples, after which the resulting accuracy value is displayed.

Gradient boosting is a method that improves accuracy by sequentially adding models.

Consider the main stages of the Gradient Boosting method.

Stage 1. Data preparation.

Input data: training dataset with features (X) and labels (y).

Entrance: prepared data for building models.

Stage 2. Model initialization.

At this stage, a basic model is created (for example, predicting the mean value for regression or the class with the highest frequency for classification).

Stage 3. Sequential construction of models.

At this stage, residuals are calculated as the difference between the actual values and the predictions of the current model, a new model is built on the residuals, and the overall forecast is updated by adding the predictions of the new model with a certain weight (learning rate).

Stage 4. Model update.

is updated after each iteration, repeating steps 3 and 4 until the specified number of iterations is reached or until the error is reduced to the required level.

Stage 5. Forecasting.

Input no data: new data (test sample).

At this stage, the final model is used to predict on new data. Consider an example of a software implementation of the Gradient Boosting method in Python:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Reading data
X, y = load_data()
# Splitting into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Building a Gradient Boosting model
model = GradientBoostingClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Predictions for the test set
y_pred = model.predict(X_test)
# Accuracy assessment
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

This code uses the Gradient Boosting algorithm to classify data. First, the data is read using the `load_data() function` and split into training

and test sets in a ratio of 80:20 using `train_test_split` . Then, a `GradientBoostingClassifier model is created` with 100 base estimators (`n_estimators=100`) and trained on the training data `X_train` and `y_train`. After training the model, class predictions are performed on the test set `X_test`. To evaluate the quality of the model, the accuracy is calculated `by` comparing the predicted values `y_pred` with the actual labels `y_test` , and the result is displayed on the screen.

Deep learning is the use of neural networks for classification. Neural networks consist of multiple layers of neurons, where each layer performs specific calculations to process and transmit information.

Consider the main stages of neural networks.

Stage 1. Data preparation.

Input data : training dataset with features (X) and labels (y).

Output: prepared data for training the model.

Stage 2. Model initialization.

At this stage, the neural network architecture is defined (number of layers, number of neurons in each layer) and weights and biases are initialized.

Stage 3. Forward propagation.

At this stage , the outputs of each layer are calculated using weights, biases, and functions. Activation. The outputs of the current layer are passed as input data for the next layer.

Stage 4. Calculation of the loss function (loss function).

At this stage, the deviation between the predicted values and the actual labels is determined, and a loss function is calculated to quantify the model error.

Stage 5. Backward Propagation.

are calculated with respect to the weights and biases using chain rule methods, and the weights and biases are updated to minimize the loss function (using optimizers such as gradient descent).

Step 6. Update model parameters.

At this stage, the weights and biases are updated based on the calculated gradients and the selected optimization method.

Step 7. Repeat the process.

At this stage, steps 3-6 are repeated for a given number of epochs or until the required accuracy is achieved.

Stage 8. Forecasting.

Input: new data (test sample).

It occurs by using the trained weights and biases to make predictions on new data.

Consider the example code in Python (using the TensorFlow library):

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Reading data
X, y = load_data()
# Splitting into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Building a neural network model
model = tf.keras.Sequential([
tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Model training
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)
# Accuracy assessment
_, accuracy = model.evaluate(X_test, y_test)
print(f'Accuracy: {accuracy}')
```

This code implements a machine learning process for binary classification using the TensorFlow library. First, the data is read and split into training and test sets using `train_test_split`. Next, a neural network with two layers is created: the first layer contains 128 neurons with ReLU activation, and the second layer contains one neuron with sigmoid activation.

The network is compiled with the Adam optimizer and the `binary_crossentropy loss function`. The model is trained on the training data for 10 epochs with a batch of 64 samples, with 20% of the data used for validation. Finally, the model's accuracy is evaluated on the test set and displayed.

Consider an example of image classification using open data from the Fashion MNIST dataset, which contains grayscale images of clothing items such as shirts, shoes, and bags. We will use a simple neural network with TensorFlow and Keras for model creation, training, and evaluation.

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import fashion_mnist
```

```
    from sklearn.metrics import classification_report,
confusion_matrix
    import matplotlib.pyplot as plt
    import seaborn as sns
    # Load the Fashion MNIST dataset
    (X_train, y_train), (X_test, y_test) =
fashion_mnist.load_data()
    # Class labels for the dataset
    class_labels = [
        'T-shirt/top', 'Trouser', 'Pullover', 'Dress',
'Coat',
        'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle
boot'
    ]
    # Normalize the images
    X_train, X_test = X_train / 255.0, X_test / 255.0
    # Display the first 5 images with their labels
    fig, axes = plt.subplots(1, 5, figsize=(15, 5))
    for i, ax in enumerate(axes):
        ax.imshow(X_train[i], cmap='gray')
        ax.set_title(f"Label:
{class_labels[y_train[i]]}")
        ax.axis('off')
    plt.tight_layout()
    plt.show()
    # Build the neural network model
    model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])
    # Compile the model
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    # Train the model
```

```
    model.fit(X_train, y_train, epochs=10,
validation_split=0.2)
    # Evaluate the model
    test_loss, test_accuracy = model.evaluate(X_test,
y_test, verbose=2)
    print(f"Test Accuracy: {test_accuracy:.2f}")
    # Make predictions
    y_pred = model.predict(X_test)
    y_pred_labels = tf.argmax(y_pred, axis=1).numpy()
    # Confusion matrix
    conf_matrix = confusion_matrix(y_test,
y_pred_labels)
    plt.figure(figsize=(10, 8))
    sns.heatmap(conf_matrix, annot=True, fmt='d',
cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()
    # Classification report
    class_report = classification_report(y_test,
y_pred_labels, target_names=class_labels)
    print("Classification Report:")
    print(class_report)
```

Fashion MNIST dataset, a standard benchmark for image classification, is used. It includes 70,000 grayscale images, each of size 28×28, across 10 categories.

The dataset is split into training and test sets, and the pixel values are normalized to the range [0, 1] to improve training stability.

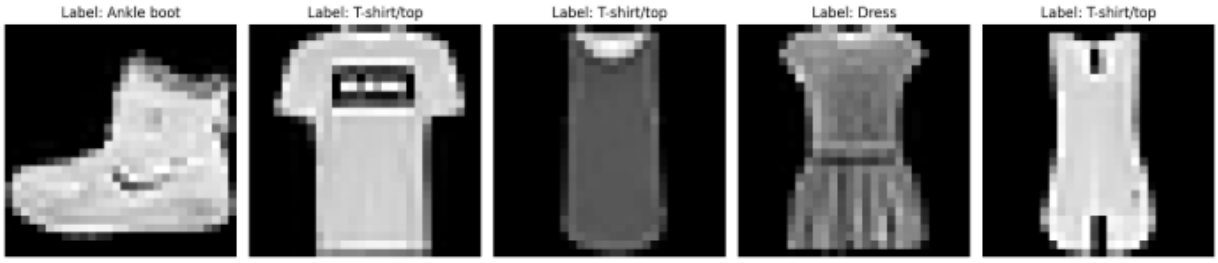
The first few images from the training set are displayed with their corresponding class labels, helping to understand the data visually.

A simple feedforward neural network is constructed using TensorFlow and Keras. It consists of a flattening layer to convert the 2D images into 1D vectors, followed by one hidden layer with 128 neurons (ReLU activation) and an output layer with 10 neurons (softmax activation for multi-class

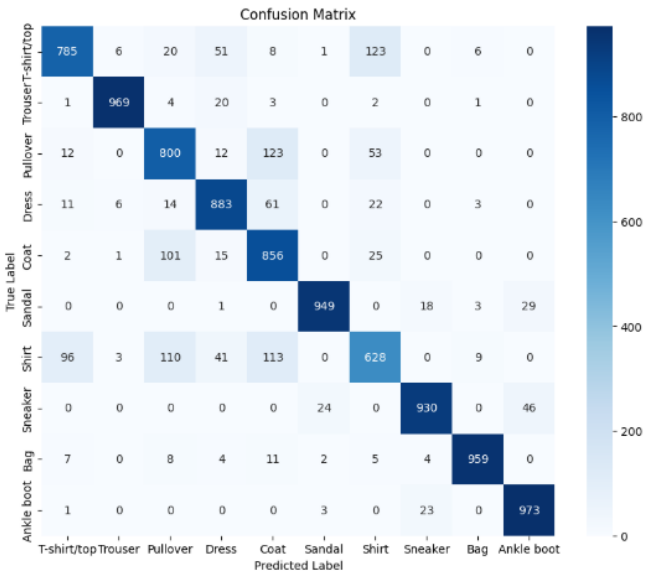
classification). The model is trained using the Adam optimizer and sparse categorical cross-entropy loss for 10 epochs with 20% of the training data reserved for validation. The model's accuracy is evaluated on the test set, and a confusion matrix is visualized using Seaborn.

A detailed classification report, including precision, recall, and F1-score for each class, is generated, offering a more granular view of the model's performance. This example demonstrates how to handle open datasets, preprocess data, build (Fig. 9.1).

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```




```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.7706 - loss: 0.6683 - val_accuracy: 0.8550 - val_loss: 0.4099
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.8584 - loss: 0.3953 - val_accuracy: 0.8676 - val_loss: 0.3670
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.8745 - loss: 0.3446 - val_accuracy: 0.8633 - val_loss: 0.3644
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.8810 - loss: 0.3219 - val_accuracy: 0.8754 - val_loss: 0.3442
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 5ms/step - accuracy: 0.8906 - loss: 0.2988 - val_accuracy: 0.8699 - val_loss: 0.3489
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.8928 - loss: 0.2910 - val_accuracy: 0.8790 - val_loss: 0.3301
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.8996 - loss: 0.2680 - val_accuracy: 0.8844 - val_loss: 0.3165
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9008 - loss: 0.2625 - val_accuracy: 0.8842 - val_loss: 0.3246
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9099 - loss: 0.2423 - val_accuracy: 0.8889 - val_loss: 0.3229
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9112 - loss: 0.2415 - val_accuracy: 0.8885 - val_loss: 0.3219
313/313 - 1s - 3ms/step - accuracy: 0.8732 - loss: 0.3560
Test Accuracy: 0.87
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```



Classification Report:

	precision	recall	f1-score	support
T-shirt/top	0.86	0.79	0.82	1000
Trouser	0.98	0.97	0.98	1000
Pullover	0.76	0.80	0.78	1000
Dress	0.86	0.88	0.87	1000
Coat	0.73	0.86	0.79	1000
Sandal	0.97	0.95	0.96	1000
Shirt	0.73	0.63	0.68	1000
Sneaker	0.95	0.93	0.94	1000
Bag	0.98	0.96	0.97	1000
Ankle boot	0.93	0.97	0.95	1000
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000

Fig. 9.1. Example of image classification using data from the Fashion MNIST dataset

9.2. Big data clustering methods

Big data clustering is the process of grouping large-scale unstructured or structured data into clusters based on the similarity of their characteristics to uncover hidden patterns and facilitate analysis. There are several clustering methods that aim to work efficiently with large amounts of data. Big data clustering methods include K-means, hierarchical clustering, DBSCAN (Density-Based Spatial Clustering of Applications with Noise), and dimensionality reduction algorithms such as t-SNE and UMAP.

K-means divides the data into k clusters, minimizing intra-cluster variation.

Hierarchical clustering creates a tree of clusters where each node represents an association of clusters at different levels of similarity.

DBSCAN detects clusters of various shapes and sizes, identifying dense regions of data.

The t-SNE and UMAP algorithms reduce the dimensionality of data, which helps visualize and detect clusters in high-dimensional data.

Let's look at a few examples of big data clustering methods with code in the Python programming language.

K-Means is an iterative clustering algorithm that partitions data into k clusters by minimizing the distance between points within the clusters and their centroids, using the method of means. Example of Python code:

```python
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
# Reading and preparing data
X = load_and_preprocess_data()
# Data standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Selecting the optimal number of clusters using the elbow method
# (in this example we believe that the optimal number is 3)
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)
# Evaluating clustering quality using silhouette score
silhouette_avg = silhouette_score(X_scaled, kmeans.labels_)
print(f'Silhouette Score: {silhouette_avg}')
```

This code performs data clustering using the K-means algorithm. First, the data is read and prepared using the `load_and_preprocess_data()` `function`, and then standardized using `StandardScaler` to normalize the features. Then, the optimal number of clusters is determined (in this example, 3) and clustering is performed using K - means with three clusters. The quality of the clustering is then evaluated using the silhouette score `silhouette_score`), which reflects how well the objects fit into their cluster compared to other clusters. The result is an average silhouette score, which gives an idea of the quality of the clustering.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that groups data points based on their density, can detect arbitrary-shaped clusters and separate noise points without the need to specify the number of clusters in advance.

Consider the Python code that performs data clustering using the DBSCAN algorithm. First, the data is read and prepared, and then it is standardized using `StandardScaler` to unify the feature scales. The DBSCAN algorithm is configured with the parameters `eps` (the maximum distance between points to combine them into a cluster) and `min_samples` (the minimum number of points to form a cluster) and is applied to the standardized data. To assess the quality of clustering, the Silhouette Score metric is used, which determines how well the objects are separated between the clusters; its average value is output to the console.

```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
# Reading and preparing data
X = load_and_preprocess_data()
# Data standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Select DBSCAN parameters (eps and min_samples)
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(X_scaled)
# Evaluating clustering quality using silhouette score
silhouette_avg = silhouette_score(X_scaled, dbscan.labels_)
print(f'Silhouette Score: {silhouette_avg}')
```

This code performs data clustering using the DBSCAN algorithm. First, the data is read and prepared using the `load_and_preprocess_data()` function, and then standardized using `StandardScaler` to normalize the features.

Then, clustering is performed using DBSCAN with parameters `eps=0.5` (maximum distance between points to be considered part of the same cluster) and `min_samples=5` (minimum number of points to form a cluster). The quality of clustering is then assessed using the silhouette score (`silhouette_score`), which reflects how well the objects fit into their cluster compared to other clusters. The result is an average silhouette score, which gives an idea of the quality of clustering.

Hierarchical Clustering is a method that builds a hierarchy of clusters by merging or splitting existing clusters. This method can work in two modes: agglomerative (bottom-up), where each point is initially considered a separate cluster and is gradually merged with others, and divisive (top-down), where all the data first forms a single cluster, which is then recursively divided into smaller subgroups.

In agglomerative clustering, we can group online store customers according to the similarity of their purchases, gradually combining users with similar preferences.

In a divisive approach, we can analyze genetic data by starting with a large group of samples and dividing them into subgroups based on hereditary characteristics. Hierarchical clustering is also used in text processing to classify articles by topic or in bioinformatics to identify relationships between species based on DNA sequences.

Consider the following Python code example:

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
# Reading and preparing data
X = load_and_preprocess_data()
# Data standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Selecting the number of clusters
agglomerative = AgglomerativeClustering(n_clusters=3)
agglomerative.fit(X_scaled)
# Evaluating clustering quality using silhouette score
silhouette_avg = silhouette_score(X_scaled, agglomerative.labels_)
print(f'Silhouette Score: {silhouette_avg}')
```

This code performs clustering of data using an agglomerative clustering algorithm. First, the data is read and prepared using the `load_and_preprocess_data() function`, and then standardized using `StandardScaler` to normalize features. Then, agglomerative clustering with three clusters is performed using `AgglomerativeClustering`, which groups the data into classes, starting with each individual feature and merging them into larger clusters. The quality of the clustering is then evaluated using a silhouette score (`silhouette_score`), which reflects how well the features fit into their cluster compared to other clusters. The result is an average silhouette score, which gives an idea of the quality of the clustering.

9.3. Application and software implementation of the K-means algorithm

K-means algorithm has a wide range of applications in various industries due to its simplicity and effectiveness in clustering data. In business, K-means is used for customer segmentation, which allows companies to develop targeted marketing campaigns, improve customer service, and personalize offers.

In finance, the algorithm can help detect anomalous transactions or cluster assets for risk analysis and portfolio optimization.

In medicine, it is used to classify patients according to disease characteristics or to detect patterns in medical images, which aids in

diagnosis. In the field of image processing, K-means is used for image segmentation, which makes it easier to distinguish objects or backgrounds in images.

In data science, the algorithm helps in reducing the dimensionality of data by grouping similar features, as well as in detecting structures in large data sets without predefined labels.

In industry, K-means can be used to cluster defects or parameters of manufacturing processes in order to improve product quality and optimize processes.

In social sciences, the algorithm helps in clustering social media users or in analyzing text to identify themes and trends.

Consider an example of a software implementation of the K-means algorithm for clustering a synthetic dataset:

```
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.datasets import make_blobs
    from sklearn.cluster import KMeans
    # Creating a synthetic dataset for the example
    X, _ = make_blobs(n_samples=300, centers=3,
random_state=42)
    # Visualization of a synthetic dataset
    plt.scatter(X[:, 0], X[:, 1], s=50, cmap='viridis')
    plt.title('Synthetic Dataset')
    plt.show()
    # Building a K-Means model using the scikit-learn
library
    kmeans = KMeans(n_clusters=3, random_state=42)
    kmeans.fit(X)
    # Get cluster centers and labels for each object
    centers = kmeans.cluster_centers_
    labels = kmeans.labels_
    # Visualization of results
    plt.scatter(X[:, 0], X[:, 1], c=labels, s=50,
cmap='viridis')
    plt.scatter(centers[:, 0], centers[:, 1], c='red',
s=200, marker='X')
```

```
plt.title('K-Means Clustering Results')
plt.show()
```

`make_blobs` function from the Scikit -learn library is used to generate a synthetic dataset with multiple centers.

Before starting clustering, we display the synthetic dataset to get a general idea of the distribution of points. Using `KMeans` from the Scikit-learn library, we create a K-means model with three clusters.

In the code `kmeans.fit(X)`, the `fit method` is used to train a K-means model on input data `X`.

The `kmeans object` is an instance of the `KMeans class`, which represents the K-means model. This object has various properties and methods for configuring and using the algorithm.

`Fit` method is one of the key methods for many machine learning models in the Scikit-learn library. In this context, using `kmeans.fit(X)` means that the K-means model will be trained on the input data X.

`X` is the input matrix that we want to use for clustering. Each row of the matrix represents one object, and the columns correspond to different features of these objects.

The call `kmeans.fit(X)` runs the K-means algorithm, which is trained on the input data `X`. During training, the model determines the centers of the clusters and assigns cluster labels to each feature based on their distance to the centers.

After calling `fit,` the model has new properties, such as `cluster_centers_` and `labels_`, which can be used to analyze and identify clusters for new data.

So, `kmeans.fit(X)` is the step where the model actually "learns" on the input data, and the results of this learning can then be used to cluster new data or gain information about the structure of clusters in the input data.

After clustering is complete, we obtain the coordinates of the cluster centers and labels for each object.

K-means We display the data as colored points, where each color corresponds to a cluster, and mark the cluster centers with red crosses (Fig. 9.2).

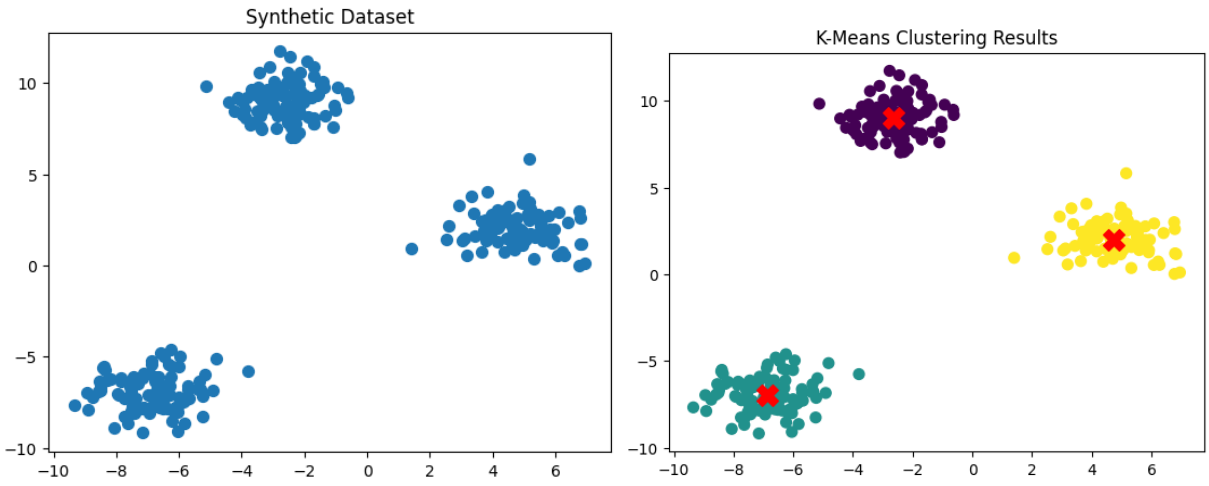


Fig. 9.2. Results of the algorithm execution K-Means for clustering a synthetic dataset

Consider a software implementation of patient clustering based on medical parameters such as glucose levels and body mass index (BMI). The results are visualized using a graph where the points are colored by clusters. The data is downloaded from an open source via URL (https://archive.ics.uci.edu/datasets), standardized and clustered using the K-means algorithm, which allows us to identify groups of patients with similar characteristics.

```
    import pandas as pd
    from sklearn.cluster import KMeans
    from sklearn.preprocessing import StandardScaler
    import matplotlib.pyplot as plt
    # Loading data from URL
    url =
"https://raw.githubusercontent.com/plotly/datasets/mast
er/diabetes.csv"
    data = pd.read_csv(url)
    # Data preparation
    X = data.drop('Outcome', axis=1)
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    # Applying K-means
    kmeans = KMeans(n_clusters=2, random_state=42)
    data['Cluster'] = kmeans.fit_predict(X_scaled)
    # Visualization of results
```

```
    plt.scatter(data['BMI'], data['Glucose'],
c=data['Cluster'], cmap='viridis')
    plt.xlabel('BMI')
    plt.ylabel('Glucose')
    plt.title('K-means clustering of patients with
diabetes')
    plt.colorbar(label='Cluster')
    plt.show()
```

This code performs clustering of diabetes patients based on their medical indicators using the K-means algorithm. First, the data is downloaded from the open source GitHub and stored in a dataframe. Next, the data is prepared for clustering: features are separated from the target variable `Outcome` , and the features are standardized to ensure the same scale. Then, the K-means algorithm is applied to cluster the data into two groups.

The clustering results are added to the dataframe as a new column `Cluster`. Finally, the results are visualized in a two-dimensional graph using the `BMI` and `Glucose parameters`, where different clusters are indicated by different colors (Fig. 9.3).

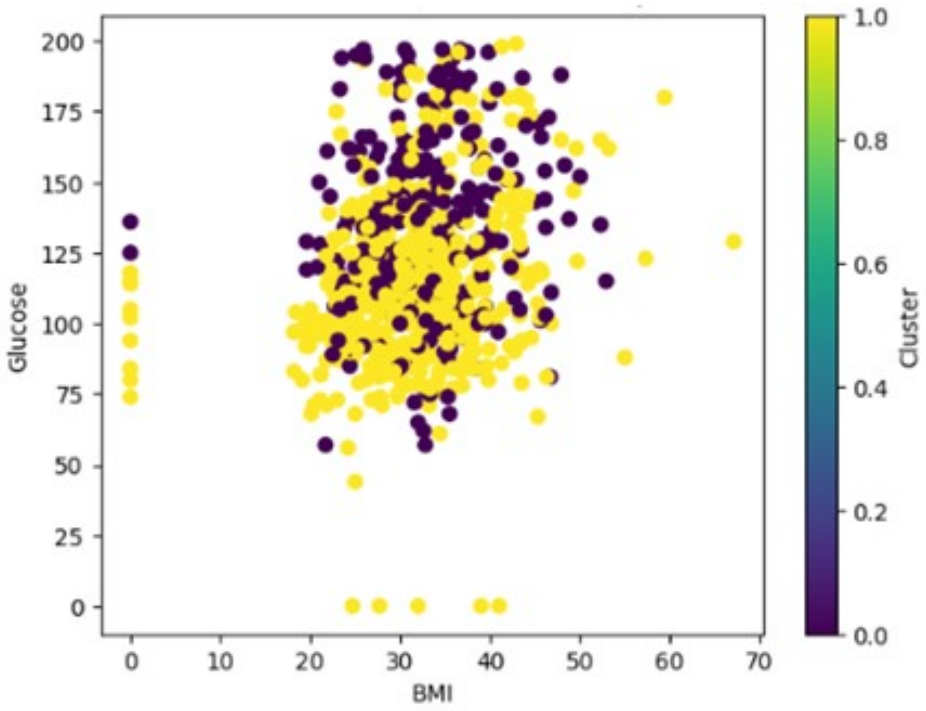


Fig. 9.3. Results of the algorithm execution K-means for clustering patients with diabetes

In the following example, we use the *titanic* dataset from the Seaborn library to cluster Titanic passengers.

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
# Loading the Titanic dataset from seaborn
data = sns.load_dataset('titanic').dropna()
# Data preparation
X = data[['age', 'fare']]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Applying K-means
kmeans = KMeans(n_clusters=3, random_state=42)
data['Cluster'] = kmeans.fit_predict(X_scaled)
# Visualization of results
plt.scatter(data['age'], data['fare'],
c=data['Cluster'], cmap='viridis')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.title('K-means clustering of Titanic passengers')
plt.colorbar(label='Cluster')
plt.show()
```

This code loads the *titanic dataset* using the Seaborn library, reads it into a `DataFrame`, and standardizes the selected metrics (age and fare) using `StandardScaler`.

The data is then clustered using the K-means algorithm into three groups. The clustering results are added to the dataframe as a new column `Cluster`. Finally, the results are visualized in a two-dimensional plot using the `age` and `fare parameters`, where different clusters are indicated by different colors (Fig. 9.4).

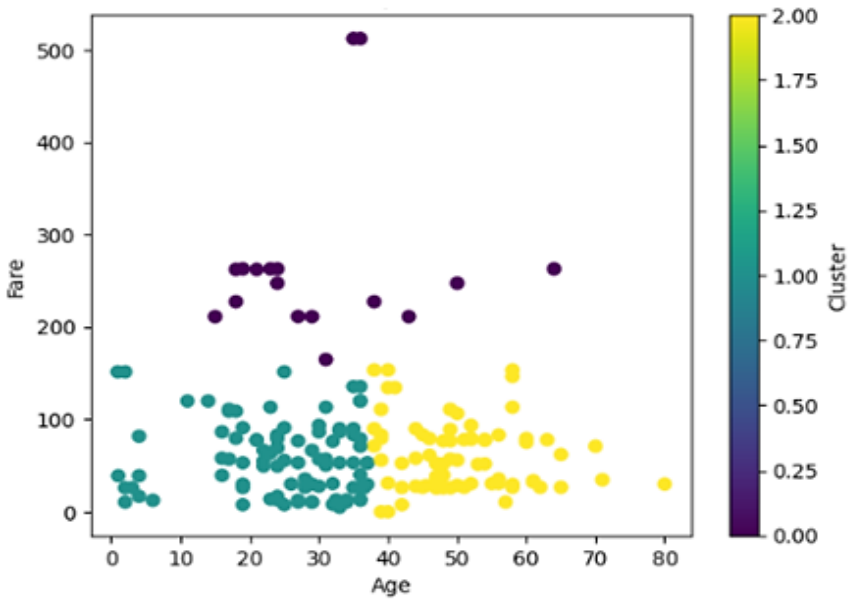


Fig. 9.4. Results of the algorithm execution K- means for clustering Titanic passengers

The results of K-means clustering of Titanic passengers show the distribution of data by two parameters: age (Age) and ticket price (Fare). The algorithm identified three main clusters, which are displayed in different colors. It can be seen that passengers with the highest fares form a separate cluster (dark purple), which may indicate their belonging to first class. The other two clusters (yellow and green) include passengers with medium and low fares, among whom there is greater variability in age. This indicates a certain pattern between age, social status and ticket price, which may be useful for further analysis of passenger survival.

In the following example, we will use the *iris dataset* from the Seaborn library, which is a classic example for clustering. The dataset consists of 150 iris flower samples from three species (Iris setosa, Iris versicolor, Iris virginica), containing measurements of the length and width of the sepals and petals of each flower.

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
# Loading the Iris dataset from seaborn
data = sns.load_dataset('iris')
# Data preparation
```

```
    X = data[['sepal_length', 'sepal_width',
'petal_length', 'petal_width']]
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    # Applying the K-means algorithm
    kmeans = KMeans(n_clusters=3, random_state=42)
    data['Cluster'] = kmeans.fit_predict(X_scaled)
    # Visualization of results
    plt.scatter(data['sepal_length'],
data['sepal_width'], c=data['Cluster'], cmap='viridis')
    plt.xlabel('Sepal Length')
    plt.ylabel('Sepal Width')
    plt.title('K-means Clustering of Iris Dataset')
    plt.colorbar(label='Cluster')
    plt.show()
```

This code loads the Iris dataset using the seaborn library, reads it into a `DataFrame`, and standardizes the selected metrics (sepal and petal length and width) using `StandardScaler`. The data is clustered using the K-means algorithm into three groups. The clustering results are added to the dataframe as a new column `Cluster`. The results are visualized on a two-dimensional graph using the `sepal_length` and `sepal_width parameters`, where different clusters are indicated by different colors (Fig. 9.5).

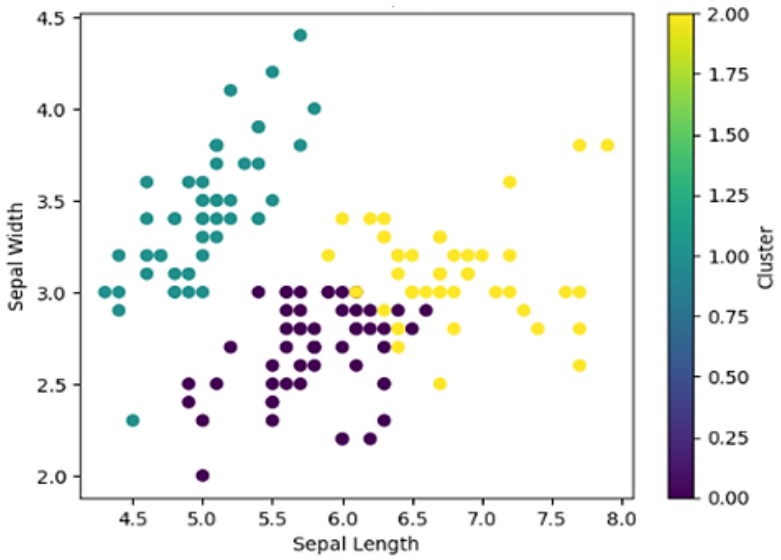


Fig. 9.5. Results of the algorithm execution K-Means for clustering iris flower samples from three species

The clustering results in Fig. 9.5 show how the K-means algorithm divided the flower samples into three clearly distinct clusters based on their sepal length and width. The purple dots (cluster 0) have relatively smaller sepal lengths and widths and are clustered at the bottom and left of the plot. The turquoise dots (cluster 1) have wider sepal widths at intermediate lengths and are mostly located at the top left. The yellow dots (cluster 2) have the largest sepal lengths and a wide spread in width, concentrated at the right of the plot. This clustering indicates that the iris samples can be divided into groups that have similar sepal length and width characteristics, which helps in studying and understanding the structure of the data.

The optimal number of clusters for K-means in large datasets is determined using methods such as the Elbow Method, Silhouette Score, and Gap Statistic. The Elbow Method estimates the sum of the squares of the distances of points to the centroids and finds an inflection point on the graph where further increases in the number of clusters produce a negligible reduction in error. The Silhouette Score measures how well each point belongs to its cluster, and the Gap Statistic compares the intra-cluster variance to a random distribution of the data. Choosing the optimal number of clusters is critical because too few clusters can lead to overgeneralization, while too many clusters can lead to overfitting and unnecessary computational costs.

Control questions and tasks for topic 9

List of theoretical questions

1. What are the main differences between big data classification and clustering methods?
2. What are the main tasks of the big data classification method, and what are the clustering method?
3. What are the main advantages and limitations of big data classification methods compared to clustering?
4. What criteria are important when choosing a clustering method for a specific problem with a large amount of data?
5. Which big data classification algorithms are considered the most effective, especially in the context of large amounts of data?
6. How to use clustering results to improve big data management strategies?

7. How can the parameters of the K-means method be optimized to achieve optimal clustering of big data?

8. What solutions to the "curse of dimensionality" problem affect the results of big data classification and clustering methods?

9. What are the possible problems that arise when implementing the K-means algorithm for large amounts of data and how can they be overcome?

10. How to determine the optimal number of clusters for the K-means method in large data sets, and why is it important for clustering results?

List of tasks for independent work

Practical task 1. Classification of handwritten digits.

It is necessary to build a classification model that can recognize handwritten digits based on their pixel representation.

Dataset description: dataset *digits* (from the S cikit-learn library); number of samples: 1797; number of classes: 10 (numbers from 0 to 9); number of features: 64 (8x8 image pixels).

Load the dataset `digits` from `sklearn.datasets`. Split the data into features (`X`) and target values (`y`).

```
from sklearn.datasets import load_digits
# Loading the dataset
digits = load_digits()
# Features (input data)
X = digits.data
# Target values (class labels)
y = digits.target
```

Display some sample images from the dataset along with their actual labels to get an idea of what the data looks like:

```
import matplotlib.pyplot as plt
# Visualization of the first 5 images
fig, axes = plt.subplots(1, 5, figsize=(10, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
ax.set_axis_off()
ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
ax.set_title(f'Label: {label}')
plt.show()
```

Split the data into training and test sets in a ratio of 80:20:

```
from sklearn.model_selection import train_test_split
# Data separation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Choose a classifier for digit recognition. Try KNeighborsClassifier, SVC, or RandomForestClassifier to get started:

```
from sklearn.neighbors import KNeighborsClassifier
# Creating a K-Nearest Neighbors model
model = KNeighborsClassifier(n_neighbors=3)
```

Train the model on the training dataset:

```
# Model training
model.fit(X_train, y_train)
```

Evaluate the model on the test dataset, calculate the classification accuracy, and display the error matrix:

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Prediction on the test set
y_pred = model.predict(X_test)
# Calculation of accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
# Error matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
# Classification report
class_report = classification_report(y_test, y_pred)
print('Classification Report:')
print(class_report)
```

Try different classification algorithms or optimize model parameters to improve results:

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
# Using GridSearchCV to find the best SVM parameters
param_grid = {'C': [0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1]}
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
grid_search.fit(X_train, y_train)
# Optimal model estimation
best_model = grid_search.best_estimator_
best_accuracy = grid_search.best_score_
print(f'Best Model Accuracy: {best accuracy:.2f}')
```

Practical task 2. Clustering data on coffee consumption in different countries.

The goal of this task is to segment countries into groups based on their coffee consumption in order to understand different consumer categories and develop targeted marketing strategies for different markets. Data clustering is required to segment coffee based on its characteristics. Requirements for completing the practical task: Python 3.x, libraries: Pandas, Numpy, Matplotlib, Seaborn, Scikit-learn, R equests. Load the dataset from the URL using the Pandas library. This dataset is available via a direct link: https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/arabica_data_cleaned.csv

```
import pandas as pd
# URL to download the dataset
url = "https://raw.githubusercontent.com/jldbc/coffee-quality-database/master/data/arabica_data_cleaned.csv"
# Loading data
df = pd.read_csv(url)
# View the first few lines
print(df.head())
```

Check for missing values and select the columns to use for clustering. We select columns that contain numeric values that will be useful for analysis.

```
# Check for missing values
print(df.isnull().sum())
# Selecting numeric columns for clustering
data = df[['Aroma', 'Flavor', 'Aftertaste', 'Acidity', 'Body', 'Balance']]
data = data.dropna() # Remove rows with missing values
```

Create a scatter plot to visualize the relationships between the selected variables.

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10, 6))
sns.scatterplot(data['Aroma'], data['Flavor'])
plt.xlabel('Aroma')
plt.ylabel('Flavor')
plt.title('Distribution of Aroma vs Flavor')
plt.show()
```

Use `KMeans` from Scikit-learn to cluster the data. Use the "elbow" method to determine the optimal number of clusters.

```
from sklearn.cluster import KMeans
# The "elbow" method for determining the optimal number of clusters
distortions = []
for k in range(1, 11):
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(data)
distortions.append(kmeans.inertia_)
# Plotting the "elbow" graph
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Distortion')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.show()
# Running K-Means with optimal number of clusters
optimal_k = 4 # Assume that the optimal number of clusters is 4
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
df['Cluster'] = kmeans.fit_predict(data)
Display the clustering results using different colors for the clusters.
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Aroma', y='Flavor', hue='Cluster', data=df, palette='viridis', s=100)
plt.xlabel('Aroma')
plt.ylabel('Flavor')
plt.title('Clusters of Coffee Aroma and Flavor')
plt.show()
```

This code performs K-Means clustering on a dataset to group similar data points based on the features "Aroma" and "Flavor." It begins by using the "elbow method" to determine the optimal number of clusters: it fits K-Means models with 1 to 10 clusters and plots the distortions (inertia) to identify where the rate of decrease sharply changes – the “elbow” point. Based on this, the optimal number of clusters is assumed to be 4.

The code then fits a K-Means model with 4 clusters, assigns cluster labels to the dataset, and visualizes the clustering results using a scatter plot where each cluster is colored differently.

Analyze what the resulting clusters mean. What consumer groups can be identified, and what actions can be taken based on this information?

```
# Calculate the average values for each cluster
cluster_means = df.groupby('Cluster')[['Aroma', 'Flavor']].mean()
print(cluster_means)
```

This code calculates the average values of the "Aroma" and "Flavor" features for each cluster. It uses the `groupby` function on the "Cluster" column in the DataFrame `df` to group the data by cluster labels assigned earlier by the K-Means algorithm, then computes the mean for "Aroma" and "Flavor" within each group. The result, `cluster_means`, shows the centroid-like average characteristics of each cluster.

List of used and recommended literature

9.1. Classification algorithms: Definition and main models [Electronic resource]. – Access mode: https://datascientest.com/en/classification-algorithms-definition-and-main-models

9.2. Al-Manaseer H., Abualigah L., Alsoud A.R., Zitar R.A., Ezugwu A.E., Jia H. A Novel Big Data Classification Technique for Healthcare Application Using Support Vector Machine, Random Forest and J48. Classification Applications with Deep Learning and Machine Learning Technologies. Studies in Computational Intelligence, 2023, vol. 1071. Springer, Cham. https://doi.org/10.1007/978-3-031-17576-3_9

9.3. Awad F.H. et al. Big Data Clustering Techniques Challenges and Perspectives: Review. Informatica, 47, 2023. R. 203–218. https://doi.org/10.31449/inf.v47i6.4445

9.4. Fouad H. Awad and Murtadha M. Hamad. Improved k-mens clustering algorithm for big data based on distributed smartphone neural engine processor. Electronics, 11(6): 883, 2022. https://doi.org/10.3390/electronics11060883

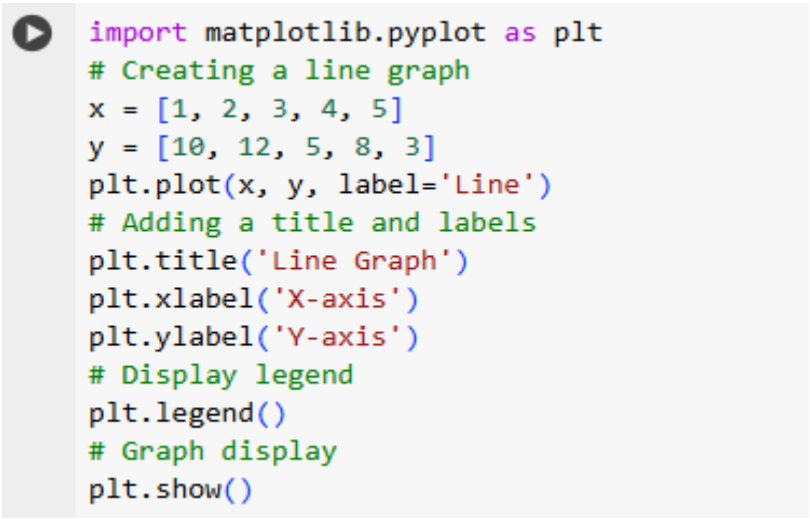
9.5. Clustering algorithms [Electronic resource]. – Access mode: https://developers.google.com/machine-learning/clustering/clustering-algorithms

Topic 10. Software methods and tools for data visualization

Python libraries and methods for data visualization allow to create various types of graphs and charts to make it easier to understand and analyze data. The Python language offers a wide range of libraries and methods for data visualization that allow to effectively analyze, explore, and present information in a clear way. The main capabilities of such libraries are the construction of various graphs (line, column, pie, histogram, heat map, 3D, etc.), customization of their design, interactivity, and integration with other tools. Matplotlib library provides basic functionality for creating static, animated, or interactive graphs. The Seaborn library adds high-level tools for working with categorical data, correlations, and heat maps. Plotly and Bokeh specialize in interactive visualizations, and the Pandas library has built-in methods for quickly constructing graphs based on DataFrames. Libraries for specialized tasks are also available, such as Geopandas for geographic data or NetworkX for graph visualization. These tools can be used to explore large data sets, analyze trends, compare categories, identify dependencies, and communicate research results in the form of informative graphical representations.

10.1. Python libraries and methods for data visualization

Matplotlib is one of the most popular Python libraries for creating static graphs and charts. It provides a wide range of capabilities for visualizing different types of data. Let's look at an example code:



```
import matplotlib.pyplot as plt
# Creating a line graph
x = [1, 2, 3, 4, 5]
y = [10, 12, 5, 8, 3]
plt.plot(x, y, label='Line')
# Adding a title and labels
plt.title('Line Graph')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
# Display legend
plt.legend()
# Graph display
plt.show()
```

This code uses the Matplotlib library to create and visualize a line plot. First, the lists `x are defined` and `y,` which represent the coordinates of the points on the plot. The `plt.plot()` function is used to draw a line connecting these points, with the `label='Line' parameter`, which adds a text label for the line in the plot legend. The plot title is set using `plt.title()`, and the labels for `the X` and `Y axes` are set using `plt.xlabel()` and `plt.ylabel()`, respectively. The `plt.legend()` `function` displays a legend on the plot that contains the line labels, and `plt.show() displays the plot in the visualization window.` This allows users to visually represent the relationship between the X and Y values in the form of a line graph (Fig. 10.1).

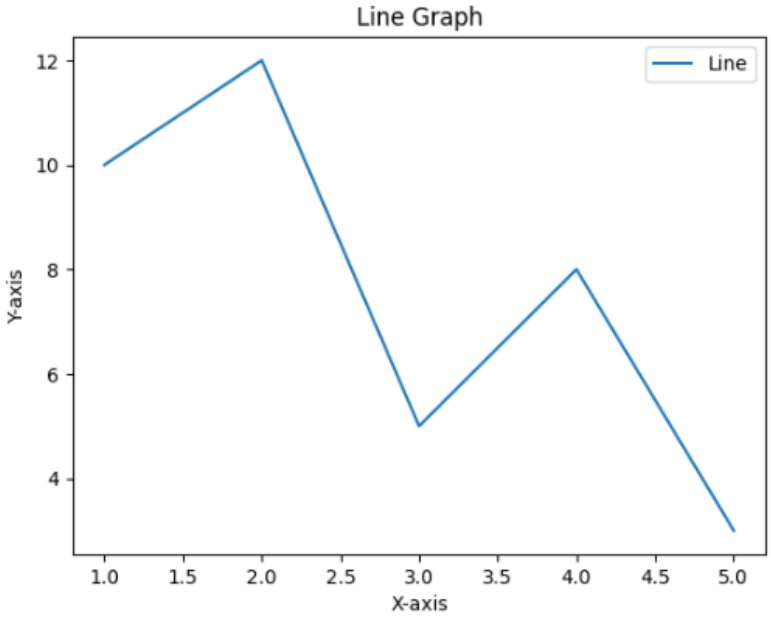


Fig. 10.1. Creating a line graph

Seaborn is a library built on top of Matplotlib that makes data visualization simpler and more harmonious, especially for statistical graphs.

Consider the code that uses the library Seaborn to visualize data from the built-in *tips* dataset, which contains information about restaurant bills and tips, which will help analyze the relationship between bill amount and tips, taking into account gender differences (Fig. 10.2).

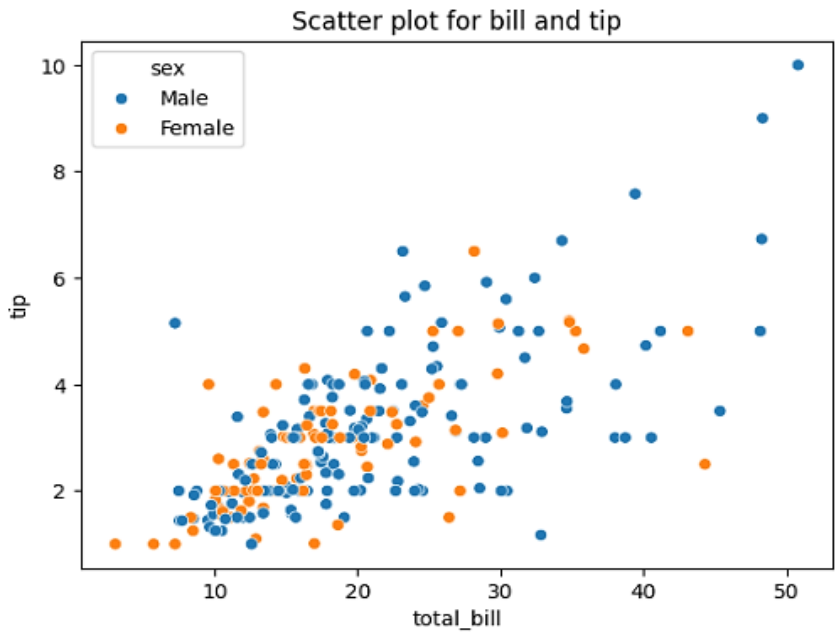


Fig. 10.2. Creating a scatter plot

```
import seaborn as sns
# Using the built-in dataset
tips = sns.load_dataset('tips')
# Creating a scatter plot
sns.scatterplot(x='total_bill', y='tip', data=tips, hue='sex')
# Adding a title
plt.title('Scatter plot for bill and tip')
# Graph display
plt.show()
```

First, the dataset is loaded using `sns.load_dataset('tips')`, then a scatter plot is created `with` axes `total_bill` and `tip` using `sns.scatterplot()`. The `hue='sex' parameter` adds color markers to the plot that represent the waiter's gender, allowing to visually distinguish points based on this criterion. The plot title is added using `plt.title()`, and then the plot is displayed on the screen using `plt.show()`.

Plotly is an interactive visualization library that provides high-quality graphs and the ability to interact with them in real time.

Consider a code example for exploring the diversity of iris species through a three-dimensional parameter space.

The code uses the `plotly.express library` to create an interactive 3D scatter plot using the built-in iris dataset (`iris`).

```
import plotly.express as px
# Using the built-in dataset
df = px.data.iris()
# Creating 3D graphics
fig = px.scatter_3d(df, x='sepal_width', y='sepal_length', z='petal_length', color='species'
# Adding a title
fig.update_layout(title='3D graph of Iris Dataset')
# Show graph
fig.show()
```

First, the dataset is loaded using `px.data.iris()`, which contains information about the length and width of sepals and petals of various iris species, `px.scatter_3d()` function creates a 3D plot where the `X axis` represents `sepal_width` (sepal width), the `Y axis` represents `sepal_length` (sepal length), and the `Z axis` represents `petal_length` (petal length). The `color='species' parameter` provides color coding of the points in the plot depending on the species of iris. Then, using the `fig.update_layout() method`, a plot title is added, and `fig.show()` displays the plot in the visualization window.

The constructed graph allows users to rotate the image and interact with the 3D data visualization (Fig. 10.3).

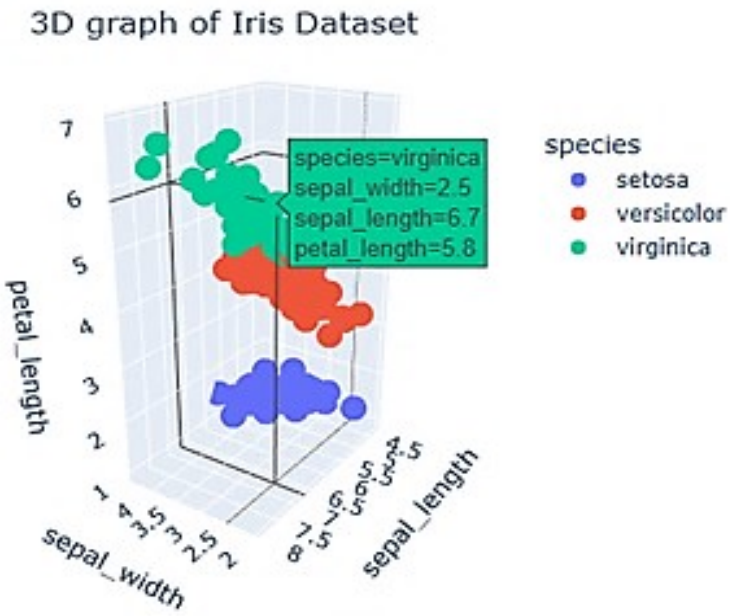


Fig. 10.3. 3D scatter plot using iris dataset

Altair is a declarative visualization library that allows users to create graphs with a declarative grammar. Code example:

```
import altair as alt
import pandas as pd
# Creating a DataFrame
data = pd.DataFrame({'x': range(10), 'y': [i**2 for i in range(10)]})
# Creating a line graph
chart = alt.Chart(data).mark_line().encode(x='x', y='y')
# Adding a title
chart.title = 'Squares of integers'
# Graph display
chart
```

This code creates a plot of integer squares using Altair from data stored in a pandas DataFrame. First, a DataFrame is created with two columns: 'x' containing values from 0 to 9, and 'y' containing the squares of those values. Then, Altair creates a plot with type `mark_line`, which defines the plot style as a line, and codes the `'x'` and `'y'` axes according to the columns of the DataFrame. The plot title is set to `'Integer Squares'` (Fig. 10.4) . Since this code is designed to run in a notebook, it will automatically display the graph when executed, without the need to call the `show()` method.

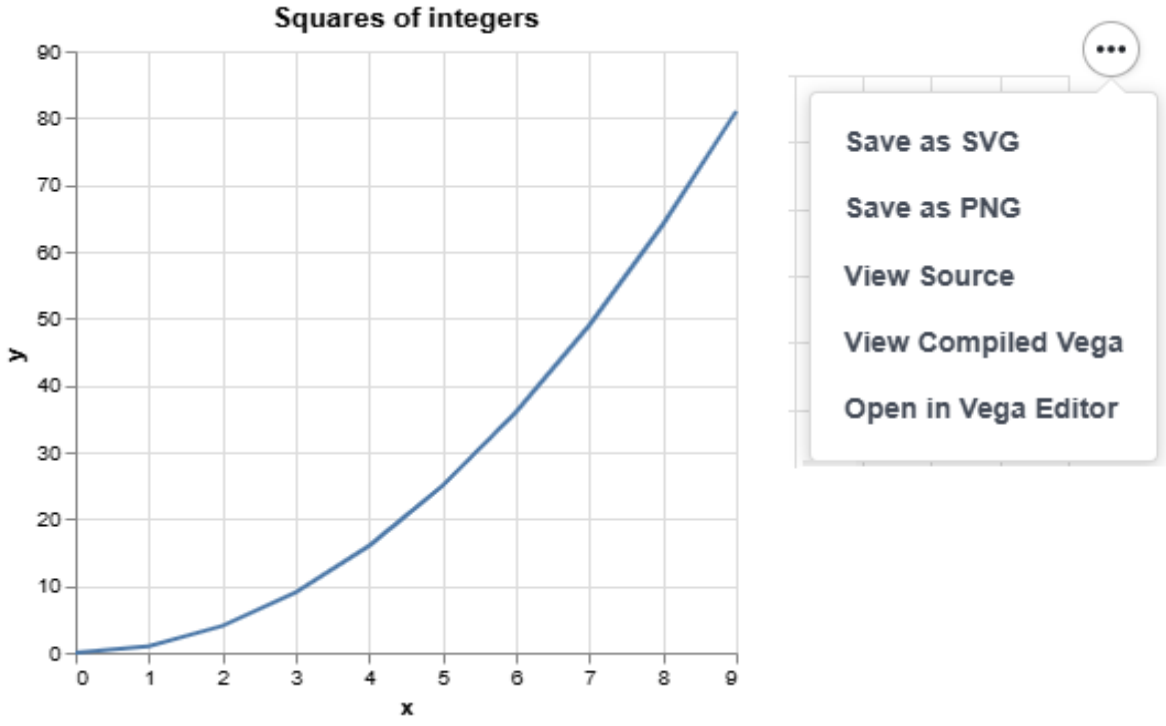


Fig. 10.4. Graph for the line of squares of integers

Bokeh is a library for creating interactive graphics and visualization for web applications. Let's look at a code example, which creates and displays a scatter plot using the Bokeh library (Fig. 10.5).

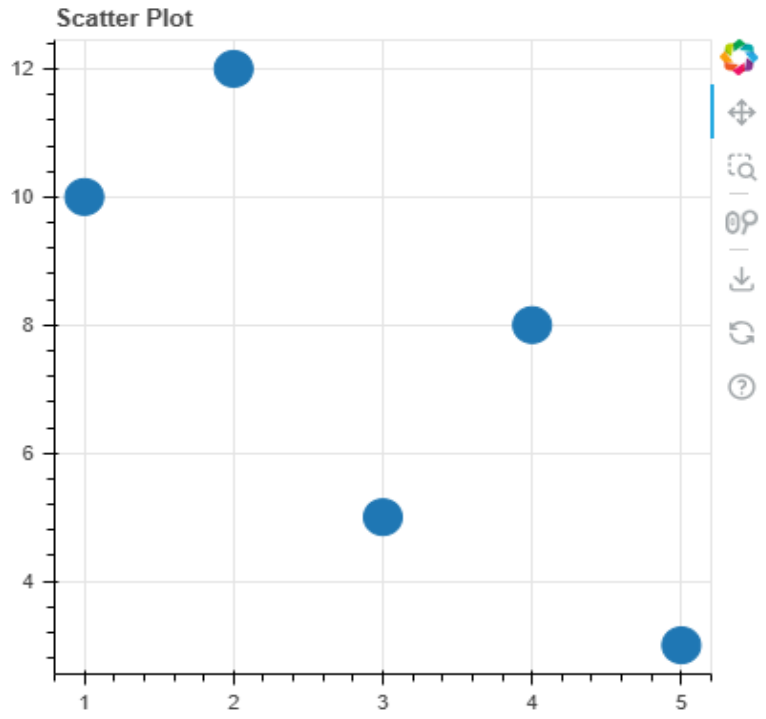


Fig. 10.5. Scatter plot

```
from bokeh.plotting import figure, show, output_notebook
from bokeh.models import ColumnDataSource
# Using output_notebook to display the graph directly in the notebook
output_notebook()
# Creating a scatter plot
p = figure(width=400, height=400)
source = ColumnDataSource(data=dict(x=[1, 2, 3, 4, 5], y=[10, 12, 5, 8, 3]))
p.scatter('x', 'y', size=20, source=source)
# Adding a title
p.title.text = 'Scatter Plot'
# Show graph
show(p)
```

First, the necessary modules are imported, and then the `output_notebook function is used` to display the graph directly in the Jupyter Notebook environment.

Next, a `figure object is created` with the specified dimensions, and a data source (`ColumnDataSource`) is defined with `x` and `y`

`coordinates`. The `scatter method` is used to add points to the plot with a size of 20, and the plot title `'Scatter Plot' is added`. The plot is displayed using the `show function`.

10.2. Features of the Plotly online tool

Plotly is a data visualization tool that has a number of features, including display and interactivity. Some of the key features of the online Plotly tool include interactive graphs, support for multiple programming languages, real-time charts, built-in graphs and charts, export and embedding, and cloud availability. These features make Plotly an effective tool for data visualization and analysis, especially when interactivity and accessibility in an online environment are important.

Plotly allows users to create interactive graphs that we can hover over, zoom in, and interact with using various controls. The Plotly tool allows users to create a variety of graph types, including line, pie, column, area, histogram, 3D graphs, and many others (Fig. 10.6).

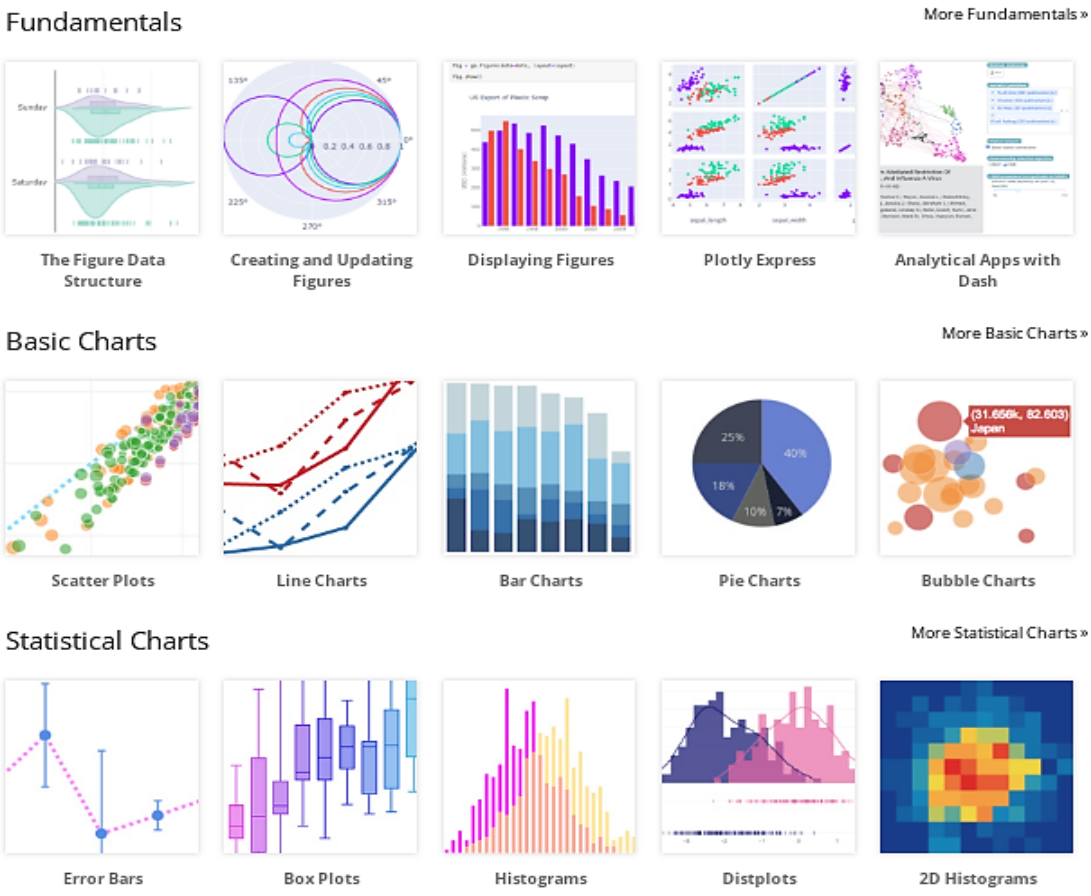


Fig. 10.6. Features of the online tool Plotly

Plotly supports not only Python, but also other programming languages such as R, Julia, and MATLAB. With Plotly, we can create real-time graphs that automatically update with new data.

Plotly has built-in graphs and charts that allow to visualize various types of data without the need for detailed programming, and it also allows users to export graphs to various formats such as PNG, JPEG, SVG, PDF. Graphs can be embedded into web pages and blogs.

The Plotly interface is available through the Plotly Cloud platform, which allows users to store and share plots online. Plotly is the foundation for the Dash framework, which allows users to create interactive web applications for data analysis. Dash is the best way to build analytical applications in Python using Plotly shapes.

Consider the code that creates a Dash-based web application to forecast restaurant revenue, allowing the user to choose between different machine learning models:

```
from dash import Dash, dcc, html, Input, Output
from sklearn.model_selection import train_test_split
from sklearn import linear_model, tree, neighbors
import plotly.graph_objects as go
import plotly.express as px
import numpy as np
app = Dash(__name__)
models = {'Regression': linear_model.LinearRegression,
'Decision Tree': tree.DecisionTreeRegressor,
'k-NN': neighbors.KNeighborsRegressor}
app.layout = html.Div([
html.H4("Predicting restaurant's revenue"),
html.P("Select model:"),
dcc.Dropdown(
id='dropdown',
options=["Regression", "Decision Tree", "k-NN"],
value='Decision Tree',
clearable=False
),
dcc.Graph(id="graph"),
])
```

The web interface consists of a title, a drop-down menu for selecting a model (linear regression, decision tree, k-NN), and a graph for displaying the results (Fig. 10.7).

```
@app.callback(
Output("graph", "figure"),
Input('dropdown', "value"))
def train_and_display(name):
df = px.data.tips() # replace with your own data source
X = df.total_bill.values[:, None]
X_train, X_test, y_train, y_test = train_test_split(
X, df.tip, random_state=42)
model = models[name]()
model.fit(X_train, y_train)
x_range = np.linspace(X.min(), X.max(), 100)
y_range = model.predict(x_range.reshape(-1, 1))
fig = go.Figure([
go.Scatter(x=X_train.squeeze(), y=y_train,
name='train', mode='markers'),
go.Scatter(x=X_test.squeeze(), y=y_test,
name='test', mode='markers'),
go.Scatter(x=x_range, y=y_range,
name='prediction') ])
return fig
app.run_server(debug=True)
```

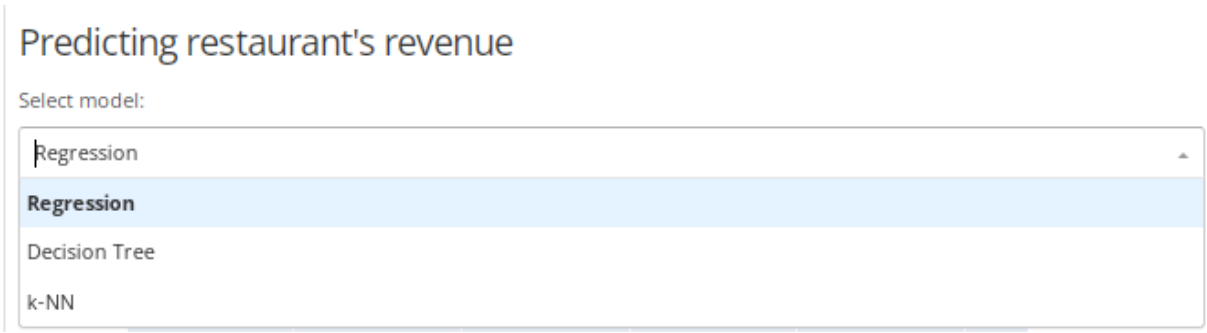


Fig. 10.7. An example of interactive machine learning algorithm selection for restaurant revenue forecasting

When a model is selected, a callback function is called that trains the selected model on the tip data from the Plotly library, performs predictions, and displays the results in the form of a graph showing the training data, test data, and predicted values (Fig. 10.8).

Fig. 10.8. Choice of different models machine learning for restaurant revenue forecasting

10.3. Tableau data visualization and business analytics software capabilities

Tableau is one of the most popular tools for data visualization and business analytics. Tableau allows users to analyze data, create reports, and interactive visualizations without the need for deep programming knowledge. Tableau is a powerful tool for building dashboards and analyzing large amounts of data in real time.

Tableau allows users to drag fields and create visualizations using an intuitive drag-and-drop interface. This allows users to quickly create graphs, charts, and maps without any special programming skills (Fig. 10.9).

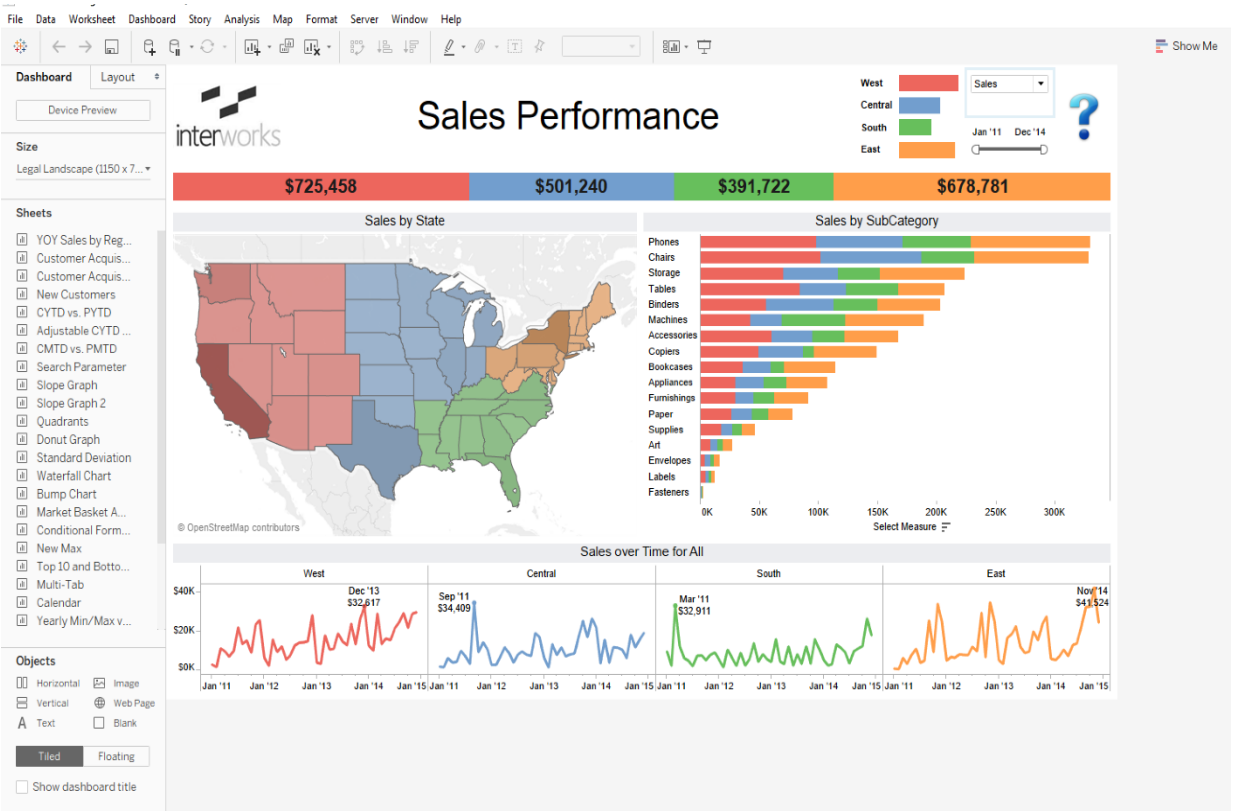


Fig. 10.9. Using Tableau software to analyze sales data for different product categories

Tableau supports connections to a variety of data sources, including Excel files, SQL databases, Google Analytics, Salesforce, Hadoop, cloud databases, APIs, and many others (Fig. 10.10).

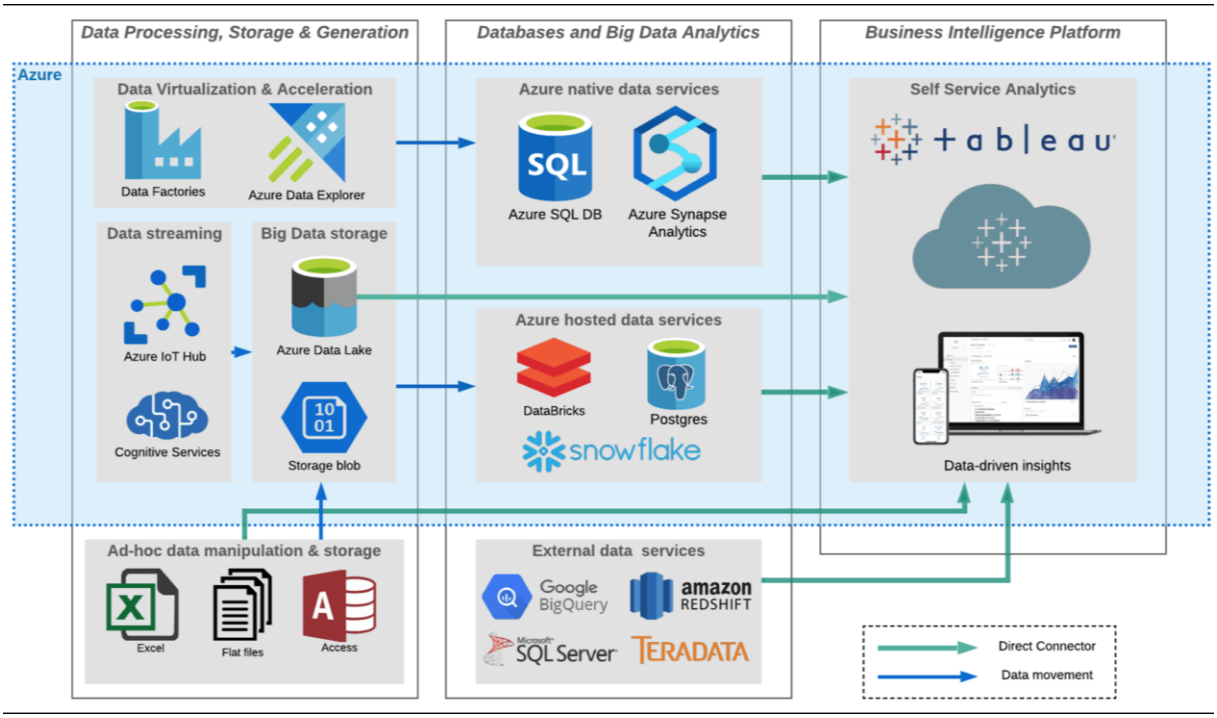


Fig. 10.10. Connecting and integrating Tableau with Microsoft Azure

The tool allows users to create various types of visualizations: bar charts, pie charts, line graphs, heat maps, geographic maps, and other specialized visualizations that allow to present data in a clear and visual format.

Tableau allows users to create interactive dashboards that allow users to filter, sort, and drill down into data.

The tool can work with large data sets, allowing to quickly perform complex calculations, aggregate, and filter data.

Tableau has collaboration functionality, sharing dashboards and reports, which makes it easy to collaborate within teams. Using Tableau Server or Tableau Online, users can share their visualizations with other team members or with clients.

Tableau Desktop is a basic tool for creating visualizations and dashboards. Users can connect to various data sources and build reports.

Tableau Server is a platform for publishing reports and dashboards to other users in an organization to ensure collaboration and data security.

Tableau Online is a cloud-based version of Tableau Server that allows users to publish and share visualizations over the Internet without the need to deploy own infrastructure.

Tableau Public is a free version of Tableau that allows users to create and publish visualizations publicly. Suitable for students or those who do not need data privacy.

Tableau Prep is a tool for preparing and cleaning data before visualizing it in Tableau Desktop.

Tableau can be used to analyze sales across regions. For example, we can create an interactive geographic map that shows sales volumes in different cities. A company can also analyze sales dynamics across product categories, identify top customers, and forecast sales based on historical data.

E-commerce companies can use Tableau to analyze Google Analytics data to create dashboards that display data about website visitor behavior, traffic sources, conversions, and ad campaign performance. For example, a company might create a Tableau dashboard to visualize metrics such as the number of daily visitors, the most popular landing pages, the bounce rate by traffic source, and the ROI of individual ad campaigns, enabling them to identify trends and optimize marketing strategies in real time.

Tableau can be used by financial analysts to analyze financial statements and make forecasts. For example, we can create charts that show changes in a company's profitability, operating expenses, and compare financial metrics to competitors.

Marketing teams can use Tableau to analyze social media performance. This tool allows users to visualize the number of interactions, likes, comments, and views across different platforms, such as Facebook, Twitter, or Instagram, and compare their performance.

Hospitals and healthcare facilities can use Tableau to analyze patient data, monitor treatment effectiveness, and improve healthcare services. For example, create a dashboard with information on average patient wait times, the number of cases processed, and analyze the effectiveness of medical procedures.

For example, we could create a chart in Tableau that shows sales for four quarters across three regions: North, South, and West. Using a bar chart, we could display sales for each quarter for each region, allowing team to see where to focus their marketing efforts.

Tableau allows explore this data interactively – for example, we can click on the bars of the graph to get additional information about each region or quarter.

10.4. Power BI as a business analytics automation tool

Power BI is Microsoft's integrated business intelligence platform that provides tools for collecting, visualizing, analyzing, and navigating data. Power BI plays a key role in implementing automated business intelligence processes. Consider the main characteristics and capabilities of Power BI as a business analytics automation tool. Power BI allows users to connect and import data from a variety of sources, including databases, cloud services, file systems, and other data sources (Fig. 10.11).

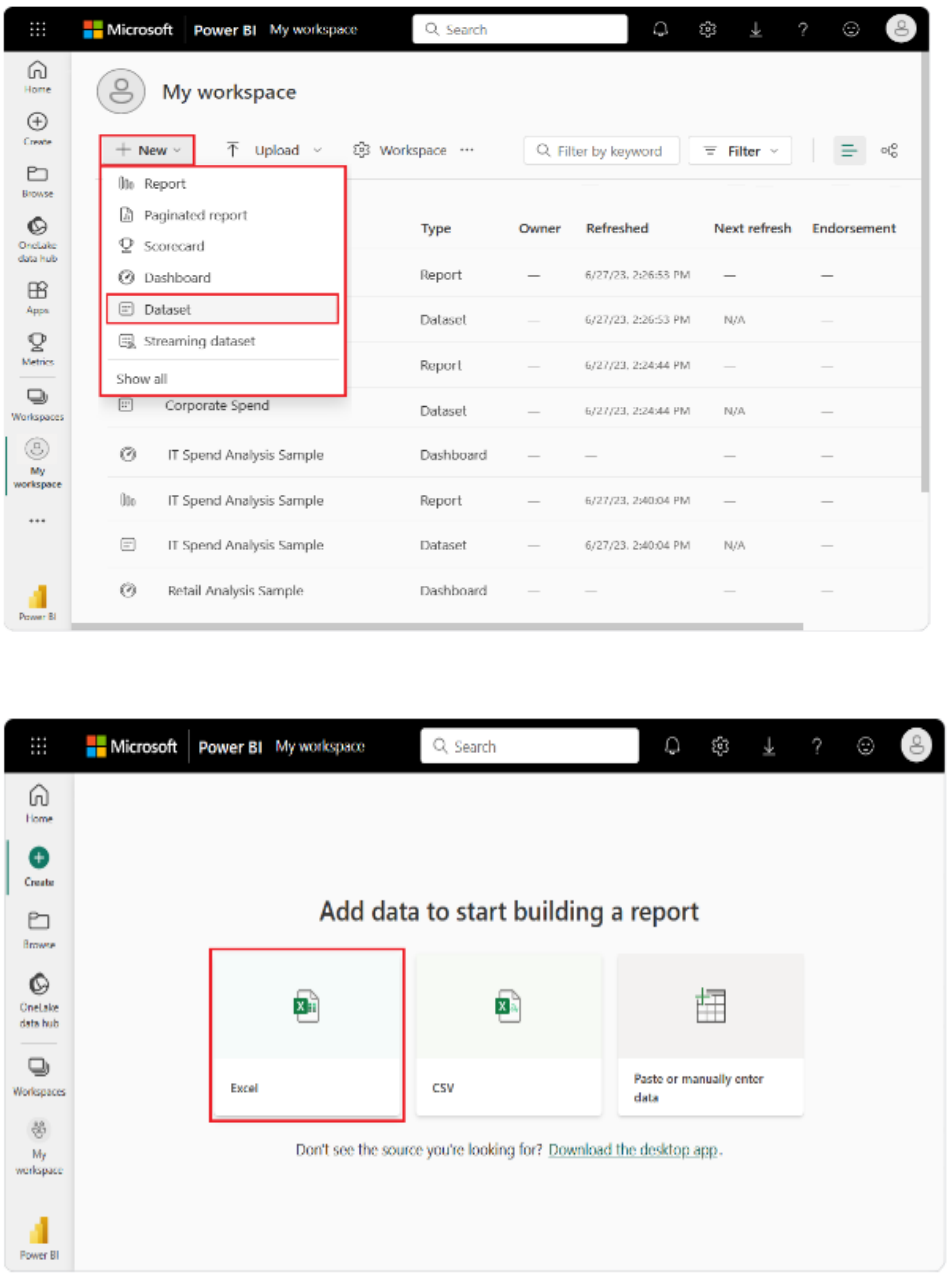


Fig. 10.11. Getting started with Power BI

Power BI visualization tools provide the ability to create a variety of graphs, charts, and other visual elements to effectively display data (Fig. 10.12). Users can interact with data using various filters and sifting, making analysis more flexible and dynamic.

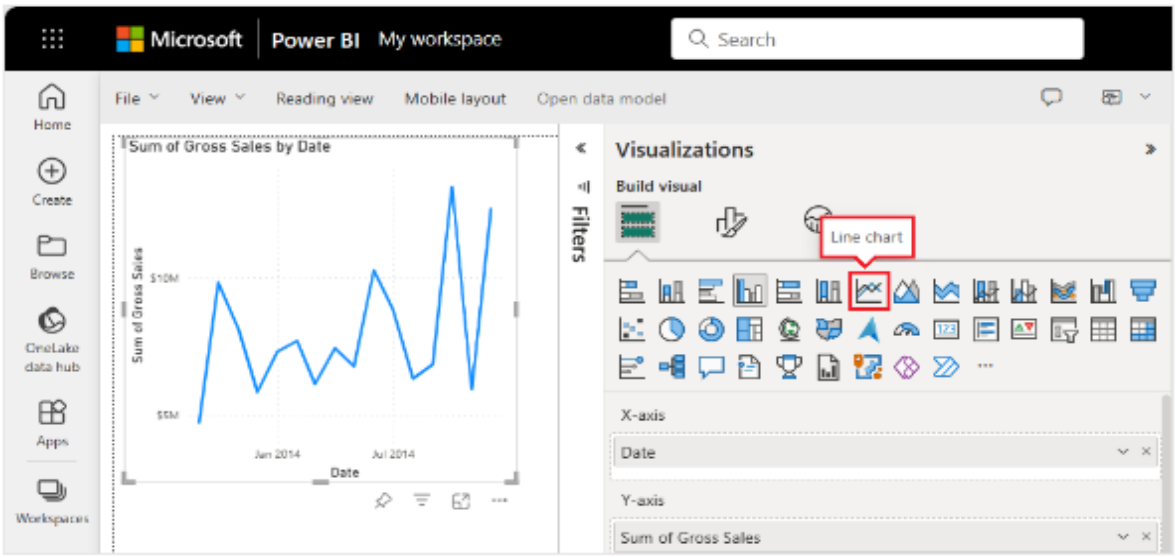


Fig. 10.12. Data visualization capabilities in Power BI

Power BI allows users to schedule automatic data updates, ensuring the relevance and accuracy of analytical information. The ability to collaborate on reports and dashboards in real time promotes teamwork and the sharing of analytical information. Power BI includes various tools for performing calculations, data modeling, and statistical analysis.

Integration with other Microsoft products, such as Excel, SharePoint, Azure, Dynamics 365, and Office 365, makes Power BI an important part of the Microsoft ecosystem for business intelligence. The Power BI service imports sample data from an Excel file as a semantic model and opens the Financial Sample page (Fig. 10.13, 10.14).

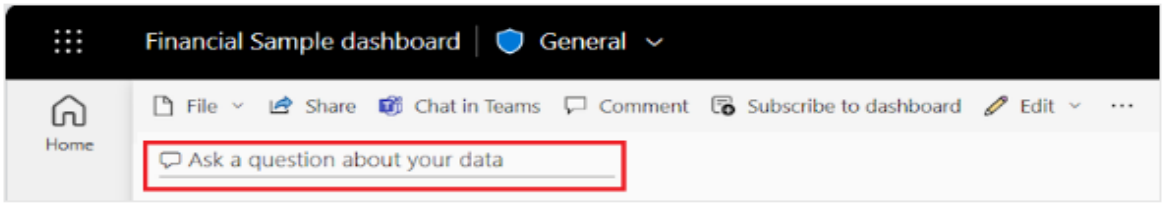


Fig. 10.13. Financial Sample page

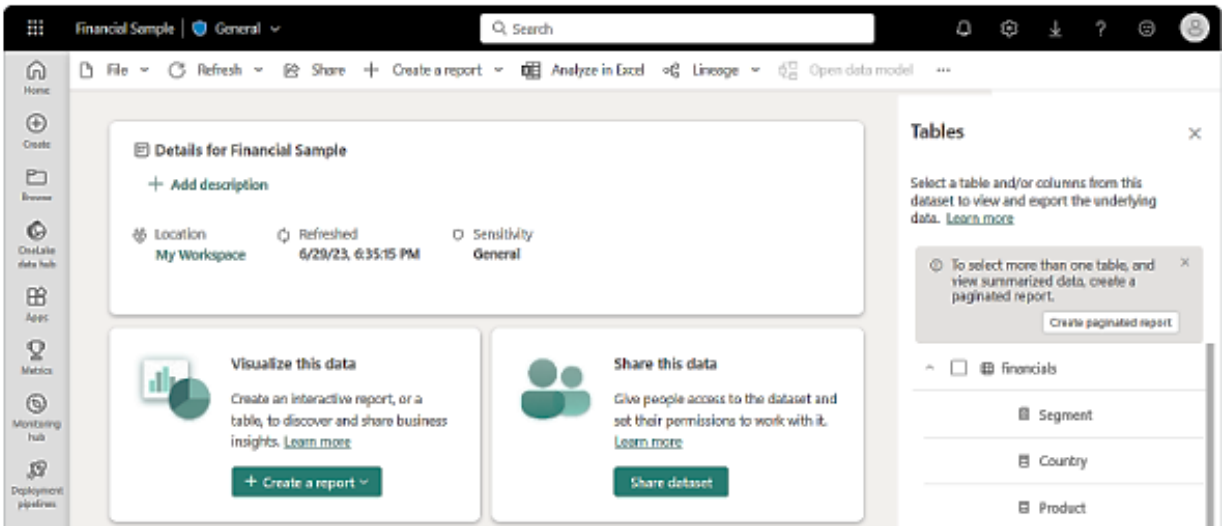


Fig. 10.14. Financial Sample features

Power BI allows users to create advanced reports and dashboards, which helps make informed business decisions based on data. Power BI helps automate analytics processes by providing users with a powerful tool for visualizing and understanding data in real time, interactively creating reports and dashboards that can reflect the real state of the business using graphs, tables, and maps. For example, with Power BI, a company can analyze sales by region, track the effectiveness of marketing campaigns, or control costs.

Power BI supports big data and complex calculations, allowing users to create predictive models, analyze trends, and determine key performance indicators (KPIs) used to assess the success of an organization or its individual processes. Thanks to collaboration features, reports can be shared with colleagues, set up automatic data refreshes, and receive notifications about important changes. Built-in artificial intelligence capabilities allow to use machine learning to automatically detect anomalies, cluster data, and conduct sentiment analysis. For example, a trading company can use Power BI to analyze buyer behavior, optimize inventory, and forecast demand, which helps improve business process efficiency and increase profits.

10.5. Visualizing big data on maps

Python language offers a set of libraries for working with geospatial data and creating cartographic visualizations.

Consider the main Python libraries used for visualizing big data on maps.

Matplotlib and Seaborn are libraries for basic visualization that can be combined with cartographic data.

GeoPandas is an extension of the Pandas library for working with geospatial data, which allows users to manipulate vector geodata. GeoPandas allows users to load geospatial data (for example, in Shapefile or GeoJSON format), process it, and create static maps. For example, we can display the location of stores in a city using coordinates from a CSV file. After loading the data into GeoPandas, we can use Matplotlib to overlay the data on a map of the area or administrative boundaries.

Folium is a library for creating interactive web maps based on the Leaflet.js library. Folium provides the ability to create interactive maps for web applications. For example, we can visualize the flow of customers between different regions or determine the busiest roads in a city. This allows users to interactively change the scale, view pop-ups with additional information, and create layers with different data types. Fig. 10.15 shows an example of using feature markers by their geographical coordinates.

Fig. 10.15. An example of using object markers on a map in Folium by their geographical coordinates

Pydeck is a Python library that provides an interface to Deck.gl, a JavaScript library for visualizing large datasets as interactive maps. Pydeck supports multi-layer maps, 3D visualizations, and works efficiently with web browsers using WebGL. Pydeck specializes in creating interactive maps with the ability to visualize 3D layers and animations.

Let's look at a few examples of using the Pydeck library for visualization of geospatial data.

Example 1. Visualization of data point.

Consider the code that creates an interactive layer for visualizing data on a map using the Pydeck library. First, the Pydeck and Pandas libraries are imported. Next, a dataframe `data is created` that contains the geographic coordinates (latitude and longitude) of three points along with the corresponding values used for color visualization.

After that, a `scatter_layer layer` of type `ScatterplotLayer is defined`, which is responsible for plotting the points on the map. This layer uses data about the position of the points (`get_position='[longitude, latitude]'`), sets the radius of each point to 10,000 units, colors the points based on their value (`get_color='[value, 140, 200]'`), and also makes the points interactive via the `pickable=True parameter`. This layer can be added to the map for further visualization:

```
!pip install pydeck pandas
```

```
Collecting pydeck
  Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: jinja2>=2.10.1 in /usr/local/lib/python3.11/dist-packages (from pydeck) (3.1.5)
Requirement already satisfied: numpy>=1.16.4 in /usr/local/lib/python3.11/dist-packages (from pydeck) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2024.2)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2>=2.10.1->pydeck) (3.0.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Downloading pydeck-0.9.1-py2.py3-none-any.whl (6.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 31.6 MB/s eta 0:00:00
Installing collected packages: pydeck
Successfully installed pydeck-0.9.1
```

```
import pydeck as pdk
import pandas as pd
# Point data
data = pd.DataFrame({
'latitude': [37.7749, 34.0522, 40.7128],
'longitude': [-122.4194, -118.2437, -74.006],
'value': [100, 200, 300]
})
# Layer settings
scatter_layer = pdk.Layer(
'ScatterplotLayer',
data,
get_position='[longitude, latitude]',
get_radius=10000,
get_color='[value, 140, 200]',
pickable=True
)
```

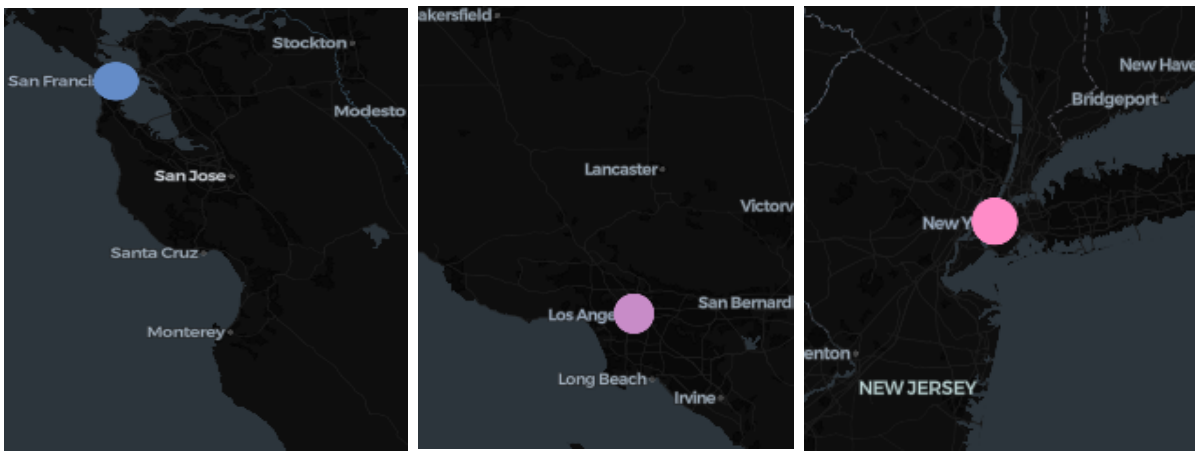


Fig. 10.16. Visualizing point data on a map using Pydeck

Figure 10.16 shows the points on the map of the United States that correspond to the cities of San Francisco, Los Angeles, and New York. The visualization is built based on the geographic coordinates (latitude and longitude) and the values that were associated with each city in the table:

San Francisco: value 100;

Los Angeles: value 200;

New York: value 300.

The points have a radius of 10,000 units and their color is based on the value (only the `value parameter has an effect`). If we need a caption or additional text for the points, we can use special layers, such as TextLayer, to add information to the map.

Example 2. Building a heat map.

Consider code that creates a heat map based on given geographic coordinates (latitude and longitude) and weight (value).

```
# Heatmap layer
heatmap_layer = pdk.Layer(
'HeatmapLayer',
data,
get_position='[longitude, latitude]',
get_weight='value',
radius_pixels=50
)
# Creating a heat map
pdk.Deck(layers=[heatmap_layer], initial_view_state=view_state).show()
```

First, a heatmap layer is defined using `HeatmapLayer`, where `get_position` specifies the position of the points on the map, and `get_weight` specifies the influence of each point on the intensity of the thermal effect. The parameter `radius_pixels=50` specifies the radius of influence of each point in pixels.

Then, using the `pdk.Deck object`, a map is created that visualizes the thermal intensity depending on the specified data, and is displayed in the window (Fig. 10.17).

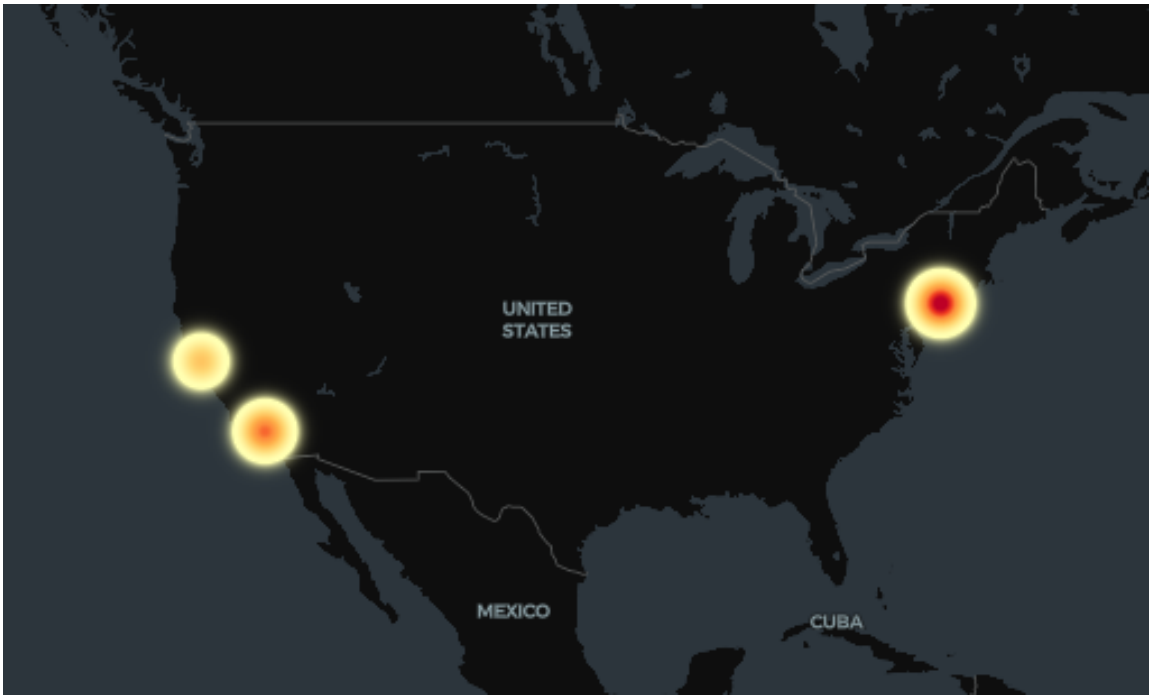


Fig. 10.17. Building a heat map using Pydeck

Example 3. Route visualization.

Consider the code that creates a visualization of a route on a map. The route data is represented as `a DataFrame`, where the `path column` contains a list of coordinates (latitude and longitude) that describe the route, and `color` specifies the color of the route in RGB format (here, red [255, 0, 0]). The `PathLayer layer` is responsible for visualizing the route line, where `get_path` specifies the route, `get_color` specifies the color of the line, `width_scale` scales the width of the route, and `width_min_pixels` sets the minimum width in pixels.

```
# Route data
routes = pd.DataFrame({
'path': [
[[-122.4194, 37.7749], [-118.2437, 34.0522], [-74.006, 40.7128]]
],
'color': [[255, 0, 0]]
})
# Routes layer
path_layer = pdk.Layer(
'PathLayer',
routes,
get_path='path',
get_color='color',
width_scale=20,
width_min_pixels=5
)
# Map with routes
pdk.Deck(layers=[path_layer], initial_view_state=view_state).show()
```

A map with a route is created via `pdk.Deck` , and the route is displayed in the visualization window (Fig. 10.18).

Fig. 10.18. Route visualization using Pydeck

Example 4. Column visualization.

Consider the code that creates a 3D visualization as columns representing data with a height proportional to `the value`. First, a new column `elevation is added`, which defines the height of the columns by multiplying the `value` by 10. Then, a `ColumnLayer is created`, where `get_position` sets the position of each column in latitude and longitude,

`get_elevation` sets the height of the columns, and `elevation_scale=50` scales it. The radius of each column is specified by `radius=5000`, and its color is specified by the value `[value, 100, 200]`.

```
# Data for 3D visualization
data['elevation'] = data['value'] * 10
# Layer of 3D columns
column_layer = pdk.Layer(
'ColumnLayer',
data,
get_position='[longitude, latitude]',
get_elevation='elevation',
elevation_scale=50,
radius=5000,
get_fill_color='[value, 100, 200]',
pickable=True
)
# Map with columns
pdk.Deck(layers=[column_layer], initial_view_state=view_state).show()
```

The map is created using `pdk.Deck`, where a column layer is added to the map, the visualization is displayed in a window (Fig. 10.19).

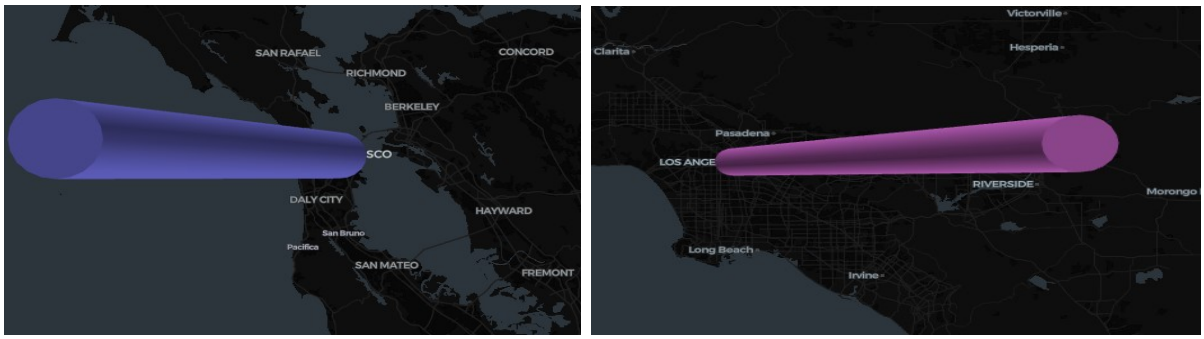


Fig. 10.19. Visualizing 3D columns using Pydeck

The considered software tools support the processing of both vector data (points, lines, polygons) and raster data (images, satellite images).

When working with large amounts of geospatial data, it is important to consider the following aspects.

1. *Data preprocessing* – we need to use libraries like Dask or PySpark to process large data sets before visualizing them.

2. *Data reduction* uses aggregation, sampling, or data transformation to reduce its volume while preserving key characteristics.

3. *Rendering optimization* uses tools that support WebGL (such as Pydeck or Plotly) to quickly display large datasets in the browser.

Optimization of rendering of large datasets on maps is achieved by using WebGL technology, which allows the browser to process graphics directly using the GPU, providing high speed and performance.

The Pydeck library supports WebGL and allows users to create interactive visualizations, such as heatmaps or 3D bars, even for millions of data points. An example is the use of Pydeck to visualize geodata about crime scenes in a city: the data is standardized, and a ScatterplotLayer or HeatmapLayer is used for each point on the map. Pydeck automatically optimizes rendering by aggregating data and efficiently using hardware resources, ensuring fast response even for large-scale data.

Control questions and tasks for topic 10

List of theoretical questions

1. What are the main Python libraries for data visualization that you know and what is their purpose?
2. How to use Matplotlib to build different types of graphs in the Python programming language?
3. How does Seaborn complement Matplotlib in creating statistically oriented graphs?
4. What are the capabilities of Plotly for creating interactive and dynamic graphs and how can it be used in the Python programming language?
5. What are the benefits of using the online data visualization tool Plotly and what are the main features of this tool?
6. What types of charts can you create with Power BI and how does it integrate with other Microsoft products?
7. How does Power BI help automate business intelligence processes and what capabilities does it provide for scheduling data updates?
8. What data analysis tools are included with Power BI and how do they make data analysis and modeling easier?
9. What Power BI capabilities for collaboration and sharing analytical information in real time do you know?
10. How can we evaluate the effectiveness of using these tools in a

business environment and how do they affect decision-making processes?

11. What are the main Python libraries used for visualizing big data on maps, and what are their advantages?

12. How to choose the type of visualization (heatmap, bars, points, routes) depending on the type of data and analysis goals?

13. What techniques can be used to optimize performance when displaying large amounts of geodata on interactive maps?

14. How to ensure accuracy and informativeness of visualization when using different map zoom levels?

15. What approaches are used to integrate real-time data into maps (e.g., transportation tracking or environmental monitoring)?

List of tasks for independent work

Practical task 1. Analyzing and visualizing a dataset using Matplotlib and Pandas.

Task goal: learn how to load a dataset, process it, and visualize key trends using Matplotlib and Pandas. Install the required libraries:

```
!pip install matplotlib pandas
```

Import the necessary modules:

```
import pandas as pd
import matplotlib.pyplot as plt
```

Load a publicly available dataset (e.g., the iris dataset from UCI Machine Learning Repository):

```
df =
pd.read_csv('https://archive.ics.uci.edu/ml/machine-
learning-databases/iris/iris.data',
     header=None,
     names=['sepal_length', 'sepal_width',
'petal_length', 'petal_width', 'species'])
```

Display basic information about the dataset:

```
print(df.head())
print(df.describe())
print(df['species'].value_counts())
```

Create scatter plots to analyze relationships between variables.

Plot `sepal_length` vs `sepal_width` for different species:

```
species_list = df['species'].unique()
```

```
    for species in species_list:
        subset = df[df['species'] == species]
        plt.scatter(subset['sepal_length'],
subset['sepal_width'], label=species)
    plt.title('Sepal Length vs Sepal Width')
    plt.xlabel('Sepal Length (cm)')
    plt.ylabel('Sepal Width (cm)')
    plt.legend()
    plt.grid(True)
    plt.show()
```

Add histograms for petal_length to analyze distribution:

```
    df['petal_length'].hist(bins=20, color='skyblue',
edgecolor='black')
    plt.title('Distribution of Petal Length')
    plt.xlabel('Petal Length (cm)')
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

Practical task 2. Visualizing image classification data using Plotly and PIL.

Task goal: use Plotly for visualizing data and predictions related to an image classification task. Install the required libraries:

```
    !pip install plotly pandas pillow
```

Import necessary modules:

```
    import plotly.express as px
    from PIL import Image
    import pandas as pd
    import numpy as np
```

Create a dataset with sample image information:

```
    data = {
        'Image_Name': ['cat.jpg', 'dog.jpg',
'flower.jpg', 'car.jpg'],
        'Category': ['Animal', 'Animal', 'Plant',
'Vehicle'],
        'Confidence_Score': [0.95, 0.88, 0.97, 0.92]
    }
```

```
df = pd.DataFrame(data)
Load a sample image for display:
img_path = 'https://via.placeholder.com/150'
img = Image.open(img_path)
img.show()
```

Visualize data using Plotly. Create a bar plot showing confidence scores for each category:

```
fig = px.bar(df, x='Image_Name',
y='Confidence_Score', color='Category',
           title='Confidence Scores by Image Category')
fig.show()
```

Simulate classification predictions and display:

```
predictions = {'Actual': ['Animal', 'Animal',
'Plant', 'Vehicle'],
                   'Predicted': ['Animal', 'Animal',
'Plant', 'Animal']}
predictions_df = pd.DataFrame(predictions)
print(predictions_df)
```

Highlight misclassified cases:

```
predictions_df['Correct'] =
predictions_df['Actual'] == predictions_df['Predicted']
misclassified =
predictions_df[~predictions_df['Correct']]
print('Misclassified Cases:')
print(misclassified)
```

Practical task 3. Data analysis using Power BI.

Download Power BI Desktop from the official Microsoft website and install it on your computer (Fig. 10.20). Import data. Open Power BI Desktop, click the "Get Data" button on the main panel. Select the data source type (e.g. Excel, CSV, database, etc.) and upload the data file. Go to the "Data" tab to view and edit the uploaded data. Clean up the data by deleting unnecessary columns or rows. Go to the "Report" tab and create a report. Select a visualization type (for example, column chart, line chart, table, etc.) and drag it onto the canvas. Drag the fields you want from the data table into the appropriate visualization areas (e.g., X and Y axes, legend, values, etc.). Add

interactive elements: filters, slicers, or other interactive elements so that users can interact with the report.

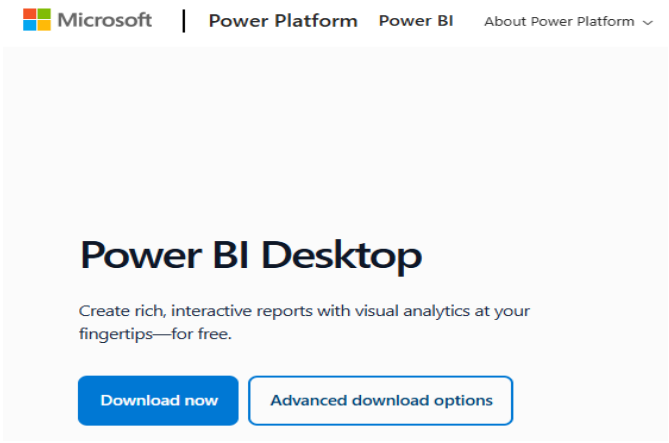


Fig. 10.20. Power BI Desktop (Download page)

Customize the interaction between visualizations so that they update when different options are selected. Save the created report. Export the report or publish it to Power BI Service to share with colleagues. Import a sales dataset (use a ready-made Excel or CSV file). Create a column chart showing the sales amount by region. Add a pie chart showing the sales distribution by product. Add filters to select a time period and regions.

List of used and recommended literature

10.1. Matplotlib [Electronic resource]. – Access mode: https://matplotlib.org/

10.2. Seaborn [Electronic resource]. – Access mode: https://seaborn.pydata.org/

10.3. Plotly [Electronic resource]. – Access mode: https://plotly.com/

10.4. Altair [Electronic resource]. – Access mode: https://altair.com/

10.5. Bokeh [Electronic resource]. – Access mode: https://bokeh.org/

10.6. Tableau [Electronic resource]. – Access mode: https://www.tableau.com/

10.7. Power BI [Electronic resource]. – Access mode: https://www.microsoft.com/en-us/power-platform/products/power-bi

10.8. Power BI Desktop [Electronic resource]. – Access mode: https://www.microsoft.com/en-us/power-platform/products/power-bi/desktop

Topic 11. Integration of artificial intelligence technologies to build distributed big data analytics systems

The integration of artificial intelligence (AI) technologies for building distributed big data analytics systems is a key direction in the development of modern information technologies. The use of AI in big data analytics allows users to automate the processing of large volumes of data, increasing the accuracy and speed of analysis. In particular, machine and deep learning technologies are used to detect patterns, predict and optimize processes in distributed systems. Tools such as Hadoop, Spark and TensorFlow provide scalability and efficiency of data processing on a large scale. The implementation of AI requires addressing issues of security, data privacy and ensuring the reliability of systems, which requires a comprehensive approach and constant improvement of technologies.

11.1. Integration of artificial intelligence technologies in distributed systems

AI technologies can be implemented directly into individual nodes of a distributed system. For example, using machine learning algorithms to analyze data at the local level can include image classification or real-time text analysis. In the case of "smart" sensors in the IoT system, such algorithms allow to detect anomalies in the collected data directly on the device, which reduces the amount of data transmitted to the central server.

As an example, a neural network deployed on a surveillance camera can identify suspicious movements and automatically send an alarm signal, processing the video stream locally. This not only increases the efficiency of the system, but also provides greater confidentiality and savings in computing resources across the entire network.

Distributed systems often deal with a large number of events. AI technologies can be integrated to process and analyze these events to obtain more detailed information. It is also possible to use remote AI services that run in the cloud or on remote servers and interact with the distributed system over the network.

Distributed machine learning models can be built, trained, and used to make decisions across multiple nodes in a distributed system. AI can be

integrated into distributed databases to enable more efficient analysis and use of data at multiple levels.

Systems that use intelligent agents can interact with a distributed system to solve certain tasks or optimize processes.

AI technologies can be integrated into distributed systems through APIs and microservices to provide flexibility and scalability. Natural language processing technologies can be used to analyze and understand text information that is processed by the distributed system.

Consider real-world examples of integrating AI technologies in distributed systems.

Example 1. Building a news classification system using TF-IDF and logistic regression.

Consider an example of building a news classification system using TF-IDF and logistic regression where we build a system to classify news articles (e.g., into categories like "sports," "technology," "politics") using PySpark. Instead of recommendations, we will determine which category a given news article belongs to.

```
    # Install the PySpark library
    !pip install pyspark
    from pyspark.sql import SparkSession
    from pyspark.ml.feature import Tokenizer,
HashingTF, IDF
    from pyspark.ml.classification import
LogisticRegression
    from pyspark.ml import Pipeline
    # Initialize Spark session
    spark =
SparkSession.builder.appName("NewsClassification").getO
rCreate()
    # Example data
    data = [
        (0, "Football players scored a goal",
"sports"),
        (1, "New AI model surpasses humans in chess",
"technology"),
        (2, "Government passes a new law", "politics"),
```

```
        (3, "Olympic games to start next month",
"sports"),
        (4, "Technology companies announce new
gadgets", "technology"),
        (5, "President gives a speech on economy",
"politics")
    ]
    columns = ["id", "text", "label"]
    df = spark.createDataFrame(data, columns)
    # Tokenize text (split text into words)
    tokenizer = Tokenizer(inputCol="text",
outputCol="words")
    # Compute TF (Term Frequency)
    hashingTF = HashingTF(inputCol="words",
outputCol="rawFeatures", numFeatures=20)
    # Compute IDF (Inverse Document Frequency)
    idf = IDF(inputCol="rawFeatures",
outputCol="features")
    # Logistic Regression for classification
    lr = LogisticRegression(featuresCol="features",
labelCol="labelIndex", maxIter=10)
    # Build a pipeline
    pipeline = Pipeline(stages=[tokenizer, hashingTF,
idf, lr])
    # Prepare data (convert text labels into numerical
indices)
    from pyspark.ml.feature import StringIndexer
    labelIndexer = StringIndexer(inputCol="label",
outputCol="labelIndex")
    df = labelIndexer.fit(df).transform(df)
    # Split data into training and test sets
    train, test = df.randomSplit([0.8, 0.2])
    # Train the model
    model = pipeline.fit(train)
    # Test the model
    predictions = model.transform(test)
```

```
    # Display results
    predictions.select("text", "label",
"prediction").show()
    # Terminate Spark session
    spark.stop()
```

We create a dataset containing news articles and their respective categories (labels). Each row contains an ID, the text of the news article, and the label (e.g., "sports," "technology," "politics").

Using the tokenizer, the text is split into individual words. For example, the text "Football players scored a goal" is converted into ["Football", "players", "scored", "a", "goal"].

Counts how often each word appears in the text.

Weighs the importance of each word by reducing the weight of commonly used words (e.g., "the," "and").

Together, TF-IDF creates numerical vectors for each piece of text.

Logistic regression is used to classify the text based on the TF-IDF vectors. The model is trained on the training data to learn how to categorize the news articles. The model's accuracy is tested on unseen data, and the classification results are displayed.

This PySpark-based model allows us to classify news articles into categories using the TF-IDF approach for text processing. Similar to the recommendation example, Spark provides distributed data processing, enabling the system to scale efficiently for large datasets.

The code begins by installing and importing the required libraries (`pyspark`) and initializing a Spark session with the application name "NewsClassification." Spark is used for large-scale data processing, and the session is the entry point for interacting with Spark functionalities.

A small dataset is created in the form of a list of tuples, where each tuple contains:

`id` is unique identifier for each sample;

`text` is short sentence describing the content;

`label` is The category of the text (e.g., "sports," "technology," or "politics").

This dataset is converted into a Spark DataFrame with specified column names.

The `Tokenizer` splits each text into individual words (tokens). For example, "Football players scored a goal" becomes `["Football", "players", "scored", "a", "goal"]`.

`HashingTF` calculates the term frequency for each token and represents it as a vector of fixed size (`numFeatures=20`), regardless of the text's length. This step converts text into numerical data.

`IDF` (Inverse Document Frequency) adjusts the term frequencies by penalizing common words (like "a" or "the") and emphasizing unique terms, resulting in a more meaningful feature representation for classification.

Logistic regression is chosen as the classification algorithm. It uses the processed feature vectors (from TF-IDF) to predict the label.

The `featuresCol` specifies the input features, while `labelCol` specifies the target label.

A Pipeline is built to organize the sequence of transformations and model training steps:

- tokenizer;
- hashingTF;
- IDF;
- logistic regression.

This modular approach makes the workflow cleaner and easier to manage.

Since logistic regression requires numerical labels, the `StringIndexer` converts the text labels ("sports," "technology," "politics") into numerical indices (e.g., 0, 1, 2). This transformation is applied to the DataFrame.

The dataset is divided into training and testing sets using an 80-20 split. The model is trained on the training set and evaluated on the test set.

The `pipeline.fit(train)` trains the logistic regression model using the training data. All the preprocessing steps in the pipeline are automatically applied to the training data during this step.

The `model.transform(test)` applies the trained model to the test data and generates predictions for each sample. The results include the original text, the actual label, and the predicted label.

The predictions are displayed in a table format, showing the input text, the true category (`label`), and the predicted category (`prediction`). This helps evaluate the model's performance. The Spark session is stopped to free

up resources. The code implements a text classification system using PySpark's machine learning pipeline. It demonstrates the entire process, including text preprocessing (tokenization and TF-IDF), model training, and prediction. The example is small-scale but showcases how Spark can handle similar tasks efficiently on large datasets.

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.4)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7)
+--------------------+----------+----------+
|                text|     label|prediction|
+--------------------+----------+----------+
|Football players ...|    sports|       0.0|
|Government passes...|  politics|       2.0|
|Technology compan...|technology|       2.0|
+--------------------+----------+----------+

+--------------------+--------+----------+
|                text|   label|prediction|
+--------------------+--------+----------+
|Government passes...|politics|       1.0|
+--------------------+--------+----------+
```

Fig. 11.1. Output results of news classification system

Fig. 11.1 represents the results of a news classification system applied to a dataset of news articles. It contains the following columns:

text – content of the news article, summarized here for brevity.

label – actual category of the article (e.g., "sports," "politics," "technology").

prediction – predicted category by the classification model, encoded as numerical indices:

0.0 corresponds to "sports,"

1.0 corresponds to "politics,"

2.0 corresponds to "technology."

The first row shows an article classified as "sports," with a prediction of 0.0, which matches the actual label.

The second row shows an article about government activities, correctly classified under "politics" with a prediction of 2.0.

The third row shows a technology-related article, correctly predicted as "technology" (2.0).

The classification system uses text data and machine learning to predict the category of news articles accurately.

The results demonstrate the system's ability to match the predicted categories with the actual labels for most inputs, showcasing its reliability.

Example 2. Fraud detection in financial transactions using PySpark and Kafka.

```
  # Install the required libraries
  !pip install pyspark kafka-python
  from pyspark.sql import SparkSession
  from pyspark.ml.feature import VectorAssembler
  from pyspark.ml.classification import
LogisticRegression
  from pyspark.sql.functions import col
  from kafka import KafkaProducer, KafkaConsumer
  import json
  # Initialize Spark Session
  spark =
SparkSession.builder.appName("FraudDetection").getOrCre
ate()
  # Sample transaction data
  data = [
      (100, 1),    # Fraud
      (200, 1),    # Fraud
      (50, 0),     # Normal
      (30, 0),     # Normal
      (1500, 1),   # Fraud
      (2500, 1),   # Fraud
      (700, 0),    # Normal
      (1200, 1),   # Fraud
      (10, 0),     # Normal
      (10000, 1)   # Fraud
  ]
  columns = ["amount", "transaction_type"]
  df = spark.createDataFrame(data, columns)
  # Prepare data for machine learning
  assembler = VectorAssembler(inputCols=["amount"],
outputCol="features")
```

```
  df = assembler.transform(df).select(col("features"),
col("transaction_type").alias("label"))
  # Split data into training and test sets
  train, test = df.randomSplit([0.7, 0.3], seed=42)
  # Initialize and train the logistic regression model
  lr = LogisticRegression(maxIter=10, regParam=0.01)
  model = lr.fit(train)
  # Evaluate the model
  predictions = model.transform(test)
  predictions.select("features", "label",
"prediction").show()
  # Initialize Kafka Producer
  producer = KafkaProducer(
      bootstrap_servers='localhost:9092',
      value_serializer=lambda v:
json.dumps(v).encode('utf-8')
  )
  # Function to send transactions to Kafka
  def send_transaction(transaction):
      producer.send('transactions', transaction)
  # Simulated transaction
  transaction = {"amount": 1500}
  send_transaction(transaction)
  # Initialize Kafka Consumer
  consumer = KafkaConsumer(
      'transactions',
      bootstrap_servers='localhost:9092',
      value_deserializer=lambda x:
json.loads(x.decode('utf-8'))
  )
  # Process incoming transactions in real-time
  for message in consumer:
      transaction = message.value
      amount = transaction["amount"]
      # Predict fraud
```

```
      prediction =
model.transform(spark.createDataFrame([(amount,)],
["amount"]))
      fraud_prediction =
prediction.collect()[0]["prediction"]
      print(f"Transaction: {transaction}, Fraud
Prediction (1 - fraud, 0 - normal):
{fraud_prediction}")
```

A small sample dataset is created with two columns: `amount` (transaction amount) and `transaction_type` (1 for fraud, 0 for normal).

`VectorAssembler` is used to convert the amount column into a feature vector. The `features` column will be used for machine learning, while the `transaction_type` column serves as the target label.

Logistic regression model is trained using PySpark's MLlib. The model learns patterns in the data to classify transactions as fraudulent (1) or normal (0). Training is done on 70% of the data, with 30% reserved for testing. The trained model is applied to the test data to make predictions. The predictions DataFrame includes the features, actual label, and prediction. This helps assess the model's accuracy.

A producer sends a simulated transaction (e.g., {"amount": 1500}) to the Kafka topic named transactions.

The consumer listens to the transactions topic. When a transaction is received, it is passed to the logistic regression model for fraud prediction.

For each incoming transaction, the logistic regression model predicts whether the transaction is fraudulent (1) or normal (0). The results are displayed in real-time, simulating a real-world fraud detection system.

PySpark ensures scalability and efficient handling of large transaction datasets. Apache Kafka integrates seamlessly with PySpark to enable real-time fraud detection. The example demonstrates end-to-end machine learning, including data preparation, model training, and inference.

Example 3. Predicting heart disease risk using patient data.

In healthcare, AI can be used to analyze patient data and predict potential diseases based on clinical factors. For example, we can predict the risk of heart disease using a dataset like the Cleveland Heart Disease dataset and Scikit-learn for machine learning.

Below is a Python implementation to predict heart disease risk using logistic regression:

```
    # Install required libraries
    !pip install pandas scikit-learn matplotlib
    # Import necessary libraries
    import pandas as pd
    import numpy as np
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import classification_report,
accuracy_score
    import matplotlib.pyplot as plt
    # Load dataset (Heart Disease dataset example)
    url    =    "https://archive.ics.uci.edu/ml/machine-
learning-databases/heart-
disease/processed.cleveland.data"
    columns = ['age', 'sex', 'cp', 'trestbps', 'chol',
'fbs', 'restecg',
               'thalach', 'exang', 'oldpeak', 'slope',
'ca', 'thal', 'target']
    # Load the dataset into a pandas DataFrame
    df = pd.read_csv(url, names=columns, na_values="?")
    df = df.dropna()  # Remove rows with missing values
    # Prepare features and labels
    X = df.drop('target', axis=1)
    y = (df['target'] > 0).astype(int)  # Convert target
into binary: 1 (heart disease), 0 (no heart disease)
    # Split data into training and testing sets
    X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3, random_state=42)
    # Train a Logistic Regression model
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train, y_train)
    # Make predictions on the test set
    y_pred = model.predict(X_test)
    # Evaluate the model
```

```
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.2f}")
    print(classification_report(y_test, y_pred))
    # Plot feature importance
    importance = model.coef_[0]
    features = X.columns
    plt.figure(figsize=(10, 6))
    plt.barh(features, importance, color="skyblue")
    plt.xlabel("Feature Coefficient")
    plt.ylabel("Feature")
    plt.title("Feature Importance in Predicting Heart
Disease Risk")
    plt.show()
```

Cleveland Heart Disease dataset is a well-known dataset used in heart disease prediction. It contains features such as age, sex, chest pain type, cholesterol level, and more.

The target variable (target) indicates the presence of heart disease. Here, it is binarized into 1 (presence) and 0 (absence).

Missing values are handled by removing incomplete rows.

Features (X) and labels (y) are prepared for model training.

A logistic regression model is trained to predict the risk of heart disease.

The model learns the relationship between input features (e.g., age, cholesterol) and the target label (heart disease).

Accuracy and a classification report (precision, recall, F1-score) are used to evaluate model performance.

The importance of each feature is visualized using a bar plot. This helps interpret which clinical factors contribute most to heart disease risk.

Patient data can be collected in real time through medical devices or electronic health records. AI models predict heart disease risk in real time, aiding early diagnosis and intervention.

Distributed systems like Apache Kafka can be used to handle large-scale, real-time patient data for predictive analysis.

The results presented in the Fig. 11.1 show the performance of a logistic regression model for predicting heart disease risk, along with a visualization of feature importance based on the model's learned coefficients.

Accuracy: 0.88

	precision	recall	f1-score	support
0	0.87	0.92	0.89	49
1	0.89	0.83	0.86	41
accuracy			0.88	90
macro avg	0.88	0.87	0.88	90
weighted avg	0.88	0.88	0.88	90

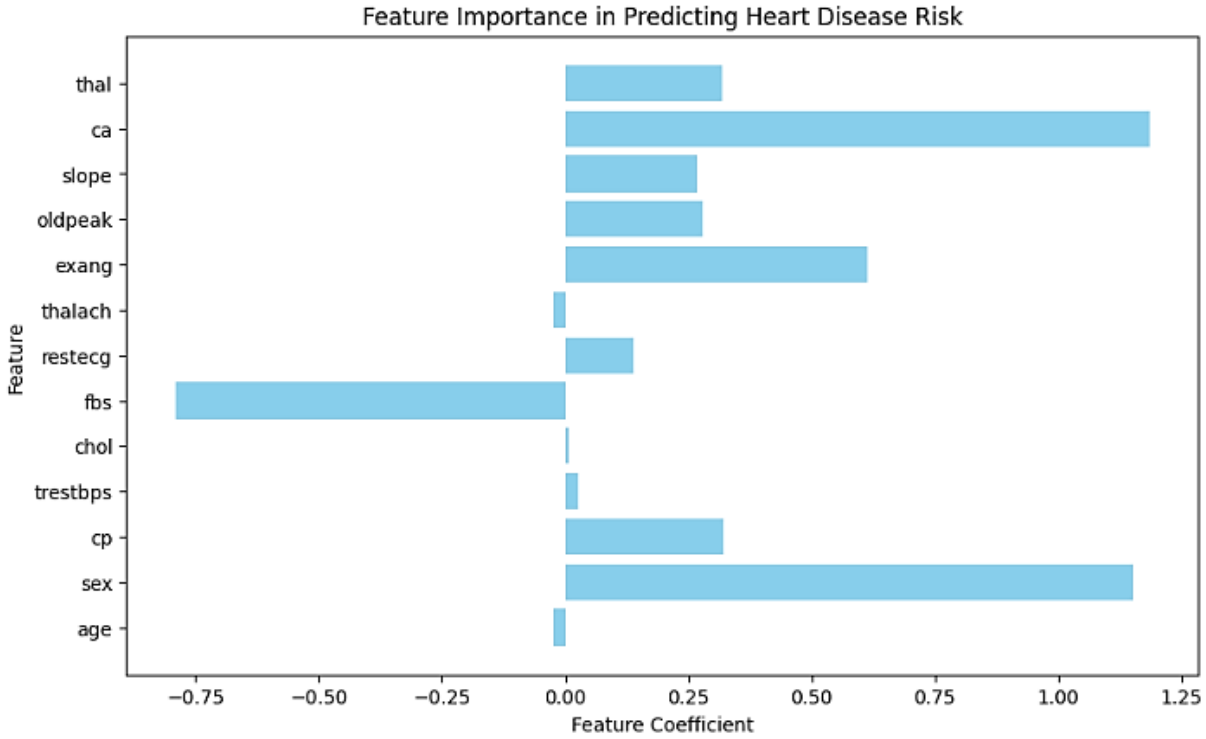


Fig. 11.1. Predicting heart disease risk results

The model achieved 88% accuracy, indicating that it correctly predicted heart disease risk in 88% of cases.

Precision for class 1 (heart disease present) is 89%, showing the proportion of correctly identified cases of heart disease out of all predicted positive cases.

Recall for class 1 is 83%, indicating the model's ability to identify true heart disease cases among all actual cases.

The F1-score, which balances precision and recall, is 86% for class 1 and 89% for class 0 (no heart disease).

The macro average and weighted average confirm balanced performance across both classes.

The bar plot illustrates the coefficients of features used in the model:

Features with positive coefficients (e.g., "thal," "ca," "cp," and "age") are positively correlated with an increased likelihood of heart disease. For instance, "thal" (thalassemia) and "cp" (chest pain type) are strong indicators.

Features with negative coefficients (e.g., "thalach" and "slope") suggest a reduced likelihood of heart disease. For example, higher "thalach" (maximum heart rate achieved) is associated with a lower risk.

Some features, such as "fbs" (fasting blood sugar) and "restecg" (resting electrocardiographic results), show minimal influence, as their coefficients are near zero.

The model's high accuracy and balanced performance suggest it is effective for predicting heart disease risk based on clinical data. The feature importance plot highlights which clinical factors have the greatest influence, helping healthcare professionals prioritize diagnostic tests and treatment decisions. Further validation with larger and more diverse datasets would ensure the model's generalizability.

Example 4. Internet of Things (IoT) and network management.

In the field of IoT, AI systems are used to analyze large volumes of data coming from connected devices. The integration of AI technologies allows for the implementation of “smart” network and resource management systems. Let us consider an example of integrating AI into an intelligent system for processing and analyzing heterogeneous data streams in real time from IoT devices in water management . The information system consists of three levels: data producer , data processor and consumers in the data.

At the data producer level, IoT devices and sensors generate heterogeneous data streams. The data can be structured (e.g. JSON, XML) or unstructured (e.g. CSV files). Data processing consists of two main components: a stream processing platform (Kafka Streams) and a complex event processing (CEP) engine, such as Esper. The stream processing platform takes heterogeneous data streams, homogenizes the data into a common format, and generates simple events. The CEP engine analyzes simple events to detect interesting situations. Data consumers are databases, smartphones, or web services that consume processed data and detected situations from the system.

The data processing layer contains the core components for homogenizing and analyzing heterogeneous data streams. The Kafka cluster

stores incoming heterogeneous data streams in Kafka topics. Data producers publish messages to input topics, while consumers can subscribe to output topics (Fig. 11.2). Kafka provides high throughput and scalability.

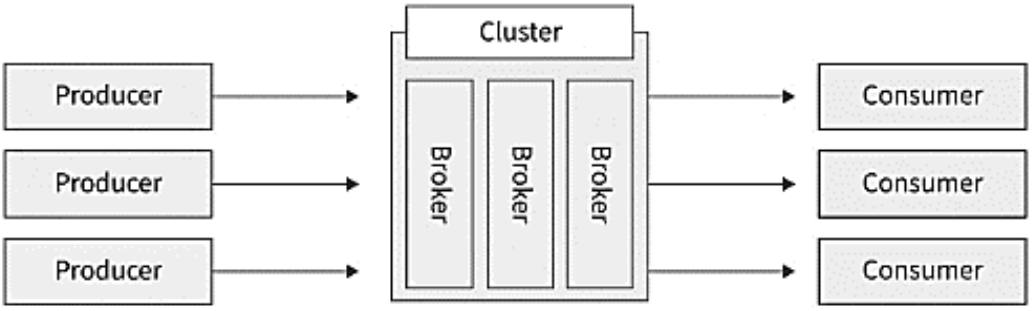


Fig. 11.2 . Kafka cluster

There is an input topic for sensor data from “smart” water meters and an output topic for detected anomalies and notifications.

The Kafka Streams program consumes messages from an input topic, homogenizes the data, and generates simple events for analysis (Fig. 11.3).

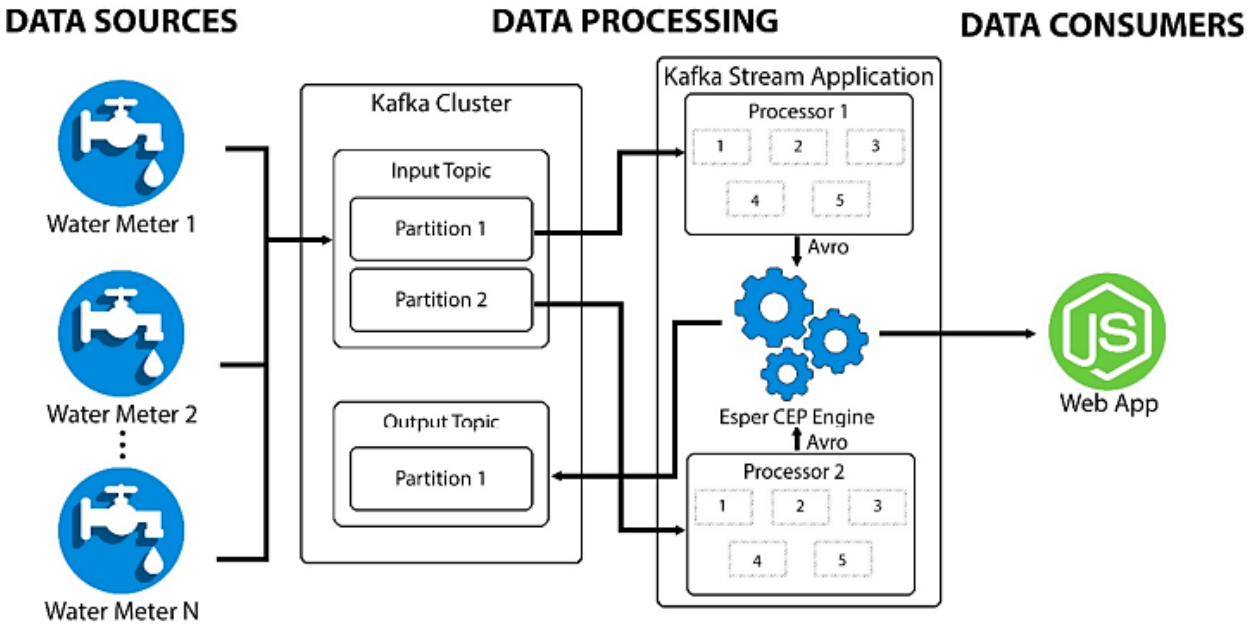


Fig. 11. 3. Software architecture adapted to the water resources management scenario

Basic steps for using the information system the following.

1. Loading data from the sections of the introduction topic.

2. Define the data format (JSON, XML, CSV).
3. Data homogenization into a common JSON format.
4. Create Avro schemas for the data.
5. Creating an Avro event for analysis.

The Esper CEP engine receives Avro events from Kafka Streams. This allows EPL patterns to be defined that match event conditions and detect interesting situations. When a pattern is matched, a new complex event is generated and published to the source topic. For example, anomalies such as water meter reading errors can trigger an alert. Heterogeneous data from smart water meters is processed and analyzed. Heterogeneous sensor data from smart water meters is consumed and homogenized in a JSON format with a schema. This generates simple “WaterMeasurement” events for analysis. A sample sensor data in JSON format looks like this:

```
{
"serialNumber": "A87SAA77188",
"dateTime": "2018-07-06T05:00:00.000Z",
"volumeM3": 5.4366397,
...
}
```

The Kafka Streams application in Java aggregates sensor data into an Avro event "WaterMeasurement":

```
class HomogenizeSensorData {
@StreamListener("input-topic")
@SendTo("cep-topic")
public WaterMeasurement process(String message) {
// Detect format
if (isJSON(message)) {
// Deserialize JSON
} else if (isCSV(message)) {
// Parse CSV
}
// Homogenize and create Avro event
WaterMeasurement event = ...;
return event;
}
}
```

The Esper CEP engine detects anomalies using EPL patterns in SQL:

```
INSERT INTO WaterMeterAnomaly
SELECT *
FROM WaterMeasurement
WHERE volumeM3 < 0
```

In general, the architecture provides real -time processing of heterogeneous data streams in an IoT water management system. The main advantages of such an architecture are:

- processing large volumes of heterogeneous data streams in real time;
- automatic data homogenization into a common format;
- real-time analysis and detection of interesting situations;
- high scalability by adding more stream processing nodes.

Long short-term memory (LSTM) neural networks are a type of recurrent neural network (RNN) that has proven effective for processing and predicting streaming data, particularly in time series analysis. LSTMs are designed to process sequences of data , and are able to process data points one at a time in a sequential manner, making them suitable for streaming data where the order of the data points matters. LSTMs can capture long-term dependencies in the data, which is important for understanding patterns and trends in time series data. Unlike traditional feed-forward neural networks, LSTMs can store information much earlier in the sequence. As new data points arrive, LSTMs can be updated and retrained in real time, allowing them to adapt to changing patterns and trends. LSTMs can handle sequences of varying lengths. In a streaming data scenario, the length of a data sequence can change over time, and LSTMs are flexible enough to handle this.

LSTMs can be used in state-preserving mode, where internal state is carried over from one data packet to another. This is particularly useful for streaming data, as it allows the network to maintain context across multiple time steps. LSTMs can automatically learn appropriate features from data, reducing the need for manual feature development. This is useful when working with streaming data, where the distribution of the data may change over time. LSTMs can handle multidimensional time series data, which is common in many streaming data applications. They can model relationships between multiple variables at once. LSTMs can be used to detect anomalies

in streaming data. By modeling the normal behavior of a system, they can detect deviations from that normal behavior in real time. LSTMs are often used for time series forecasting. They can be trained to predict future data points in a sequence, which is valuable for forecasting in streaming data applications. LSTMs can be integrated into real-time decision-making systems. To enable more efficient and intelligent processing, a predictive analytics module based on LSTM neural networks can be added between Kafka Streams and Esper CEP engine so that the LSTM model can take unified data streams from Kafka Streams as input and make multi-step predictions about expected values. For example, in a water resource usage scenario, we can predict expected consumption and detect when actual values differ significantly from the forecast.

Consider the pseudocode for AI integration:

```
DataPreparation(AvroStream):
format avro data as time series
LSTMModel(HistoricalData):
train LSTM network
return model
Predict(Model, AvroStream):
make multi-step predictions
calculate prediction errors
flag anomalies
AugmentAvroStream(AvroStream, Predictions, Anomalies):
add predictions and anomalies to Avro events
return AugmentedStream
```

Adding predictive capabilities allows the CEP engine to perform at a higher level, detecting pattern anomalies rather than isolated outliers. Choosing the right architecture and hyperparameters for a particular streaming application is crucial to achieving the best performance. The integration of recurrent neural networks can significantly improve the intelligent distributed streaming processing system. Although this AI-based approach requires further research, it has the potential to improve the efficiency of heterogeneous data analytics.

Example 5. Energy demand prediction and renewable energy integration.

In the energy sector, AI can optimize energy production and demand prediction by integrating renewable energy sources. Distributed systems allow for real-time monitoring and control, improving energy efficiency and

reducing costs. This example demonstrates the use of machine learning to predict energy demand and optimize the distribution of renewable energy resources. We'll use Python libraries such as Pandas and Scikit-learn to process the data and train a predictive model.

```
    import pandas as pd
    import numpy as np
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import mean_absolute_error
    import matplotlib.pyplot as plt
    # Simulated energy demand data with renewable energy
generation
    np.random.seed(42)
    data = {
        'hour': list(range(24)) * 30,   # 30 days of
hourly data
        'day_of_week': [i // 24 % 7 for i in range(24 *
30)],
        'temperature': np.random.uniform(15, 35, size=24
* 30),  # in degrees Celsius
        'wind_speed': np.random.uniform(0, 20, size=24 *
30),  # in m/s
        'solar_radiation':    np.random.uniform(0,    1,
size=24 * 30),  # normalized
        'energy_demand':   np.random.uniform(50,   500,
size=24 * 30),  # in kWh
    }
    df = pd.DataFrame(data)
    # Data preparation
    X   =   df[['hour',   'day_of_week',   'temperature',
'wind_speed', 'solar_radiation']]
    y = df['energy_demand']
    X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2, random_state=42)
    # Train a random forest regressor
```

```
    model = RandomForestRegressor(n_estimators=100,
random_state=42)
    model.fit(X_train, y_train)
    # Make predictions
    y_pred = model.predict(X_test)
    # Evaluate the model
    mae = mean_absolute_error(y_test, y_pred)
    print(f"Mean Absolute Error: {mae:.2f} kWh")
    # Feature importance
    feature_importances = model.feature_importances_
    feature_names = X.columns
    plt.barh(feature_names,        feature_importances,
color='skyblue')
    plt.title('Feature  Importance  in  Energy  Demand
Prediction')
    plt.xlabel('Importance')
    plt.ylabel('Feature')
    plt.show()
    # Visualize actual vs predicted energy demand
    plt.figure(figsize=(12, 6))
    plt.plot(y_test.values,    label='Actual    Energy
Demand', color='blue', alpha=0.7)
    plt.plot(y_pred,  label='Predicted  Energy  Demand',
color='orange', alpha=0.7)
    plt.legend()
    plt.title('Actual vs Predicted Energy Demand')
    plt.xlabel('Sample Index')
    plt.ylabel('Energy Demand (kWh)')
    plt.show()
    # Renewable energy integration
    # Optimize renewable energy distribution for peak
demand
    renewable_energy_generated  =  np.random.uniform(50,
300, size=24)  # hourly generation
    predicted_demand = model.predict(pd.DataFrame({
        'hour': list(range(24)),
```

```
        'day_of_week': [1] * 24,  # Assume it's Monday
        'temperature':     np.random.uniform(20,     30,
size=24),
        'wind_speed': np.random.uniform(5, 15, size=24),
        'solar_radiation':  np.random.uniform(0.3,  0.9,
size=24),
    }))
    # Balance renewable energy with demand
    energy_deficit = predicted_demand -
renewable_energy_generated
    for i, deficit in enumerate(energy_deficit):
        if deficit > 0:
            print(f"Hour {i}: Deficit of {deficit:.2f}
kWh, consider grid supply or storage.")
        else:
            print(f"Hour {i}: Surplus of {-deficit:.2f}
kWh, consider storing or selling back to the grid.")
```

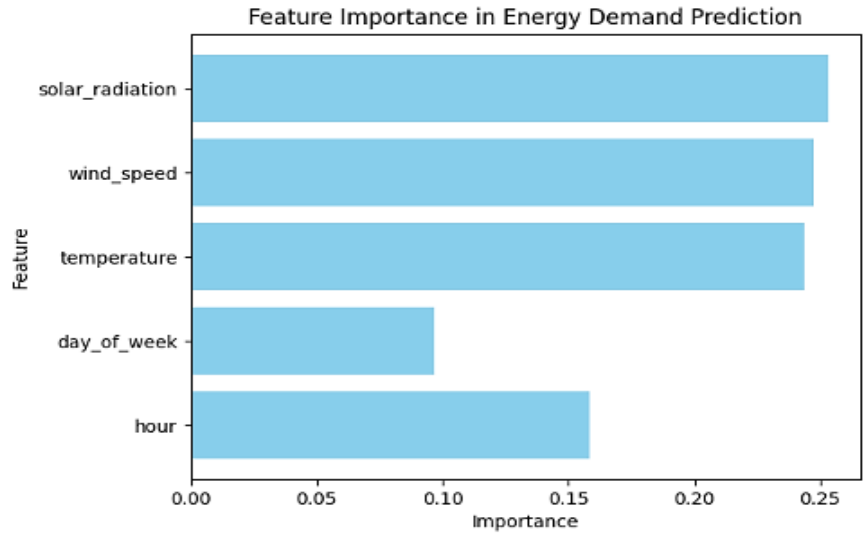
The data includes features such as time (hour, day of the week), temperature, wind speed, solar radiation, and the corresponding energy demand. These are typical factors affecting energy demand and renewable generation.

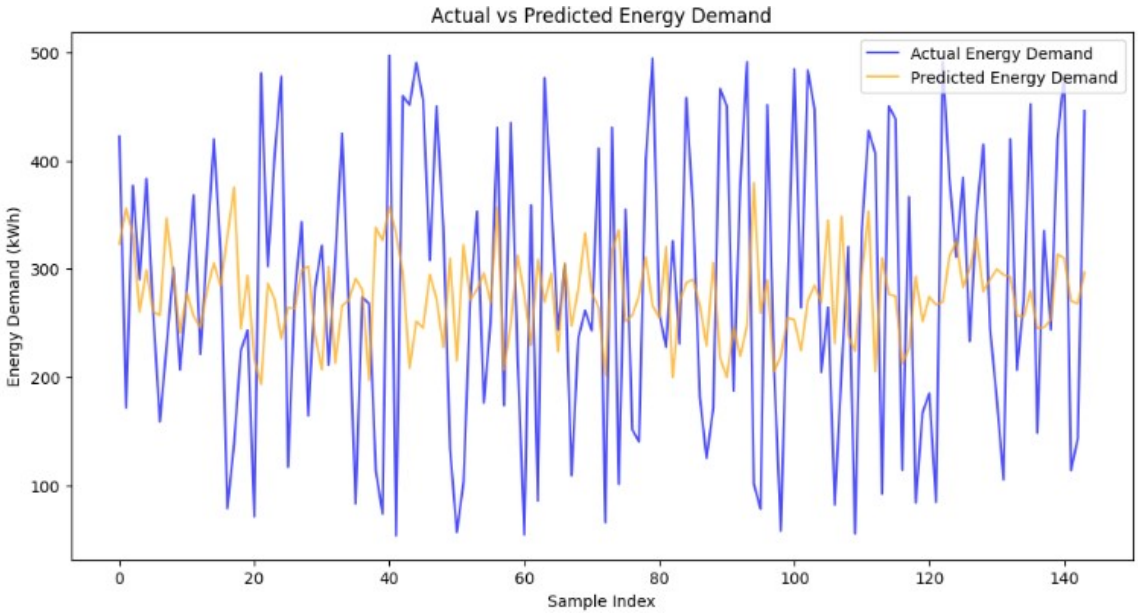
Random Forest Regressor is used to predict energy demand based on historical data. This method handles nonlinear relationships and feature interactions well.

The bar chart reveals which factors most significantly impact energy demand. For instance, temperature and solar radiation may be highly influential. The comparison plot between actual and predicted energy demand showcases the model's performance.

The feature importance chart identifies key predictors for energy optimization. Predicted energy demand is compared with simulated renewable energy generation to balance the supply and demand. Surpluses can be stored or sold back to the grid, while deficits can trigger additional grid supply. This approach demonstrates how AI can optimize energy demand prediction and the integration of renewable energy sources, enhancing efficiency and sustainability in distributed energy systems.

```
Mean Absolute Error: 118.52 kWh
```






```
Hour 0: Deficit of 101.47 kWh, consider grid supply or storage.
Hour 1: Surplus of 45.50 kWh, consider storing or selling back to the grid.
Hour 2: Deficit of 169.59 kWh, consider grid supply or storage.
Hour 3: Deficit of 187.66 kWh, consider grid supply or storage.
Hour 4: Deficit of 207.44 kWh, consider grid supply or storage.
Hour 5: Surplus of 12.30 kWh, consider storing or selling back to the grid.
Hour 6: Deficit of 92.17 kWh, consider grid supply or storage.
Hour 7: Deficit of 207.26 kWh, consider grid supply or storage.
Hour 8: Deficit of 87.96 kWh, consider grid supply or storage.
Hour 9: Deficit of 9.02 kWh, consider grid supply or storage.
Hour 10: Deficit of 95.22 kWh, consider grid supply or storage.
Hour 11: Deficit of 77.61 kWh, consider grid supply or storage.
Hour 12: Deficit of 27.27 kWh, consider grid supply or storage.
Hour 13: Deficit of 61.77 kWh, consider grid supply or storage.
Hour 14: Deficit of 144.80 kWh, consider grid supply or storage.
Hour 15: Surplus of 58.68 kWh, consider storing or selling back to the grid.
Hour 16: Deficit of 62.51 kWh, consider grid supply or storage.
Hour 17: Deficit of 123.85 kWh, consider grid supply or storage.
Hour 18: Deficit of 61.53 kWh, consider grid supply or storage.
Hour 19: Deficit of 177.34 kWh, consider grid supply or storage.
Hour 20: Deficit of 100.55 kWh, consider grid supply or storage.
Hour 21: Deficit of 22.82 kWh, consider grid supply or storage.
Hour 22: Deficit of 178.51 kWh, consider grid supply or storage.
Hour 23: Deficit of 215.15 kWh, consider grid supply or storage.
```

Fig. 11.4. Comparison of actual (blue line) and predicted (orange line) energy demand

The results in Fig. 11.4 demonstrate the analysis of energy demand prediction, feature importance, and optimization strategies for energy management in a distributed system. Here's a breakdown of the findings:

Solar radiation has the highest importance, indicating that it strongly influences energy demand, likely due to its direct impact on renewable energy generation. Wind speed and temperature are also significant factors, reflecting their effects on energy generation and consumption patterns.

Hour and day of week play smaller roles, capturing time-based variations in energy demand.

This analysis helps identify key factors to prioritize in improving predictive models and optimizing energy usage strategies.

The model performs well in tracking the overall trends of energy demand, as the predicted values closely follow the actual ones.

Some deviations exist, but they are generally within acceptable margins, as indicated by the calculated Mean Absolute Error (MAE) of 118.52 kWh.

This performance demonstrates the model's ability to capture key patterns and provide reliable predictions for energy management.

Surplus hours (e.g., Hour 1, Hour 5, etc.) suggest excess energy that can be stored or sold back to the grid for cost savings or revenue generation.

Deficit hours (e.g., Hour 0, Hour 3, etc.) indicate a shortfall, requiring grid supply or stored energy to meet demand.

Balancing demand during deficit periods through grid connections or demand-side adjustments.

The results underscore the importance of combining predictive analytics and optimization strategies for efficient energy management. By identifying key factors and forecasting demand, manufacturing systems or grid managers can enhance energy conservation, reduce costs, and ensure a sustainable balance between supply and demand.

11.2. Examples of the use of artificial intelligence technologies to optimize big data analytics processes

The combination of two cutting-edge technologies: artificial intelligence and big data analytics involves the use of AI-driven algorithms and machine learning techniques to analyze, interpret, and extract useful information from large and complex data sets. The main goal of artificial

intelligence in big data is to automate and improve the data analysis process, making it faster, more accurate, and more scalable.

AI for big data is based on machine learning models that can recognize patterns, make predictions, and continuously improve their performance with minimal human intervention. These models are trained on datasets, allowing them to identify trends, anomalies, and correlations that humans may not be able to detect or would take a long time to detect. In this way, AI for big data enables organizations to transform raw data into strategic assets, facilitating informed decision-making and gaining competitive advantage in their respective industries.

Big data and AI are not just complementary, they are interdependent. Big data provides the raw material, the vast data sets, with which artificial intelligence can work its "magic". The synergy between the two can be illustrated by the following steps.

Big data encompasses the collection of vast amounts of structured and unstructured data from various sources, including sensors, social media, customer interactions, etc. This data is the foundation for AI applications.

Big data technologies like Hadoop and Spark make it easy to store and process massive data sets. This infrastructure ensures that data is available to AI algorithms.

Before AI can analyze data, it often requires preprocessing. This step involves cleaning, transforming, and structuring the data to make it suitable for machine learning models.

Machine learning algorithms are applied to trained data. These algorithms can include supervised learning for prediction, unsupervised learning for pattern recognition, and reinforcement learning for decision making.

AI models are trained on historical data to learn patterns and relationships. Once trained, they can make predictions or make decisions based on new inputs in real time. The end result of this process is actionable information that can guide decision-making. Artificial intelligence algorithms uncover hidden patterns, anomalies, trends, and predictions in big data that can be used for a variety of purposes, from improving products and services to optimizing business operations.

When it comes to choosing the right AI model for big data, there is no one-size-fits-all solution. The choice depends on the specific needs and goals

of the organization. Machine learning is a fundamental component of AI for big data, comprising various techniques such as supervised learning, unsupervised learning, and deep learning. Supervised learning, for example, is used for classification and regression tasks, making it suitable for predictive analytics with big data.

Natural Language Processing (NLP) is a subset of artificial intelligence that focuses on the interaction between computers and human language. It is especially important for large-scale analysis of unstructured text data, such as customer reviews, social media posts, or news articles.

Computer vision allows machines to interpret and understand visual information from the world, including images and videos. This technology is invaluable for tasks such as image recognition, object detection, and facial recognition, which can be applied to big data analytics scenarios.

Let's look at a Python code example that uses the OpenCV library to perform real-time face recognition from a webcam video stream. This example uses a pre-trained face detection model:

```
import cv2
# Loading a pre-trained face recognition model
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
# Opening a video stream from a webcam
cap = cv2.VideoCapture(0)
while True:
# Reading a frame from a video stream
ret, frame = cap.read()
# Convert frame to black and white format
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Detect faces in the frame
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
# Drawing rectangles around detected faces
for (x, y, w, h) in faces:
cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
# Frame display
cv2.imshow('Face Detection', frame)
# Exit the loop when pressing the 'q' key
if cv2.waitKey(1) & 0xFF == ord('q'):
break
# Freeing resources
cap.release()
cv2.destroyAllWindows()
```

First, a pre-trained face detection model is loaded from the `haarcascade_frontalface_default.xml` file. Then, for each frame from the video stream, the image is converted to black and white for better processing, after which the algorithm detects faces and draws rectangles around them. The video stream is displayed with the rectangles superimposed (Fig. 11.6), and the program can be stopped by pressing the 'q' key.

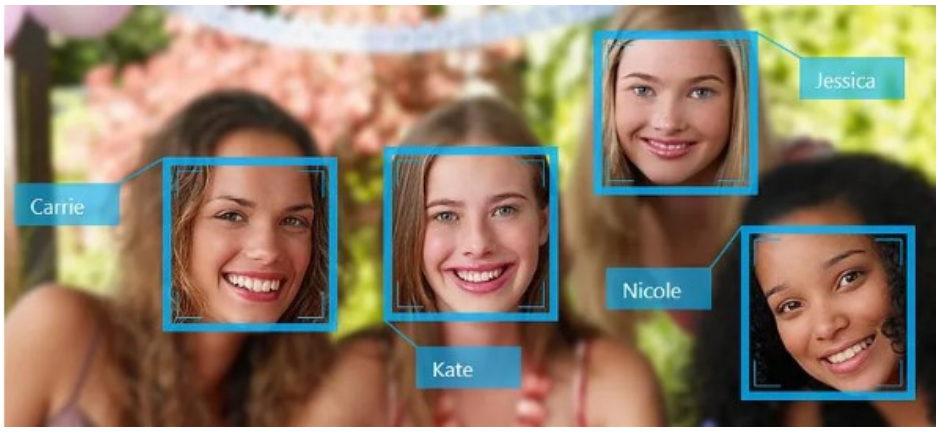


Fig. 11.6 . Face recognition example

Reinforcement Learning used in cases where decision – making is crucial and is well suited to optimizing complex systems and processes, such as supply chain management or autonomous vehicles, through learning through interaction.

Deep learning, a subset of machine learning, involves neural networks with multiple layers. It is particularly effective for tasks that require high-accuracy pattern recognition, such as speech recognition or image classification.

Choosing the best AI technology depends on the specific goals of big data analytics project. In many cases, a combination of various AI techniques may be needed to extract the most valuable information from diverse data sets. AI-driven algorithms automate the data analysis process, significantly saving time and reducing human error. These algorithms efficiently process huge data sets, uncovering hidden patterns and trends that might otherwise go unnoticed.

Whether predicting customer behavior, equipment failures, or market trends, AI enables decision-making with actionable information. AI algorithms can detect anomalies in data sets, a critical capability for tasks such as fraud detection, network security, and quality control. AI-powered recommendation systems use big data to offer personalized content and product suggestions, such as Netflix and Amazon. Natural language processing in AI allows organizations to analyze and understand customer sentiment, feedback, and textual opinions, helping to improve products and services.

AI is being used to analyze patient data, help diagnose diseases, predict patient outcomes, and even personalize treatment plans based on individual medical records. Financial institutions are using artificial intelligence for fraud detection, credit risk assessment, and chatbots for customer service.

In retail, AI-based recommendation engines personalize the shopping process, optimize inventory management, and enable dynamic pricing strategies.

In manufacturing, predictive maintenance using artificial intelligence reduces downtime by predicting equipment failures, and quality control systems improve product quality.

Artificial intelligence also improves marketing campaigns by analyzing customer behavior, segmenting audiences, and optimizing ad targeting.

Artificial intelligence for big data is a great combination that enables organizations to extract value from their huge and complex data sets. Big data analytics focuses on collecting, storing, and processing large amounts of data, while AI is concerned with creating algorithms and models for tasks such as pattern recognition and decision-making.

The main purpose of Big Data is it is data management and analysis, while the goal of AI extends to creating intelligent systems that can perform tasks autonomously. Big data provides the “raw material” and artificial intelligence processes and interprets this material to generate information and provide control.

11.3. Trends in the development of artificial intelligence technologies and their impact on the modern approach to big data analytics

Trends in artificial intelligence technologies are having a significant impact on the approach to big data analytics. The application of AI is expanding into various industries, including medicine, finance, manufacturing, and others. This expansion allows for the use of more data sources and deeper understanding of large data sets.

The growth of computing power allows the use of more sophisticated machine learning models and neural networks to analyze large amounts of data, leading to more accurate and faster results.

The increasing efficiency of Natural Language Processing (NLP)

technologies allows for the analysis and understanding of text information from various sources, making it easier to work with large volumes of text. Image processing and video analytics technologies are becoming more accurate and efficient, allowing for the analysis of large volumes of multimedia data.

AI technologies in healthcare are used to predict diseases, tailor treatments, and optimize medical diagnostics. For example, machine learning algorithms analyze large amounts of patient data, such as medical records, genetic information, and device data, to detect early signs of diseases, such as cancer or cardiovascular disease. This allows doctors to predict the development of diseases and take preventive measures. AI helps develop individual treatment plans, taking into account the specific characteristics of each patient, including genetic markers, medical history, and lifestyle. This increases the effectiveness of treatment and reduces the risk of side effects. AI technologies optimize medical diagnostics by analyzing medical images and laboratory results with high accuracy. For example, deep learning systems can automatically recognize pathologies in X-rays or MRI scans, which significantly reduces the time to establish a diagnosis and reduces the likelihood of human error. The introduction of AI into medicine contributes to a more accurate, faster, and personalized approach to treatment and diagnostics, improving the quality of medical services and patient health.

Algorithms allow data to be processed across multiple nodes in a system without the need for centralized exchange of large amounts of information, providing greater privacy and security. For example, in federated learning, a machine learning model is trained on multiple devices or servers that exchange only updated model weights, not the data itself, which allows information to be kept private. This is an approach used in mobile phones, where each device can train the model locally on its data and then send only the changes to a central model, which collects and summarizes those changes without the need to transmit or store the raw data on a server.

Another example is the use of blockchain to coordinate decentralized model training, where nodes process data independently of each other and record the training results on the blockchain, ensuring transparency and security of the process without the need for centralized management. This can be useful in industries where data confidentiality is critical, such as financial technology or healthcare. Decentralized machine learning algorithms reduce

the risks associated with centralized storage and processing of big data, ensure scalability of systems, and increase the level of security and confidentiality of the processed information, making them ideal for use in distributed networks and systems with high security requirements.

Voice processing technologies are gaining popularity for interacting with big data analytics systems through voice commands and audio data analysis.

Control questions and tasks for topic 11

List of theoretical questions

1. What artificial intelligence technologies do you consider most effective for integration into distributed systems?
2. What challenges may arise when integrating artificial intelligence technologies into distributed systems, and how can they be overcome?
3. How do artificial intelligence technologies help solve the problems of optimizing big data analytics processes in real time?
4. What popular trends in the development of artificial intelligence technologies do you see that could influence the approach to big data analytics?
5. How can natural language processing technologies improve the analysis of textual information in large amounts of data?
6. What opportunities do you see in the development of the integration of artificial intelligence technologies and big data analytics in the future?

List of tasks for independent work

Practical task. Image classification using a convolutional neural network (CNN).

It is necessary to use a convolutional neural network (CNN) for image classification, using the example of determining pneumonia from chest X-rays. Use the command `!pip install tensorflow keras matplotlib kaggle` to install the TensorFlow, Keras, Matplotlib, and Kaggle libraries required for loading and analyzing data.

Create a kaggle.json file containing your Kaggle credentials (username and key) to be able to download datasets directly from Kaggle. Download the chest x-ray dataset used to diagnose pneumonia using the command `!kaggle`

`datasets download -d paultimothymooney/chest-xray-pneumonia`.

Unzip the downloaded file using the `!unzip command`.

Import the necessary libraries for building and training a neural network, processing images, and visualizing the results.

Set paths to training, validation, and test datasets. Use `ImageDataGenerator` for image preprocessing such as resizing and pixel normalization.

Load data using `flow_from_directory` methods.

Create a CNN with multiple convolutional and connection layers using the Sequential API. Add convolution (`Conv2D`), pooling (`MaxPooling2D`), smoothing (`Flatten`), dense layers (`Dense`), and a dropout layer (`Dropout`) to reduce overfitting.

Use the `binary_crossentropy loss function`, the `adam optimizer` , and the accuracy metric to compile the model.

Train the model on training data using the `fit` method, with validation on a separate validation dataset. Evaluate the model's accuracy on test data using the `evaluate method`.

Plot graphs showing the accuracy and loss during training and validation of the model to analyze its performance. You should obtain a model that is capable of classifying X-rays as containing or not containing signs of pneumonia, and analyze its performance based on the resulting graphs and metrics.

List of used and recommended literature

11.1. Corral-Plaza, D., Medina-Bulo, I., Ortiz, G., & Boubeta-Puig, J. A stream processing architecture for heterogeneous data sources in the Internet of Things. Computer Standards & Interfaces. 70 , 2020. 103426. DOI: 10.1016/j.csi.2020.103426.

11.2. The History of AI [Electronic resource]. – Access mode: https://www.w3schools.com/ai/ai_history.asp

11.3. Maksym Ilin, Liubov Oleshchenko. Current state and development prospects of heterogeneous streaming data processing methods. Information Technology: Computer Science, Software Engineering and Cyber Security, 1, 2023. P. 100-106. https://doi.org/10.32782/IT/2023-1-13

Topic 12. Big data cloud providers

Using Big Data cloud providers opens up a lot of opportunities for enterprises and organizations working with large amounts of data. Cloud providers provide the ability to scale computing and storage resources depending on the user's needs. The ability to quickly scale resources up or down enables efficient processing of large amounts of data without requiring investment in dedicated equipment.

12.1. Benefits of using big data cloud providers

Big Data cloud services provide tools for real-time data analysis. Businesses can quickly respond to changes in large data streams, which is critical for making timely decisions.

Cloud providers ensure high security standards for storing and processing sensitive data. Customers can trust providers with regards to privacy and compliance with regulatory requirements.

Cloud platforms offer a variety of services and tools for processing and analyzing big data, such as Apache Hadoop, Apache Spark, integrated databases, etc. Users can use ready-made solutions without the need to manage and maintain their own infrastructure components.

Instead of investing in large server farms and equipment, users can use the services of cloud providers on a pay-per-use basis. This reduces capital investment costs and enables cost optimization based on actual needs.

Cloud solutions integrate with other services and applications. Flexibility and speed of integration with infrastructure and applications already in use are ensured. Cloud platforms automate data backup and recovery processes. Data reliability and availability are ensured, as well as the ability to quickly recover in the event of accidents or data loss.

Using cloud providers to process and store big data can provide significant resource savings compared to traditional on-premises solutions.

Reducing infrastructure capital expenditures can provide savings of up to 70-80%. For example, a company can avoid the costs of purchasing, deploying, and maintaining servers and network devices by instead using infrastructure as a service (IaaS) from cloud providers such as Amazon Web Services or Microsoft Azure. A startup that chooses to use AWS for its computing needs can save up to 80% on initial hardware costs.

Pay-as-you-go can provide savings of up to 30-50%. The customer pays only for the computing and storage resources that are actually used, instead of constantly maintaining redundant infrastructure.

For example, using Google Cloud BigQuery for big data analytics reduces data storage and computing costs because the company pays only for the queries that are executed and the amount of data that is processed.

Scaling resources on demand can save up to 20-30% (depending on peak load volumes). For example, by using cloud services such as AWS EMR (Elastic MapReduce), companies can automatically scale their computing resources to process large amounts of data during peak loads and scale them down during off-peak periods, avoiding unnecessary costs for excess capacity. Retailers can efficiently scale resources during holiday sales and reduce costs during normal times.

Using cloud providers can reduce maintenance and upgrade costs by up to 20-40%. Cloud providers take care of keeping hardware and software up to date, allowing companies to reduce infrastructure upgrade and maintenance costs. Using Azure HDInsight to manage Hadoop or Spark clusters can significantly reduce software upgrade and cluster management costs.

Flexibility in choosing and integrating different services can provide savings of up to 15-25%. By being able to integrate different services and platforms in the cloud, companies can reduce the costs of developing and integrating systems. For example, integrating AWS Redshift with other AWS services, such as S3 or Kinesis, allows companies to avoid the costs of creating complex integrations between separate platforms.

Using cloud providers for Big Data can lead to overall resource savings of 40-70% depending on the scale of the project, scaling needs, and specifics of data usage. Cloud providers provide companies with the opportunity to significantly reduce infrastructure, maintenance, and resource scaling costs, making them an effective solution for processing big data.

12.2. Overview of big data cloud providers

Big data cloud providers offer a wide range of tools and platforms for processing, storing, and analyzing large amounts of data, providing organizations with scalability, flexibility, and resource savings.

Amazon Web Services (AWS) is one of the market leaders, offering services such as Amazon EMR for data processing and Amazon S3 for storage, as well as tools for machine learning, analytics, and visualization.

For real-time data collection, we can use Amazon Kinesis, which allows users to process large data streams, for example, from websites, mobile applications, or IoT devices. Data that requires long-term storage can be placed in Amazon S3, which provides secure and scalable cloud storage. For example, we could create a Kinesis Stream to collect clickable data from a website and set up an integration with Amazon S3 to store this data for further analysis.

Amazon Redshift is a fast, fully managed data warehouse service that allows users to perform analytics on petabytes of data using SQL queries. For example, a company might load transaction data from S3 into Redshift, set up indexes on important fields like date and amount, and execute queries to identify trends or patterns in customer purchases or financial transactions. This process helps businesses make more informed decisions and improve their strategies..

Amazon QuickSight is a business intelligence (BI) service that allows users to quickly create interactive dashboards and data visualizations. Amazon QuickSight can be connected to Redshift or S3 to create visualizations that provide managers and analysts with a convenient way to monitor key performance indicators (KPIs) (Fig. 12.1).

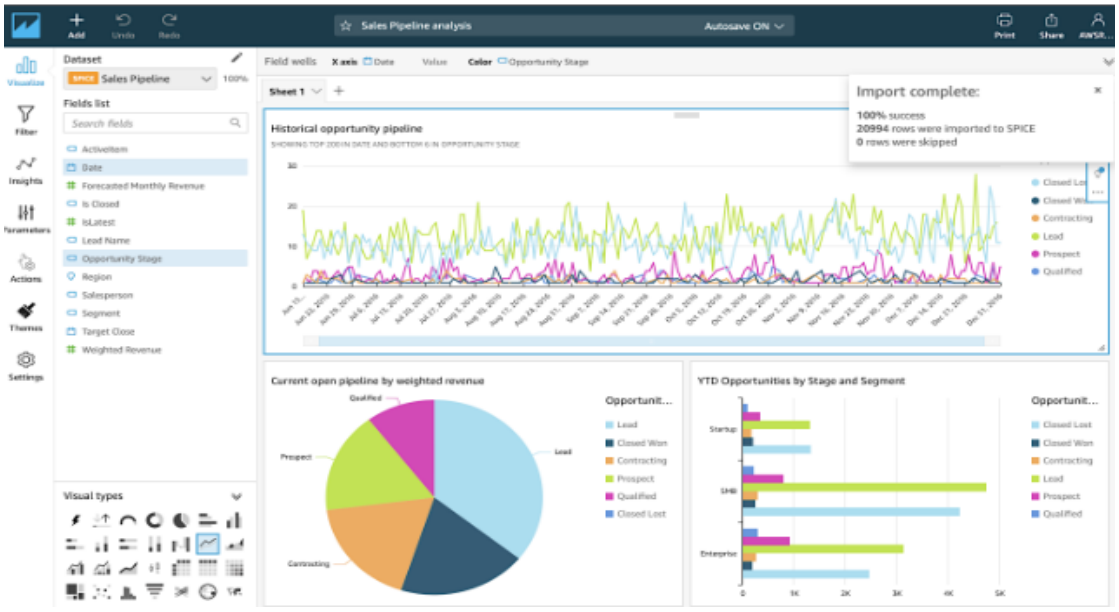


Fig. 12. 1. Data visualization in Amazon QuickSight

Amazon SageMaker provides tools for developing, training, and deploying machine learning models at scale. Data from Amazon S3 or Redshift can be used to train a machine learning model in SageMaker, and then deploy that model for predictive analytics, such as predicting customer churn or detecting fraud.

Amazon EMR (Elastic MapReduce) is a cloud-based big data processing platform that allows users to run and scale clusters of Hadoop, Spark, Presto, and other data analytics frameworks. It is suitable for tasks such as log analysis or processing large data sets for machine learning. An example is creating a cluster in Amazon EMR, loading data from S3, and performing data processing using Apache Spark.

One of the main advantages of EMR is its integration with other AWS services such as S3, EC2, and RDS, which provides data and infrastructure management.

EMR also scales automatically, adapting resources to current needs, which allows users to reduce costs. Thanks to this, users can process large volumes of data faster and more efficiently, paying only for the resources actually used. EMR supports enterprise-level security, including data encryption and access control, making it an ideal solution for organizations that work with sensitive information (Fig. 12.2).

Fig. 12. 2. Amazon EMR integration capabilities

Microsoft Azure offers a similar set of services, including Azure HDInsight for data processing, Azure Data Lake for storage, and Azure Synapse for analytics and integration with other Microsoft services. Azure HDInsight is a fully managed big data processing service that supports various frameworks such as Hadoop, Spark, Hive, and HBase, and allows users to efficiently scale data processing for various analytical tasks. Azure Data Lake provides a high-performance, scalable data warehouse that allows users to store data in any format while providing access for analytics and machine learning. Azure Synapse (formerly known as Azure SQL Data Warehouse) is an analytics platform that combines data integration, data warehousing, and big data analytics, providing a comprehensive approach to building scalable analytics solutions in the cloud (Fig. 12.3). These services allow companies to process and analyze data, integrating it with other Azure services such as Power BI for visualization and Azure Machine Learning for building artificial intelligence models.

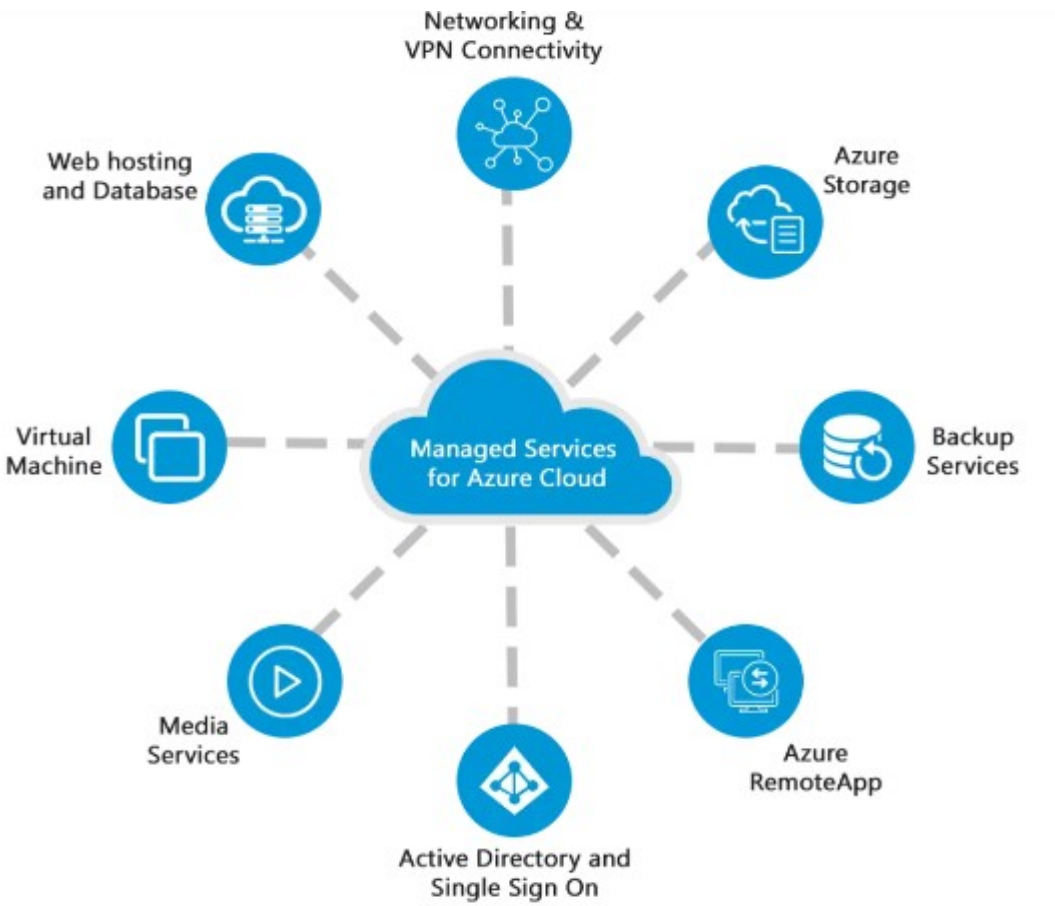


Fig. 12. 3. Azure Cloud capabilities

Google Cloud Platform (GCP) is also a strong player in this area, offering BigQuery for fast SQL analytics, Google Cloud Storage for storing large amounts of data, and machine learning tools like the AI Platform (Fig. 12.4).

Fig. 12. 4. Google Cloud Platform services capabilities

In addition to these major providers, there are others, such as IBM Cloud, which focuses on analytics and machine learning, and Oracle Cloud, which provides comprehensive data management and analytics.

IBM Cloud offers services such as IBM Watson for developing intelligent applications, IBM Cloud Pak for Data for data integration and management, and IBM Analytics Engine for analyzing large amounts of information. With these capabilities, companies can build, deploy, and manage analytics applications and solutions in the cloud, gaining valuable information for decision-making.

The cloud providers reviewed provide solutions that allow organizations to reduce infrastructure costs by paying only for the resources they use, which can result in savings of up to 70% compared to traditional solutions. These platforms also provide high data security, automated backups, and the ability to quickly scale based on business needs.

An overview of the most well-known big data cloud providers includes a description of the providers' main characteristics, their services, and approaches to working with large volumes of data (Table 12.1).

Table 12.1. **Overview of well-known big data cloud providers**

Cloud service	Characteristics	Advantages
Amazon Web Services (AWS)	AWS Elastic MapReduce (EMR) provides the ability to scale computation to process large amounts of data. Services: Amazon S3 for storage, EMR for compute, Redshift for analytics.	AWS offers a large set of tools for data analysis and processing. Simplified configuration: EMR allows quickly deploy and manage Hadoop clusters.
Microsoft Azure	High compatibility with other Microsoft products such as SQL Server and Excel. Services: Azure HDInsight for compute, Azure Data Lake Storage for storage.	Integration with other cloud and on-premises solutions from Microsoft.
Google Cloud Platform (GCP)	Google Cloud is known for its tools for developing and using machine learning models. Services: BigQuery for analytics, Cloud Storage for storage.	Integration with artificial intelligence services such as TensorFlow.
IBM Cloud	Focus on enterprises, providing ready-made solutions for large corporations. Services: IBM Analytics Engine, Db2 Big SQL for analytics, Cloud Object Storage for storage.	Providing a wide range of integrated services from a single IBM provider.
Oracle Cloud	Database focus: Increased focus on database management in the cloud. Services: Oracle Big Data Service, Oracle Cloud Infrastructure Object Storage.	Integration integration with existing Oracle databases.

Amazon Web Services (AWS) is renowned for its Elastic MapReduce (EMR) service, which simplifies the deployment and management of Hadoop clusters for large-scale data processing. AWS provides a comprehensive suite of services, including Amazon S3 for storage, EMR for compute, and Redshift for analytics. Its vast array of tools and simplified configuration process make it a popular choice for organizations needing scalable and efficient data analysis. Microsoft Azure stands out for its high compatibility with other Microsoft products, such as SQL Server and Excel. Its services, like Azure HDInsight for computation and Azure Data Lake Storage for storage, enable seamless integration with both cloud and on-premises Microsoft solutions. This makes Azure an excellent choice for businesses already invested in the Microsoft ecosystem.

Google Cloud Platform (GCP) excels in the field of machine learning and artificial intelligence. With tools like BigQuery for analytics and Cloud Storage for data storage, GCP provides robust support for data analysis while offering seamless integration with AI services like TensorFlow. This makes it perfect for organizations looking to harness AI-driven information from their data. IBM Cloud caters primarily to large enterprises, offering ready-made solutions tailored for corporate environments. Services such as IBM Analytics Engine, Db2 Big SQL, and Cloud Object Storage enable companies to access a wide range of integrated services from a single provider, streamlining operations and reducing complexity.

Oracle Cloud is heavily focused on database management, providing services such as Oracle Big Data Service and Oracle Cloud Infrastructure Object Storage. Oracle Cloud’s ability to integrate seamlessly with existing Oracle databases makes it an attractive choice for organizations relying heavily on Oracle’s database solutions.

12.3. AWS Big Data platform

Amazon Web Services offers a wide range of services for processing, analyzing, and storing large amounts of data under the name AWS Big Data. Let's take a closer look at the main components and services of this platform.

Amazon S3 (Simple Storage Service) is an object storage service that provides secure and scalable storage for any amount of data. S3 is often used to store large amounts of unstructured data, such as logs, images, or videos. It offers high durability and availability, making it ideal for backup, archiving,

and content distribution. For example, media companies often use S3 to store and serve high-resolution videos to users worldwide. With its easy integration into various applications and support for advanced features like versioning, lifecycle policies, and data encryption, S3 is a popular choice for both small businesses and large enterprises looking for reliable and flexible cloud storage solutions. For example, Amazon S3 is widely used by streaming video companies like Netflix to store and distribute multimedia content. Movies and TV shows are uploaded to S3 and then delivered to users via a global content delivery network (CDN), ensuring fast and reliable access to the video regardless of the viewer's location.

Amazon EMR (Elastic MapReduce) is a service for processing and analyzing large amounts of data based on Apache Hadoop and Apache Spark. EMR allows users to create and scale clusters for data processing using various frameworks. It is particularly useful for big data applications such as data warehousing, log analysis, machine learning, and ETL (Extract, Transform, Load) processes. For instance, a financial institution might use EMR to analyze transaction data in real-time, detecting fraud patterns, or to process large datasets for predictive analytics. With its flexible and cost-effective pricing model, EMR can scale to handle petabytes of data while integrating seamlessly with other AWS services like S3, DynamoDB, and Redshift, providing a comprehensive data processing ecosystem. For example, financial institutions use Amazon EMR to detect fraudulent transactions in real time. By analyzing large amounts of financial transaction data with Apache Spark, companies can quickly identify anomalous patterns in user behavior and take action to prevent fraud.

Amazon Redshift is a high-performance cloud database for big data analytics. Redshift is used to store and analyze structured data, providing high query speed. It is particularly designed for data warehousing applications, where large volumes of structured data need to be stored and queried quickly. For example, a retail company might use Redshift to analyze customer purchasing behavior, allowing for rapid querying of millions of transactions to uncover trends. Redshift integrates with various BI tools and AWS services, offering scalable storage and compute resources that allow businesses to efficiently manage their analytics workloads, while its columnar storage format and parallel processing capabilities further enhance performance and reduce query times. For example, e-commerce companies

use Amazon Redshift to analyze customer behavior and optimize sales. By processing large amounts of data on product views, orders, and user interaction with the site, Redshift allows quickly build analytical reports, forecast demand, and generate personalized recommendations for customers.

AWS Glue is a fully managed service for preparing and loading data for analysis. Glue automates data preparation processes, allowing to integrate data from different sources. It simplifies tasks like data cleaning, transformation, and cataloging, making it easier to prepare data for analytics in a timely and cost-effective manner. For instance, a company might use AWS Glue to combine data from various databases, CSV files, and APIs, transforming and organizing it into a unified format for analysis in Amazon Redshift or other data warehouses.

With built-in support for scheduling, monitoring, and managing data workflows, Glue ensures that the data pipeline is efficient, scalable, and ready for machine learning or business intelligence applications. For example, financial institutions use AWS Glue to bring together data from transactions, customer profiles, and external sources such as stock quotes. Glue automatically cleans, transforms, and catalogs data, allowing analysts to quickly query and extract valuable insights for risk assessment, fraud detection, or financial product personalization.

Amazon Kinesis is a set of services for working with streaming data. Kinesis allows users to collect, process, and analyze streaming data in real time. It is especially valuable for applications that need real-time information from continuously generated data, such as social media feeds, financial transactions, or data from IoT devices. For example, a company might use Kinesis to monitor sensor data from a fleet of vehicles in real time, triggering alerts for maintenance issues or tracking driving behavior.

Kinesis integrates seamlessly with other AWS services like Lambda, S3, and Redshift, enabling users to build real-time data processing pipelines and store or analyze data as it flows into the system, offering scalability and fault tolerance for handling massive streams of data.

For example, social media and online platforms use Amazon Kinesis to analyze real-time streaming data about user activity.

The system processes events such as likes, comments, and video views, allowing platforms to instantly tailor content recommendations, detect suspicious activity, or analyze the popularity of trending topics.

Amazon Athena is a service that allows users to perform analytical queries on data stored in Amazon S3 using standard SQL. Athena enables running analytical queries without the need to load data into a database, making it a cost-effective and flexible solution for querying large datasets. For example, a data scientist might use Athena to analyze log files stored in S3, without the complexity of setting up a separate data warehouse. Since Athena is serverless, users only pay for the queries they run, which helps manage costs. For example, marketing companies use Amazon Athena to analyze large website log files stored in Amazon S3. Using SQL queries, analysts can quickly obtain information about traffic sources, user behavior, and the effectiveness of advertising campaigns without having to set up complex analytics systems or migrate data to separate databases.

Amazon QuickSight is a visualization and business intelligence service. QuickSight allows users to create visualizations and dashboards for data analysis, in particular, using data from other AWS services. It enables businesses to turn complex datasets into interactive charts, graphs, and reports that are easy to understand and share.

For example, a company could use QuickSight to visualize sales trends across different regions by pulling data from Amazon Redshift or S3. Its scalability, integration with AWS data services, and pay-per-session pricing model make it a powerful tool for organizations of all sizes to democratize data access and make data-driven decisions.

AWS Big Data provides a comprehensive set of tools for working with large amounts of data in the cloud. From efficient storage to data analysis and visualization, this platform allows organizations to harness the power of computing and analytics to solve complex analytics problems.

One of the key strengths of AWS Big Data tools is their seamless integration with services like Amazon EMR for big data processing, AWS Glue for data preparation, and Amazon Athena for interactive querying. When combined with Amazon QuickSight, these tools enable end-to-end data pipelines – from raw data ingestion to insightful visualization – within a unified cloud environment. This ecosystem empowers data engineers, analysts, and decision-makers to collaborate efficiently, reduce time-to-insight, and scale analytics solutions without the need for extensive infrastructure management.

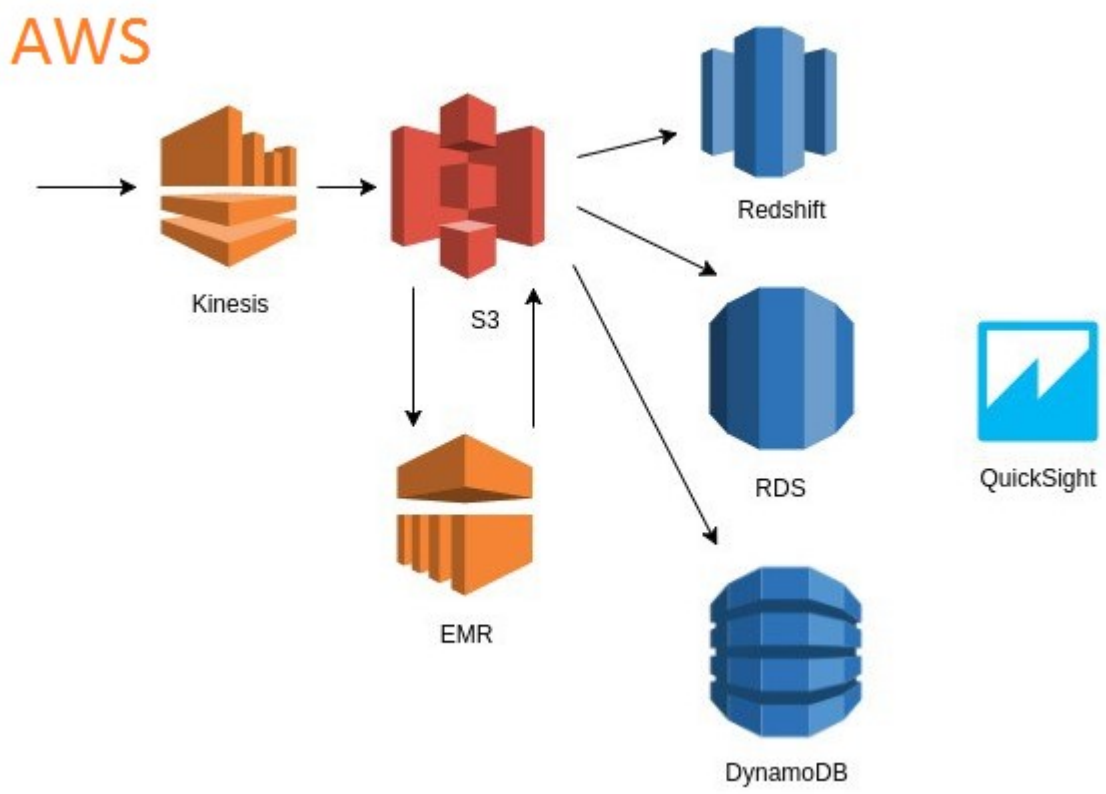


Fig. 12.5. Conceptual diagram related to streaming, storing, processing, and visualizing big data using AWS services

Fig. 12.5 illustrates the process of collecting, storing, processing, and analyzing big data using AWS services. The data flow begins with Amazon Kinesis, which is used to collect and process streaming data in real time.

The collected data is then transferred to Amazon S3 (Simple Storage Service), a scalable object store, where it can be stored for further processing and analysis. Once stored in S3, the data can be transferred to various analytical services. Amazon Redshift is used for analytical querying and working with large sets of structured data in the data warehouse. Amazon RDS (Relational Database Service) processes structured data in a relational database, while Amazon DynamoDB works with unstructured or semi-structured data in a high-performance NoSQL database.

To process large amounts of data, Amazon EMR (Elastic MapReduce) is used, a service for distributed data processing using Apache Spark, Hadoop, and other frameworks. To visualize the results of data analysis, Amazon QuickSight is used, which helps create reports, graphs, and analytical dashboards. This architecture allows efficiently process, analyze, and visualize large amounts of data in a scalable AWS environment.

Netflix, a global leader in video streaming, relies heavily on AWS Big Data services to analyze massive volumes of viewing data from over 200 million users worldwide. Using Amazon EMR and AWS Glue, Netflix processes logs and user behavior in near real-time to personalize content recommendations. Amazon S3 serves as its central data lake, storing petabytes of data that feed into machine learning models for improving streaming quality, preventing churn, and optimizing content delivery.

Airbnb uses the AWS Big Data ecosystem to manage and analyze data from millions of listings and user interactions. With Amazon Redshift and Amazon Athena, Airbnb can run complex queries on large datasets to extract outcomes on pricing trends, seasonal demand, and guest preferences to improve customer experience, and detect potential fraud patterns efficiently.

Samsung leverages AWS Big Data tools to manage data generated by its smart devices and IoT infrastructure, uses Amazon Kinesis for real-time data streaming and Amazon EMR to process and analyze this information at scale. This allows monitor device health, analyze usage patterns, and deliver smarter software updates and services based on live consumer data.

Pinterest utilizes AWS to scale its data processing pipelines that handle billions of pins and user interactions. Using Amazon S3 for storage, AWS Glue for ETL, and Amazon QuickSight for internal dashboards, Pinterest transforms massive datasets into actionable insights for product development and user engagement strategies.

To fully leverage AWS Big Data capabilities, companies typically use a suite of specialized software tools provided by AWS and integrated third-party technologies. Amazon EMR is widely adopted for large-scale data processing tasks using Apache Hadoop, Spark, Hive, and Presto. Amazon Redshift serves as a fast, fully managed data warehouse for running complex SQL analytics on structured data. For real-time data ingestion and streaming analytics, services like Amazon Kinesis and AWS Lambda are commonly used. Data transformation and preparation are often handled with AWS Glue, a serverless ETL service that simplifies building and maintaining data pipelines. Amazon Athena allows querying structured and semi-structured data stored in Amazon S3 using standard SQL without needing to manage infrastructure. For machine learning tasks, companies frequently use Amazon SageMaker alongside Big Data workflows to build, train, and deploy models at scale. Visualization and reporting are typically performed using Amazon

QuickSight, which enables the creation of interactive dashboards and charts directly connected to AWS data sources.

Control questions and tasks for topic 12

List of theoretical questions

1. What are the main advantages of using cloud providers for processing and analyzing large amounts of data?
2. Which cloud providers are among the leaders in the Big Data industry, and how do they compare in functionality and performance?
3. What are the key components that make up the AWS Big Data platform, and how do they interact with each other to process and analyze data?
4. How is AWS S3 used to store data in the AWS Big Data system, and how does it integrate with other services?
5. How does Amazon EMR help with different big data frameworks, such as Apache Hadoop and Apache Spark?
6. How does Amazon Redshift solve big data analytics problems, and how does it integrate with other AWS components?
7. How does AWS Glue help in the process of preparing and loading data for further analysis in the AWS Big Data system?
8. How does Amazon Kinesis work with streaming data, and how can it be used in analytics projects?
9. How does Amazon Athena allow execute SQL queries against data stored in Amazon S3, and how does it integrate with other services?
10. How does Amazon QuickSight facilitate data visualization and analytics in the AWS Big Data system, and what capabilities does it provide for users in creating dashboards and reports?

List of tasks for independent work

Practical task 1. Overview of Big Data cloud providers.

Choose three Big Data cloud providers: Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. Review the official documentation for each platform. Find examples of Big Data services (e.g., Amazon EMR, Google BigQuery, Azure Synapse Analytics). Review the main features of these services. Describe the key differences between the

providers in the form of a presentation or table. Use the free tier of one of the providers to run a simple Big Data project, such as loading data into a table and running a SQL query.

Practical task 2. Researching the AWS Big Data platform.

Sign up for AWS and log in to the console.

Go to the Amazon EMR (Elastic MapReduce) section.

Create an EMR cluster to process data using Apache Spark. Upload a test dataset (e.g., a CSV file) to Amazon S3. Write simple code in PySpark to analyze the data, for example, calculate the average value of a specific column:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Big Data Analysis").getOrCreate()
data = spark.read.csv("s3://your-bucket-name/your-file.csv", header=True, inferSchema=True)
data.groupBy("column_name").avg().show()
```

Run PySpark jobs on an EMR cluster. Save the results to S3 and view them through the AWS console. Write a report explaining how you configured the cluster and performed the analysis.

List of used and recommended literature

12.1. Amazon Web Services [Electronic resource]. – Access mode: https://aws.amazon.com/

12.1. AWS Big Data [Electronic resource]. – Access mode: https://www.infopulse.com/blog/aws-big-data-platform

12.2. Amazon EMR [Electronic resource]. – Access mode: https://aws.amazon.com/emr/

12.3. Microsoft Azure [Electronic resource]. – Access mode: https://azure.microsoft.com/

12.4. Google Cloud Platform [Electronic resource]. – Access mode: https://cloud.google.com/

12.5. IBM Cloud [Electronic resource]. – Access mode: https://www.ibm.com/cloud

12.6. Oracle Cloud [Electronic resource]. – Access link: https://www.oracle.com/cloud/

TERMINOLOGICAL INDEX

Liubov Oleshchenko is Candidate of Technical Sciences, Associate Professor, Associate Professor at the Department of Computer Systems Software, Faculty of Program Systems and Applied Mathematics, Igor Sikorsky Kyiv Polytechnic Institute.

Graduated with honors from T. Shevchenko Chernihiv State Pedagogical University, specialty "Mathematics and Informatics".

Defended Ph.D. thesis in specialty 05.13.06 "Information Technologies", which focused on the development of information technologies (mathematical models and software) for organizing intercity passengers transportation, as well as the development of software tools for the optimal placement of gas stations in a region using probabilistic gravitational models and GIS technologies.

Certified Cisco Networking Academy Instructor for CCNA programs. In 2023, received the prestigious Instructor Excellence Advanced Level Award in recognition of advanced-level teaching excellence and significant contributions to Cisco Networking Academy.

Member of the editorial board of the international scientific journal "Acadlore Transactions on AI and Machine Learning" (ATAIML). Member of the committee for the International Student Research Competition in Artificial Intelligence. Expert of the National Agency for Higher Education Quality Assurance and accreditation evaluator of educational programs.

Scientific supervisor and researcher of the scientific project "Methods for optimizing software performance and using artificial intelligence technologies to improve big data analytics systems," state registration number: 0124U001790 (2024–2026).

Author of over 100 scientific publications, 10 textbooks, and 30 certificates of authorship.

Scientific interests: mathematical modeling, programming, computer networks, IoT, big data analytics, machine learning, artificial intelligence.

Electronic online educational publication

Oleshchenko Liubov

BIG DATA AND ANALYTICS

Textbook

Proofreading by *N. V. Murashova*
Computer layout *by A. A. Mushnytsky*

Managing Editor: S.V. Syrota.

"ПРОСВІТА"
КИЇВ

Industrial and Commercial Company “Prosvita Ltd”
46 Taras Shevchenko Blvd., Kyiv, 01032, Ukraine
Tel.: +38 (067) 440 29 96
E-mail: prosvita.kyiv@gmail.com

Certificate DK No. 221 dated October 16, 2000

Approved for publication: April 8, 2026
Publisher’s sheets: 14.3

Electronic text data: (1 file: 9,64 MB)