НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Наталія АУШЕВА

«___»_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра

спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

на тему: «Обробка та передавання надвеликих масивів даних в режимі реального часу для елементів сенсорної мережі»

Виконав (-ла):
студент (-ка) IV курсу, групи ТМ-82
Демченко Олександр Едуардович _____

Керівник:
професор кафедри АПЕПС, д.т.н., доцент,
Федорова Наталія Володимирівна _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2022

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший

спеціальність 122 «Комп’ютерні науки»

освітня програма «Комп’ютерний моніторинг та геометричне моделювання процесів і систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Наталія АУШЕВА
(підпис)
”___”_____2022р.

ЗАВДАННЯ
на дипломну роботу студенту

Демченко Олександр Едуардович
(прізвище, ім’я, по батькові)

1. Тема роботи «Обробка та передавання надвеликих масивів даних в режимі реального часу для елементів сенсорної мережі»
керівник роботи Федорова Наталія Володимирівна, професор
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від ”__” травня 2022р. № _____

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи мова програмування Java, середовище розробки Intellij IDEA 2021, програмна платформа і каркас для організації розподіленого зберігання Apache Hadoop.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати поставлену задачу, проаналізувати існуючі системи і алгоритми обробки надвеликих масивів даних, проаналізувати способи передачі даних, розробити систему для обробки та передавання надвеликих масивів даних в режимі реального часу для елементів сенсорної мережі

5. Перелік ілюстративного матеріалу мета розробки, робота програмної системи, візуалізація алгоритмів і результатів роботи системи.
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”___”_____2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	**Затвердження теми роботи**		
2.	*Вивчення та аналіз задачі*		
3.	*Розробка архітектури та загальної структури системи*		
4.	*Розробка структур окремих підсистем*		
5.	*Програмна реалізація системи*		
6.	*Оформлення пояснювальної записки*		
7.	**Захист програмного продукту**		
8.	**Передзахист**		
9.	*Захист*		

Студент _____ _____Демченко О.Е._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Федорова Н.В._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Дипломну роботу розроблено на 63 аркушах, вона містить 38 рисунків, 1 таблицю та 13 бібліографічних найменувань за “Списком використаних джерел”. Також робота включає 2 додатки.

Метою даної дипломної роботи є розроблення системи для обробки та передавання надвеликих масивів даних в режимі реального часу для елементів сенсорної мережі.

Для реалізації системи було обрано програмне забезпечення у вигляді веб додатку, яке є зручним та зрозумілим для користувачів

Ключові слова: Big Data, обробка, передавання надвеликі масиви даних.

ABSTRACT

The thesis is developed on 63 sheets, it contains 38 figures, 1 table and 13 bibliographic titles on the "List of used sources". The work also includes 2 applications.

The aim of this thesis is to develop a system for processing and transmitting large data sets in real time for elements of the sensor network.

To implement the system, we chose software in the form of a web application that is convenient and understandable for users

Keywords: Big Data, processing, transmission of ultra-large data sets.

ЗМІСТ

ВСТУП

Сучасний світ стрімко розвивається: за останнє десятиліття кількість даних, які генеруються і відправляються через мережу стрімко зростає. Про це свідчить і статистика від аналітиків компанії IBS, яка провела дослідження й оцінила об'єм світових даних. Їх кількість до 2003 року становила до 5 ексабайтів даних (1 ексабайт ≈ 1 млрд гігабайтів) у 2008 році — 0,18 зеттабайта (1 зеттабайт ≈ 1024 ексабайта), така ж кількість даних вироблялась кожні 10 днів у 2015 році, і щодня у 2020 році. Ці дані оточують нас і в явному чи неявному вигляді впливають на життя багатьох людей. Зі стрімким зростанням обсягів даних з'явилась потреба у ефективній обробці, аналізу, інтерпретації результатів для забезпечення максимальних результатів у відповідних задачах [1].

Великі компанії розробляють нові технології, інструменти, алгоритми, архітектурні рішення для ефективної обробки і аналізу надвеликих масивів даних. Наприклад, такі компанії як Google, Microsoft, Facebook, Amazon розробляють власні фреймворки, алгоритми та методи для вирішення власних задач у області надвеликих даних. Тому припущення щодо того що «дані — це нова нафта» є як ніколи актуальним насьогодні і підкреслює важливість надвеликих даних в сучасному світі.

Надвеликі обсяги даних прийнято називати Big Data — це великий, різноманітний набір інформації, який зростає з постійними темпами і не може бути належним чином оброблений традиційними додатками через свій величезний обсяг. Складність аналізу надвеликих масивів даних полягає у специфіці їх збору, поділу, зберігання, передачі, візуалізації та збереження конфіденційності інформації. Великі дані часто надходять від аналізу даних і надходять у кількох форматах. Під обробкою надвеликих даних часто розуміється застосування прогнозної аналітики чи інших передових алгоритмів та методів з метою вилучення з множини даних корисної інформації [1].

Той, хто володіє правильною формою даних, вміє їх коректно обробити і представити, може успішно орієнтуватися на ринку, робити прогнози на майбутні події та підлаштовуватись до ринкових тенденцій. Більша частина даних, які генеруються користувачами мережі на сьогоднішній день є неструктурованими або напівструктурованими, а це означає, що вони мають різні форми, розміри і навіть зв'язки. Отже, обробляти, аналізувати та візуалізувати такі дані не є простою і звичайною задачею. Ця задача є великою проблемою для більшості компаній.

Існує безліч різних сервісів для обробки даних в режимі реального часу, але всі вони мають одне з головних обмежень – розмір набору даних. Так якщо деякі просто не сприймають великорозмірні набори даних, то інші готові це зробити проте не безкоштовно. Яскравим прикладом таких сервісів є ElasticSearch, Apache Solr та Algolia. Всі вони чудово справляються з задачами обробки коли мова йде про відносно невеликі набори даних, в інших випадках сервіс або відмовляється приймати дані для обробки, або пропонує зробити це купивши тижневу підписку.

Виходячи з цього було прийнято рішення створити власне програмне забезпечення для обробки надвеликих наборів даних та передавання їх з метою подальшого аналізу чи візуалізації. Для цього було проаналізовано найпопулярніші алгоритми та засоби обробки та передавання надвеликих масивів даних.

1 ЗАДАЧА ОБРОБКИ ТА ПЕРЕДАВАННЯ НАДВЕЛИКИХ МАСИВІВ ДАНИХ

Метою роботи є розробка системи для ефективної обробки надвеликих масивів даних з сенсорних мереж, що є легкою в розгортанні та масштабуванні, в режимі реального часу для подальшої аналітики та візуалізації даних.

Обробка включає в себе послідовність маніпуляцій з даними за допомогою відповідних алгоритмів з метою отримання «нових» даних, інформації або для виконання різних обчислень.

При виконанні роботи були визначені наступні задачі:

− створити систему, яка буде обробляти надвеликі масиви даних, в режимі реального часу для елементів сенсорної мережі, такі як температура, тиск, вологість та зберігати їх із можливістю подальшого передавання.

Для вирішення цієї комплексної задачі була виконана декомпозиція і визначені, сформовані наступні підзадачі:

– провести аналіз існуючих систем, визначити їх переваги і недоліки;

– провести аналіз алгоритмів для аналізу надвеликих обсягів даних;

– розробити архітектуру системи для обробки та передавання надвеликих масивів даних;

– реалізувати алгоритми для обробки;

– розробити легку в розгортанні і масштабуванні систему.

Вхідними даними системи є метеорологічні дані, які зберігаються на сервері.

Вихідними даними системи є агреговані дані, які мають зміст для кінцевого користувача. Тобто аналітик може з легкістю їх інтерпретувати і використати для своїх цілей.

Для програмного продукту були сформовані наступні вимоги:

– декомпозиція модулів системи на мікросервіси для малої зв'язності компонентів;

– підвищена відмовостійкість системи при великих навантаженнях;

– легке масштабування та розгортання системи в різних середовищах;

Опрацювання задачі за темою роботи містить наступні основні питання, за якими потрібно розділити роботу над темами: зберігання надвеликого об’єму даних для подальшого доступу та взаємодії, обробка даних в залежності від виду аналізу та передавання отриманих результатів для подальшого поширення та взаємодії з обробленими даними.

1.1 Зберігання даних

Існує три способи зберігання цифрових даних [2]:

– Традиційний – на дисках, стрічках, локальних сховищах тощо;

– У публічних «хмарах»: від таких гігантів, як Amazon, Microsoft і Google;

– У приватних «хмарах»: характерний для корпоративного сегмента.

Розберемо деякі плюси та мінуси цих підходів.

Традиційний спосіб. Такий підхід є найбільш звичним для більшості з нас. Інформація записується на локальні сховища – диски, RAID масиви, стрічки та інші.

Плюси:

– Це звично. Дані завжди поруч, і нам так спокійніше.

– Швидкість доступу. Як правило, до локального носія можна легко та швидко підключитися.

– Ціна. Хоча вона може бути мінусом.

Мінуси:

– Ненадійність. Диски та сервери виходять з ладу внаслідок фізичного зносу.

– Доступ до даних.

– Масштабування. Потрібно купувати нові носії та десь їх розміщувати.

Публічні хмари. Цей підхід надає можливість зберігати дані у хмарі за певну плату, яка залежить від обсягу даних та супутніх послуг.

Плюси:

– Це зручно. Підприємства максимально спрощують базові сценарії роботи.

– Це безпечно. Більшість вендорів захищає дані не тільки паролем користувача, а й власними алгоритмами шифрування.

– Це надійно. Є можливість реплікування даних.

– Нові горизонти у майбутньому (швидкий та безпечний обмін даними).

Мінуси:

1. Психологічний фактор Ваші дані далекі від вас;

2. Ціна. Хмарне зберігання дорожче за локальне.

3. Швидкість доступу. Все-таки швидкість доступу в інтернет навіть у передових країнах у середньому вимірюється мегабайтами в секунду (що, як мінімум, у десятки разів повільніше за доступ до локальних сховищ) [2].

Приватні «хмари» багато в чому схожі на публічні і при використанні корпоративного середовища можуть давати відчуття більшого контролю над безпекою даних.

Дані можна записувати та зберігати в трьох основних формах: сховище файлів, блочне сховище та сховище об’єктів.

1. Сховище файлів

Сховище файлів, також зване сховище на рівні файлів або на основі файлів, є ієрархічною методологією зберігання, яка використовується для організації та зберігання даних. Іншими словами, дані зберігаються у файлах, файли організовані в папки, а папки організовані за ієрархією каталогів і підкаталогів.

2. Блочне зберігання

Блочне зберігання, яке іноді називають сховищем на рівні блоків, є технологією, яка використовується для зберігання даних у блоках. Потім блоки зберігаються як окремі частини, кожна з яких має унікальний ідентифікатор. Розробники віддають перевагу блочним сховищам для обчислювальних ситуацій, які вимагають швидкої, ефективної та надійної передачі даних.

3. Сховище об'єктів

Об’єктне сховище, яке часто називають об’єктним сховищем, є архітектурою зберігання даних для обробки великих обсягів неструктурованих даних. Ці дані не відповідають традиційній реляційній базі даних із рядками та стовпцями або їх не

можна легко організувати. Приклади включають електронну пошту, відео, фотографії, веб-сторінки, аудіофайли, дані датчиків та інші типи медіа та веб-вмісту (текстові чи нетекстові).

Для зберігання даних, незалежно від форми, користувачам потрібні пристрої зберігання даних. Пристрої зберігання даних поділяються на дві основні категорії: безпосереднє зберігання даних і мережеве сховище.

Пряме сховище, також відоме як сховище з прямим підключенням (DAS), це, як випливає з назви. Це сховище часто знаходиться в безпосередній близькості та безпосередньо підключено до комп'ютерної машини, яка отримує до нього доступ. Часто це єдина машина, підключена до неї. DAS також може надавати гідні локальні послуги резервного копіювання, але спільне використання обмежено. Пристрої DAS включають дискети, оптичні диски — компакт-диски (CD) і цифрові відеодиски (DVD) — жорсткі диски (HDD), флеш-накопичувачі та твердотільні накопичувачі (SSD).

Мережне сховище дає змогу більш ніж одному комп'ютеру отримувати доступ до нього через мережу, що робить його кращим для обміну даними та спільної роботи. Його можливості зберігання поза межами сайту також роблять його кращим для резервного копіювання та захисту даних. Дві поширені налаштування мережевого сховища – це мережеве сховище (NAS) і мережа зберігання даних (SAN).

NAS часто є єдиним пристроєм, що складається з резервних контейнерів для зберігання даних або резервного масиву незалежних дисків (RAID). Сховище SAN може бути мережею з кількох пристроїв різних типів, включаючи SSD та флеш-сховище, гібридне сховище, гібридне хмарне сховище, програмне забезпечення та пристрої резервного копіювання та хмарне сховище. Ось чим відрізняються NAS і SAN:

NAS:

- Один накопичувач або RAI
- Система зберігання файлів
- Мережа Ethernet TCP/IP
- Обмежені користувачі

– Обмежена швидкість
– Обмежені можливості розширення
– Нижча вартість і легке налаштування

SAN:

– Мережа з кількох пристроїв
– Блокова система зберігання
– Мережа Fibre Channel
– Оптимізовано для кількох користувачів
– Швидша продуктивність
– Дуже розширюваний
– Вища вартість і складне налаштування

1.2 Обробка даних

Обробка Big Data – це набір методів або моделей програмування для доступу до великомасштабних даних для отримання корисної інформації для підтримки та надання рішень. На сьогоднішній день виділяють 3 типи: структуровані, неструктуровані та напівструктуровані дані.

Структуровані дані складаються з чітко визначених типів даних із шаблонами, які дозволяють легко знайти їх. Такі дані зазвичай знаходяться в реляційних базах даних (RDBMS). У полях зберігаються дані з роздільною довжиною, як-от номери телефонів, номери соціального страхування або поштові індекси. Дані можуть бути створені людиною або машиною, якщо дані створюються в структурі СУБД [3]. Цей формат надзвичайно зручний для пошуку, як із запитами, створеними людьми, так і за допомогою алгоритмів із використанням типів даних та назв полів, таких як алфавітний або числовий, валюта чи дата.

Неструктуровані дані – «все інше» – складаються з даних, які зазвичай не так легко знайти, включаючи такі формати, як аудіо, відео та публікації в соціальних мережах. Неструктуровані дані мають внутрішню структуру, але не структуровані за

допомогою попередньо визначених моделей даних або схеми. Він може бути текстовим або нетекстовим, створеним людиною або машиною. Він також може зберігатися в нереляційній базі даних, як-от NoSQL.

Типові неструктуровані дані, створені людиною, включають:

– Текстові файли: електронні таблиці, презентації, електронні листи, журнали.
– Соціальні мережі: дані з Facebook, Twitter, LinkedIn.
– Мобільні дані: текстові повідомлення, місцезнаходження.
– Комунікації: чат, миттєві повідомлення, телефонні записи.
– Медіа: MP3, цифрові фотографії, аудіо та відео файли.
– Бізнес-додатки: документи MS Office.

Типові машинно-генеровані неструктуровані дані включають:

– Супутникові зображення: дані про погоду, форми рельєфу.
– Наукові дані: освоєння космосу, сейсмічні знімки, атмосферні дані.
– Цифрове спостереження: фото та відео спостереження.
– Дані датчиків: рух, погода, океанографічні датчики.

Напівструктуровані дані – дані, які містять внутрішні теги та позначки, які ідентифікують окремі елементи даних, що дає змогу аналітикам даних визначати групування та ієрархію інформації. Як документи, так і бази даних можуть бути напівструктурованими. Цей тип даних становить лише близько 5-10% кола даних, але має критичні випадки використання в бізнесі, якщо використовується в поєднанні зі структурованими та неструктурованими даними.

Приклади напівструктурованих даних:

– Мова розмітки XML Це напівструктурована мова документа. XML – це набір правил кодування документів, які визначають формат, читається людиною та машиною. Його цінність полягає в тому, що його структура, керована тегами, дуже гнучка, і кодери можуть адаптувати його для універсалізації структури даних, зберігання та транспортування в Інтернеті.

– Відкритий стандарт JSON – це ще один напівструктурований формат обміну даними. Його структура складається з пар ім’я/значення (або об’єкт, хеш-таблиця тощо) і впорядкований список значень (або масив, послідовність, список). Оскільки

структура є взаємозамінною між мовами, JSON чудово передає дані між веб-додатками та серверами.

– Напівструктуровані дані також є важливим елементом багатьох баз даних NoSQL, які відрізняються від реляційних, оскільки не відокремлюють організацію (схему) від даних. Це робить NoSQL кращим вибором для зберігання інформації, яка не легко вписується у формат запису та таблиці, наприклад текст різної довжини[3].

Існує три основні методи обробки даних – ручний, механічний та електронний.

1. Ручна обробка даних

У цьому методі обробки даних дані обробляються вручну. Весь процес збору даних, фільтрації, сортування, обчислення та інших логічних операцій виконується з людським втручанням без використання будь-яких інших електронних пристроїв або програмного забезпечення для автоматизації. Це недорогий метод і не вимагає практично ніяких інструментів, але дає великі похибки, високі витрати праці та багато часу.

2. Механічна обробка даних

Дані обробляються механічно за допомогою пристроїв і машин. Це можуть бути прості пристрої, такі як калькулятори, друкарські машинки, друкарський верстат тощо. За допомогою цього методу можна виконати прості операції з обробки даних. Він має набагато менше помилок, ніж ручна обробка даних, але збільшення даних робить цей метод більш складним і важким.

3. Електронна обробка даних

Дані обробляються за сучасними технологіями з використанням програмного забезпечення та програм обробки даних. Набір інструкцій дається програмному забезпеченню для обробки даних і отримання результату. Цей метод є найдорожчим, але забезпечує найшвидшу швидкість обробки з найвищою надійністю та точністю виводу.

Приклади обробки даних

Обробка даних відбувається в нашому повсякденному житті незалежно від того, усвідомлюємо ми це чи ні. Ось кілька реальних прикладів обробки даних:

– Програмне забезпечення для торгівлі акціями, яке перетворює мільйони даних про біржі в простий графік

– Компанія електронної комерції використовує історію пошуку клієнтів, щоб рекомендувати подібні продукти

– Компанія з цифрового маркетингу використовує демографічні дані людей, щоб розробити стратегію кампаній для певного місця

– Самокерований автомобіль використовує дані в режимі реального часу від датчиків, щоб визначити, чи є на дорозі пішоходи та інші автомобілі

1.3 Передавання даних

Передавання даних — це процес використання обчислювальних методів і технологій передачі або передачі електронних чи аналогових даних з одного комп'ютерного вузла в інший. Дані передаються у вигляді бітів і байтів по цифровому або аналоговому середовищі, і цей процес забезпечує цифровий або аналоговий зв'язок та їхнє переміщення між пристроями.

Передавання файлів стосується обміну файлами даних між комп'ютерними системами. Це дозволяє спільно використовувати, передавати або передавати файл або логічний об'єкт даних між різними користувачами та комп'ютерами як локально, так і віддалено». Файли даних можуть бути структурованими або неструктурованими, включаючи документи, мультимедіа, графіку та PDF-файли [4].

Фактори, що впливають на сьогоднішні вимоги до передачі файлів:

– Обсяги даних. Вимоги до робочого навантаження для передачі файлів, як правило, пов'язані з більш частою пакетною обробкою та більшими та різноманітними файлами, ніж у минулому.

– Надвеликі дані та IoT: підприємства впроваджують технологію передачі файлів, щоб забезпечити обмін файлами транзакцій у таких областях, як Інтернет речей (IoT) та аналітика надвеликих даних. Цей подвійний удар по швидкості та обсягу файлу створює особливу проблему для технологій передачі файлів.

– Безпека: проблеми кібербезпеки продовжують зростати, що призводить до впровадження досконаліших технологій безпеки. Системи передачі файлів, де це можливо, повинні компенсувати накладні витрати на безпеку за рахунок підтримки апаратних прискорювачів, нового програмного забезпечення для процесів безпеки та підвищення пропускної спроможності під час передачі файлів [4].

Висновки до розділу 1

В даному розділі була описана мета розробки та ціль. При виконанні роботи була визначена задача, проведена декомпозиція її на підзадачі і наведено можливості для їх вирішення.

Описано вхідні, вихідні дані, а також сформовані вимоги для розробки програмного забезпечення. Роботу розділено на три основні теми це зберігання, обробка та передавання надвеликих масивів даних в результаті чого кожну з тем було детально розглянуто й проаналізовано для вирішення поставленого завдання.

Крім того представлене детальне пояснення виконуваної роботи.

2 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ ТА АЛГОРИТМІВ ОБРОБКИ НАДВЕЛИКИХ МАСИВІВ ДАНИХ

Системи для обробки надвеликих наборів даних — це вираз, який було створено для позначення обсягів наборів даних, які настільки величезні, що звичайне програмне забезпечення для підготовки даних не може їх контролювати.

На даний момент існує безліч неймовірних систем, інструментів і технологій для обробки надвеликих даних. Кожен із них є надзвичайними в тому, що вони роблять. Проте в кожного є свої переваги й недоліки в залежності від поставленої задачі або алгоритму обробки [5].

Розглянемо популярні системи та їх алгоритми для обробки.

2.1 Apache Hadoop

По-перше, це постійна класика і один з найпопулярніших фреймворків, які використовуються сьогодні. Він настільки поширений, що майже став синонімом надвеликих даних. Apache Hadoop — інфраструктура, яка полегшує роботу з кластерами (рисунок 2.1).

Рисунок 2.1 — Логотип Apache Hadoop

Основні елементи Hadoop — це: розширена файлова система (HDFS), YARN, Hadoop Common та алгоритм крупномасштабного виконання програми (MapReduce).

Розподілена файлова система Hadoop (HDFS) – розподілена файлова система, яка працює на стандартному або бюджетному обладнанні. HDFS забезпечує кращу пропускну здатність даних, ніж традиційні файлові системи, на додаток до високої відмовостійкості та вбудованої підтримки великих наборів даних.

Yet Another Resource Negotiator (YARN) – керує та контролює вузли кластера та використання ресурсів. Він розкладає роботи та завдання.

MapReduce – фреймворк, який допомагає програмам виконувати паралельні обчислення даних. Завдання карти приймає вхідні дані та перетворює їх у набір даних, який можна обчислити в парах ключових значень. Вихідні дані завдання карти споживаються завданнями скорочення, щоб узагальнити вихідні дані та забезпечити бажаний результат.

Hadoop Common – надає загальні бібліотеки Java, які можна використовувати в усіх модулях.

Переваги Apache Hadoop:

– Більшого обсягу пам’яті та обчислювальної потужності можна досягти, додавши більше вузлів до кластера Hadoop. Це позбавляє від необхідності купувати зовнішнє обладнання. Тому це дешевше рішення.

– Він може обробляти неструктуровані та напівструктуровані дані.

– Кластери Hadoop забезпечують зберігання та розподілені обчислення в одному.

– Фреймворк Hadoop має вбудовану потужність і гнучкість, щоб робити те, що раніше було неможливо.

– Шар HDFS у hadoop має характеристики самовідновлення, реплікації та відмовостійкості. Він автоматично реплікує дані, якщо сервер або диск зазнають збою.

– Hadoop пропонує масштабованість, надійність і велику кількість бібліотек для різних додатків за меншу вартість. Це допомагає розподілити дані на різних серверах і запобігає перевантаженню мережі.

Недоліки Apache Hadoop:

– Не підходить для невеликих за обсягом даних.

– Об’єднання кількох операцій із набором даних є складним.

– У нього немає шифрування на рівні пам'яті або мережі.

– Управління кластером є складним, тобто в кластері такі операції, як налагодження, розповсюдження програмного забезпечення, журнали збору тощо, є занадто важкими [5].

Переваги алгоритму обробки MapReduce:

– Модель проста і легко програмується.

– Він просто реалізує деякі інтерфейси для завершення розподіленої програми, яку можна розповсюдити на велику кількість дешевих комп’ютерів для виконання. Іншими словами, написання розподіленої програми точно так само, як написання простої послідовної програми. Саме завдяки цій функції програмування Mapreduce стало дуже популярним.

– Хороша масштабованість.

– Коли ваші обчислювальні ресурси не задоволені, ви можете розширити його обчислювальну потужність, просто додавши машини.

– Гнучкий

– структуровані та неструктуровані дані

– Паралельна обробка

– Модель програмування, природно, підтримує паралельну обробку, що підходить для автономної обробки PB-рівня та великих обсягів даних

– Сильна відмовостійкість

– Початкова мета Mapreduce полягає в тому, щоб дозволити розгортати програми на дешевих комп’ютерах, що вимагає високої відмовостійкості. Наприклад, якщо машина зависла, вона може передати вищезазначені обчислювальні завдання на інший вузол для виконання, щоб завдання не зазнало невдачі, і цей процес не вимагає участі людини, а повністю виконується внутрішньо за допомогою hadoop.

Недоліки MapReduce

– Поганий в потокових обчисленнях. Вхідні дані потокових обчислень є динамічними, у той час як набір вхідних даних Mapreduce є статичним і не може змінити свій потік. Це з тим, що характеристики самого Mapreduce визначають, що джерело даних може бути статичним.

– Не підходить для DAG (направленого графа) обчислення залежностей декількох програм, вхід останнього додатка є виходом попереднього додатка, в цьому випадку Mapreduce не неможливо зробити, але кожен Mapreduce після використання завдання буде записаний на диск, що призведе до великої кількості дискових операцій введення-виводу, що призведе до дуже низької продуктивності .

2.2 Apache Spark

Spark є спадкоємцем царства обробки надвеликих даних. Spark і Hadoop часто протиставляються як вибір «або/або», але насправді це не так [6]. Екосистема Hadoop може вмістити двигун обробки Spark замість MapReduce, що призводить до різноманітних змін середовища, які можуть включати поєднання інструментів і технологій з обох екосистем.

Spark - інфраструктура кластерних обчислень, подібна до Hadoop MapReduce (рисунок 2.2).

Рисунок 2.2 — Логотип Spark

Однак Spark не займається ні зберіганням файлів у файловій системі, ні керуванням ресурсами. Spark обробляє дані за допомогою вбудованих колекцій RDD (Resilient Distributed Datasets), які дозволяють виконувати обчислення у великих кластерах. Завдяки RDD можна здійснювати такі операції, як map, join, reduce, записувати дані на диск та завантажувати їх [6].

Стійкі розподілені набори даних (RDD) є основною структурою даних Spark. Це незмінна розподілена колекція об'єктів. Кожен набір даних у RDD розділений на логічні розділи, які можуть обчислюватися на різних вузлах кластера. RDD можуть містити будь-який тип об’єктів Python, Java або Scala, включаючи визначені користувачем класи.

Формально RDD — це розділена колекція записів, доступна лише для читання. RDD можна створювати за допомогою детермінованих операцій над даними у стабільному сховищі або іншими RDD. RDD — це відмовостійка колекція елементів, з якими можна працювати паралельно.

Існує два способи створення RDD – розпаралелювання наявної колекції у вашій програмі драйвера або посилання на набір даних у зовнішній системі зберігання, наприклад спільна файлова система, HDFS, HBase або будь-яке джерело даних, що пропонує формат введення Hadoop.

Spark використовує концепцію RDD для досягнення швидших та ефективних операцій MapReduce. Спільний доступ до даних у MapReduce повільний через реплікацію, серіалізацію та дискове введення. Більшість програм Hadoop витрачають більше 90% часу на виконання операцій читання-запису HDFS.

Визнаючи цю проблему, дослідники розробили спеціалізований фреймворк під назвою Apache Spark. Ключовою ідеєю Spark є стійкі розподілені набори даних (RDD); він підтримує обчислення обробки в пам'яті. Це означає, що він зберігає стан пам’яті як об’єкт для всіх завдань, і об’єкт можна спільно використовувати між цими завданнями. Обмін даними в пам’яті відбувається в 10-100 разів швидше, ніж мережа та диск, проте, насправді, в реальності це 3-5 разів.

Переваги Apache Spark:

– Швидкість

– Простота використання

– Розширена аналітика

– Багатомовний

– Розширений доступ до надвеликих даних

– Спільнота з відкритим кодом

Недоліки Apache Spark:

– Немає автоматичного процесу оптимізації

– Система управління файлами

– Менше алгоритмів

– Проблема з невеликими файлами

– Не підходить для багатокористувацького середовища

Переваги RDD:

– Ліниве оцінювання – дані всередині RDD не оцінюються, доки не буде ініційовано дію для обчислення.

– Обчислення в пам'яті – дані всередині RDD зберігаються в пам'яті, а не на диску, таким чином, продуктивність збільшується в багато разів.

– Незмінність – після створення RDD його не можна змінити. Будь-яке перетворення на ньому створить новий RDD.

– Відмовостійкість – якщо робочий вузол виходить з ладу, використовуючи Lineage (відстеження всіх перетворень, які необхідно застосувати до цього RDD, включно з тим, звідки він повинен читати дані), ми можемо повторно обчислити втрачений розділ RDD з оригінальний.

– Розбиття на розділи – RDD поділяються на менші блоки, які називаються розділами (логічні фрагменти даних), коли виконуються деякі дії, завдання запускається на розділ. Кількість розділів безпосередньо відповідає за паралельність.

Недоліки RDD:

– Код RDD іноді може бути дуже непрозорим. Розробникам може бути важко з’ясувати, що саме намагається обчислити код.

– Spark не може оптимізувати RDD, оскільки Spark не може заглядати всередину лямбда-функцій і оптимізувати операції. У деяких випадках, коли filter()

передається після широкого перетворення, Spark ніколи не виконуватиме фільтр першим перед широким перетворенням, наприклад, reduceByKey() або groupByKey().

2.3 Apache Storm

Apache Storm — це розподілена обчислювальна система в реальному часі, додатки якої розроблені як орієнтовані ациклічні графи (рисунок 2.3) [7].

Рисунок 2.3 — Логотип Apache Storm

Storm призначений для легкої обробки необмежених потоків і може використовуватися з будь-якою мовою програмування. Він був протестований на обробку більше одного мільйона кортежів в секунду на вузол, має високу масштабованість і забезпечує гарантії виконання завдань обробки. Унікальний для елементів у цьому списку, Storm написаний на Clojure, мові програмування, подібній до Lisp.

Apache Storm можна використовувати для аналітики в реальному часі, розподіленого машинного навчання та багатьох інших випадків, особливо з високою швидкістю даних. Storm може працювати на YARN та інтегруватися в екосистеми Hadoop, забезпечуючи існуючі реалізації рішення для обробки потоків у реальному часі.

Apache Storm інтегрується з технологіями черг і баз даних, які ви вже використовуєте. Топологія Apache Storm споживає потоки даних і обробляє ці потоки

довільно складними способами, перерозподіляючи потоки між кожним етапом обчислення, наскільки це необхідно.

Переваги Apache Storm:

– Доступність. Apache Storm є відкритим кодом і безкоштовним у використанні, що робить його доступним рішенням як для малого, так і для великого бізнесу.

– Гнучкість. Apache Storm забезпечує гнучкість завдяки інтеграції в будь-яку мову програмування.

– Масштабованість. Система має високу масштабованість і додає додаткові ресурси лінійно в міру збільшення завантаження даних.

– Гарантія на обробку даних. Розподілена система забезпечує доставку даних у разі простою вузла.

Недоліки Apache Storm:

– Складно встановити та налаштувати для розгортання. Система інтегрується з різними іншими технологіями. Створити такі зв’язки між Storm та іншими додатками іноді важко.

– Немає підтримки на рівні рамки. Розробка проекту починається з нуля, що ускладнює розробку нових розробників.

– Не підходить для менших наборів даних. Apache Storm є розподіленою системою і не є хорошим вибором для невеликих програм [7].

Висновки до розділу 2

В даному розділі було розглянуто інформацію про наявні системи та алгоритми для обробки надвеликих масивів даних. Для кожного з інструмента та метода описано основні переваги та недоліки, в результаті чого можна зрозуміти основні відмінності між ними та моменти для кращого їх використання.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

При розробці програмного забезпечення розглянуто безліч технологій, мов програмування та алгоритмів які підходять для обробки даних. Під час оцінки було виявлено технології, які забезпечують найбільшу гнучкість, швидкість роботи, а також простоту в використанні з можливістю в майбутньому розширити та вдосконалити систему.

3.1 Розподілена файлова система HDFS

Для збереження даних було прийнято рішення використовувати HDFS. Розподілена файлова система Hadoop (HDFS) — це відмовостійка файлова система зберігання даних, що працює на звичайному обладнанні. Вона була розроблена для вирішення проблем, які не вдалося вирішити традиційним базам даних. Тому весь його потенціал використовується тільки при роботі з надвеликими даними (рисунок 3.1) [8].

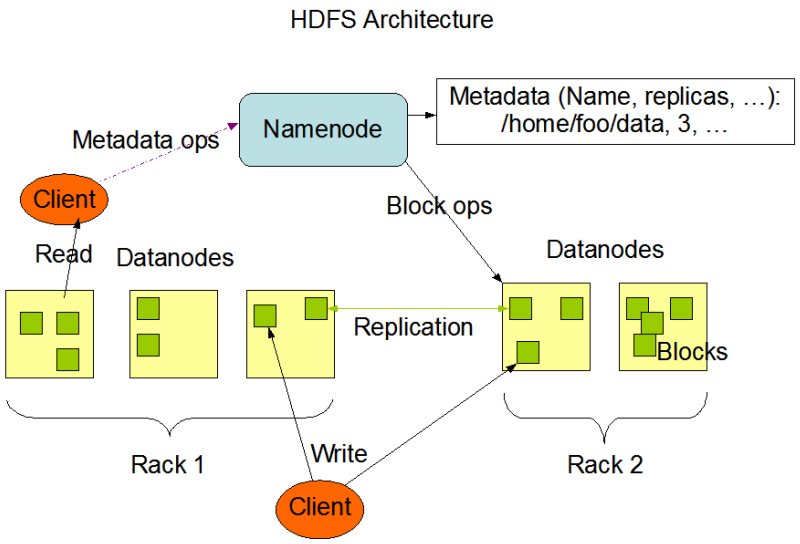

Рисунок 3.1 — Архітектура HDFS

HDFS ділить файли на блоки та зберігає кожен блок у DataNode. Декілька вузлів даних пов'язані з головним вузлом у кластері NameNode. Головний вузол розподіляє репліки цих блоків даних у всьому кластері. Він також вказує користувачеві, де знайти потрібну інформацію. Однак, перш ніж NameNode зможе допомогти вам зберігати дані та керувати ними, спочатку необхідно розбити файл на дрібніші керовані блоки даних. Цей процес називається поділом блоку даних.

Цілі й мета HDFS:

1. Апаратний збій. Збій апаратного забезпечення більше не є винятком; це стало звичайним терміном. Примірник HDFS складається із сотень або тисяч серверних машин, кожна з яких зберігає частину даних файлової системи. Існує величезна кількість компонентів, які дуже чутливі до апаратних збоїв. Це означає, що є деякі компоненти, які завжди не працюють. Отже, основна архітектурна мета HDFS — швидке й автоматичне виявлення/відновлення несправностей.

2. Потоковий доступ до даних. Програми HDFS потребують потокового доступу до своїх наборів даних. Hadoop HDFS в основному призначений для пакетної обробки, а не для інтерактивного використання користувачами. Сила полягає у високій пропускній здатності доступу до даних, а не в низькій затримці доступу до даних. Він зосереджений на тому, як отримати дані з максимально швидкою швидкістю під час аналізу журналів.

3. Великі набори даних. HDFS працює з великими наборами даних. У стандартній практиці файл у HDFS має розмір від гігабайт до петабайт. Архітектура HDFS повинна бути розроблена таким чином, щоб вона найкраще підходила для зберігання та отримання величезних обсягів даних. HDFS має забезпечувати високу пропускну здатність сукупних даних і мати можливість масштабувати до сотень вузлів в одному кластері. Крім того, він повинен бути достатньо хорошим, щоб працювати з тоннами мільйонів файлів на одному екземплярі.

4. Проста модель когерентності. Він працює на основі теорії моделі доступу до файлів «запис-один-читання-багато». Після створення, запису та закриття файлу його не слід змінювати. Це вирішує проблеми узгодженості даних і забезпечує високу пропускну здатність доступу до даних. Програма на основі MapReduce або програма

веб-сканера ідеально вписується в цю модель. Відповідно до приміток apache, існує план підтримки додавання записів до файлів у майбутньому.

5. Переміщення обчислень дешевше, ніж переміщення даних. Якщо програма виконує обчислення поблизу даних, з якими вона працює, це набагато ефективніше, ніж далеко. Цей факт стає сильнішим під час роботи з великим набором даних. Головною перевагою цього є збільшення загальної пропускної здатності системи. Це також мінімізує перевантаження мережі. Припущення полягає в тому, що краще перемістити обчислення ближче до даних, ніж переміщувати дані до обчислень.

6. Переносність між різнорідними апаратними та програмними платформами. HDFS розроблено з властивістю портативності, щоб його можна було переносити з однієї платформи на іншу. Це забезпечує широке поширення HDFS. Це найкраща платформа для роботи з великим набором даних.

За замовчуванням розмір блоку не може перевищувати 128 Мб. Кількість блоків залежить від розміру файлу. Всі блоки, крім останнього, мають однаковий розмір (128 МБ), а останній це те, що залишилося від файлу (рисунок 3.2) [8].

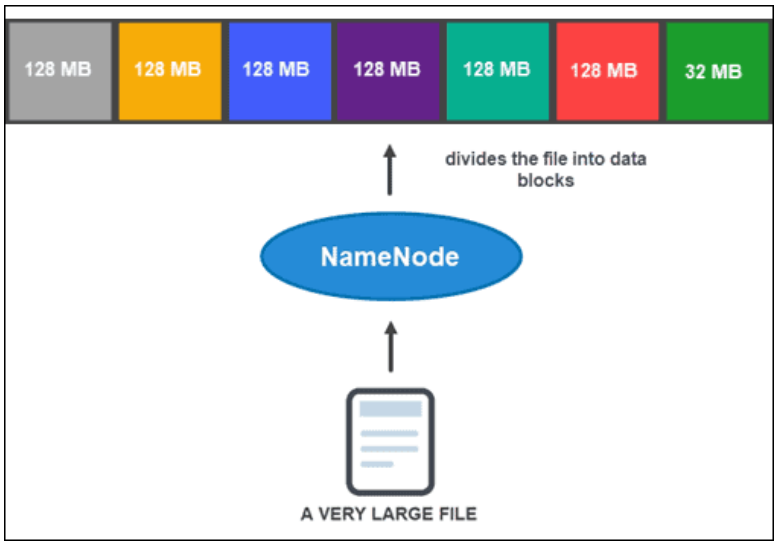

Рисунок 3.2 — Приклад поділу файла на блоки в HDFS

Наприклад, файл розміром 800 МБ розбитий на сім блоків даних. Шість із семи блоків мають розмір 128 МБ, а сьомий блок даних – решта 32 МБ.

3.2 Модель обробки даних MapReduce

MapReduce – це модель розподілених обчислень від компанії Google, яка використовується в технологіях Big Data для паралельних обчислень над дуже великими (до кількох петабайт) наборами даних у комп'ютерних кластерах, та фреймворк для обчислення розподілених завдань на вузлах (node) кластеру.

Hadoop може запускати програми MapReduce, написані різними мовами: Java, Ruby, Python та C++. Програми Map Reduce у хмарних обчисленнях мають паралельний характер, тому дуже корисні для виконання великомасштабного аналізу даних з використанням кількох машин у кластері [9].

Тепер давайте подивимося, як працює Hadoop MapReduce, детально зрозумівши наскрізний потік виконання завдання Hadoop MapReduce з компонентами:

1. Вхідні файли

Дані для завдання MapReduce зберігаються у вхідних файлах, а вхідні файли зазвичай зберігаються у HDFS. Формат цих файлів довільний, у той час як файли журналу на основі рядків і двійковий формат також можна використовувати.

2. InputFormat

Тепер InputFormat визначає, як ці вхідні файли розділяються та читаються. Він вибирає файли або інші об'єкти, які використовуються для введення. InputFormat створює InputSplit.

3. InputSplits

Він створюється за допомогою InputFormat, логічно представляє дані, які будуть оброблятися окремим Mapper (Ми розберемося з Mapper нижче). Для кожного розбиття створюється одне завдання на карту; таким чином кількість завдань карти буде дорівнювати кількості InputSplits. Розбиття розділено на записи, і кожен запис буде оброблено картографом.

4. RecordReader

Він взаємодіє з InputSplit в Hadoop MapReduce і перетворює дані в пари ключ-значення, придатні для читання картографом. За замовчуванням він використовує TextInputFormat для перетворення даних у пару ключ-значення. RecordReader

взаємодіє з InputSplit, доки читання файлу не буде завершено. Він призначає байтове зміщення (унікальний номер) кожному рядку у файлі. Далі, ці пари ключ-значення відправляються в картограф для подальшої обробки.

5. Mapper.

Він обробляє кожен вхідний запис (з RecordReader) і генерує нову пару ключ-значення, і ця пара ключ-значення, згенерована Mapper, повністю відрізняється від вхідної пари. Вихід Mapper також відомий як проміжний вихід, який записується на локальний диск. Вихідні дані Mapper не зберігаються в HDFS, оскільки це тимчасові дані, і запис у HDFS створить непотрібні копії (також HDFS є системою з високою затримкою). Вихід картографів передається в комбінатор для подальшої обробки

6. Combiner.

Combiner також відомий як «міні-редуктор». Hadoop MapReduce Combiner виконує локальну агрегацію на виході картографів, що допомагає звести до мінімуму передачу даних між картографом і редуктором (нижче ми побачимо редуктор). Після виконання функціональних можливостей комбінатора вихідні дані передаються в роздільник для подальшої роботи.

7. Partitioner

Hadoop MapReduce, Partitioner з’являється у випадку, якщо ми працюємо над більш ніж одним редуктором (для одного редуктора розділення не використовується).

Partitioner отримує вихідні дані з комбінаторів і виконує розбиття. Розбиття виводу відбувається на основі ключа, а потім сортується. За допомогою хеш-функції ключ (або підмножина ключа) використовується для отримання розділу.

Відповідно до значення ключа в MapReduce, кожен вихідний об’єднувач розділяється, і запис, що має те саме значення ключа, потрапляє в той самий розділ, а потім кожен розділ надсилається в редуктор. Розбиття дозволяє рівномірно розподілити вихід карти по редуктору. Вивчіть MapReduce Partitioner детально.

8. Shuffling and Sorting

Тепер вихід перемішується до вузла зменшення (який є звичайним підпорядкованим вузлом, але фаза зменшення буде виконуватися тут, тому його називають вузлом редуктора). Перемішування — це фізичне переміщення даних, яке

здійснюється по мережі. Після того, як усі картографи завершені та їх вихідні дані перемішуються на вузлах редукторів, цей проміжний вихід об’єднується та сортується, який потім надається як вхід для зменшення фази.

9. Reducer

Він бере набір проміжних пар ключ-значення, створених картографами, як вхідні дані, а потім запускає функцію зменшення для кожного з них, щоб генерувати вихідні дані. Вихід редуктора є кінцевим виходом, який зберігається в HDFS. Перейдіть за цим посиланням, щоб дізнатися більше про Reducer.

10. RecordWriter

Він записує ці вихідні пари ключ-значення з фази Reducer у вихідні файли.

11. OutputFormat

Спосіб запису цих вихідних пар ключ-значення у вихідні файли за допомогою RecordWriter визначається OutputFormat. Екземпляри OutputFormat, надані Hadoop, використовуються для запису файлів у HDFS або на локальний диск. Таким чином, кінцевий вихід редуктора записується в HDFS екземплярами OutputFormat.

Отже, таким чином Hadoop MapReduce працює над кластером. На закінчення можна сказати, що потік даних у MapReduce є комбінацією різних фаз обробки, таких як вхідні файли, формат введення в Hadoop, InputSplits, RecordReader, Mapper, Combiner, Partitioner, Shuffling and Sorting, Reducer, RecordWriter і OutputFormat. Тому всі ці компоненти відіграють важливу роль у роботі Hadoop MapReduce.

Приклад роботи алгоритму MapReduce зображено на рисунку 3.3.

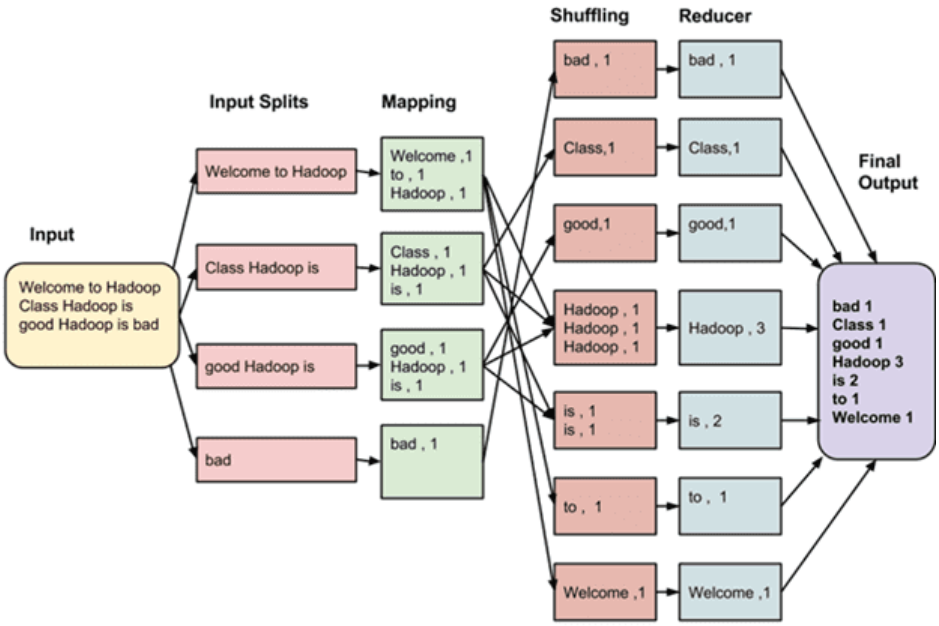

Рисунок 3.3 — Алгоритм роботи MapReduce

Метод MapReduce передбачає обробку даних у 3 етапи[10].

1. Map – вхідні дані спочатку поділяються на дрібніші блоки. Потім кожен блок призначається для перетворювача обробки. Результат записується у форматі «ключ-значення» у тимчасове сховище.

2. Shuffle - фаза споживає вихідні дані етапу Map. ЇЇ завдання – об'єднати відповідні записи з вихідних даних попередньої фази, таким чином, щоб усі дані одного ключа лежали в одному робочому вузлі.

3. Reduce – на цьому етапі збираються вихідні значення етапу Shuffle. Підбивається підсумок всього набору даних.

3.3 Мова програмування Java

Java — це універсальна, заснована на класах, об'єктно-орієнтована мова програмування, розроблена для менших залежностей реалізації. Це обчислювальна платформа для розробки додатків. Тому Java швидка, безпечна та надійна. Вона широко використовується для розробки Java-додатків у ноутбуках, центрах обробки

даних, ігрових консолях, наукових суперкомп'ютерах, мобільних телефонах тощо [11].

На початку 90-х Java, що спочатку носила назву Oak, а потім Green, була створена командою під керівництвом Джеймса Гослінга для Sun Microsystems, компанії, яка зараз належить Oracle.

Спочатку Java було розроблено для використання на цифрових мобільних пристроях, таких як мобільні телефони. Однак, коли Java 1.0 була випущена для широкої публіки в 1996, її основна увага змістилася на використання в Інтернеті, забезпечуючи інтерактивність з користувачами, даючи розробникам можливість створювати анімовані веб-сторінки.

Однак після версії 1.0 було випущено багато оновлень, таких як J2SE 1.3 у 2000 р., J2SE 5.0 у 2004 р., Java SE 8 у 2014 р. та Java SE 10 у 2018 р.

З роками Java перетворилася на успішну мову для використання як в Інтернеті, так і за її межами.

Чому вибирають Java?

Java був розроблений з урахуванням кількох ключових принципів:

– Простота використання . Основи Java прийшли із мови програмування C++. Хоча C++ — сильна мова, його синтаксис складний і відповідає деяким вимогам Java. Java побудувала і вдосконалила ідеї C++, щоб надати мову програмування, яка була потужною і простою у використанні.

– Надійність: Java потрібна для зниження ймовірності фатальних помилок через помилки програміста. Маючи це на увазі, було введено об'єктно-орієнтоване програмування. Коли дані та їх обробка були запаковані разом в одному місці, Java була надійною.

– Безпека: Оскільки Java спочатку призначалася для мобільних пристроїв, які будуть обмінюватися даними через мережу, вона була створена з урахуванням високого рівня безпеки. Java, мабуть, найбезпечніша мова програмування на сьогоднішній день.

– Незалежність від платформи: програми повинні працювати незалежно від того, на яких машинах вони виконуються. Java була написана як переносна і

кросплатформова мова, яка не дбає про операційну систему, обладнання або пристрої, на яких він працює.

Команді Sun Microsystems вдалося об'єднати ці ключові принципи, і популярність Java можна прослідкувати завдяки тому, що це надійна, безпечна, проста у використанні та переносима мова програмування.

3.4 Платформа Spring Framework

Spring Framework (Spring) — це програма з відкритим вихідним кодом, яка забезпечує підтримку інфраструктури для розробки додатків Java. Одна з найпопулярніших фреймворків Java Enterprise Edition (Java EE), Spring допомагає розробникам створювати високопродуктивні програми за допомогою простих старих об’єктів Java (POJO).

Spring Framework складається з функцій, організованих приблизно 20 модулів. Ці модулі згруповані в основний контейнер, доступ до даних/інтеграцію, Інтернет, АОП (аспектно-орієнтоване програмування), інструментування та тестування, як показано на рисунку 3.4.

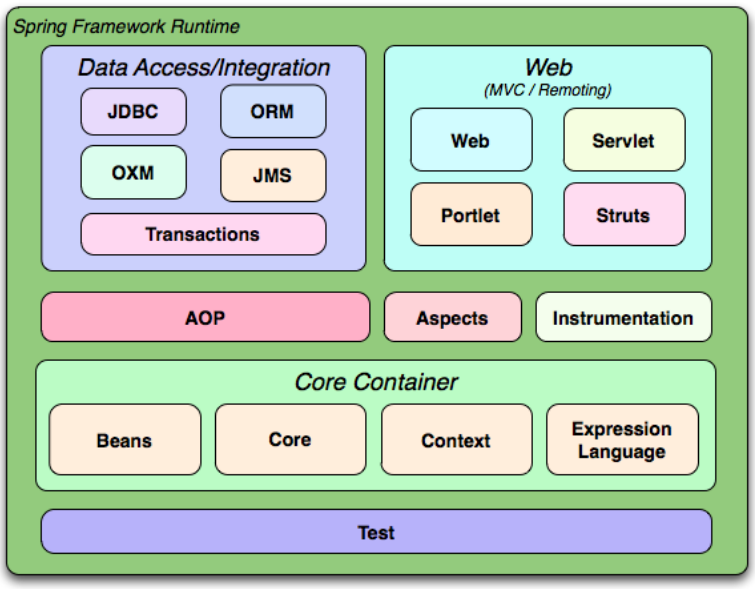

Рисунок 3.4 — Модулі Spring Framework

Spring вважається надійним, недорогим і гнучким каркасом. Spring покращує ефективність кодування та скорочує загальний час розробки додатків, оскільки він ефективний у використанні системних ресурсів і має велику підтримку. Spring усуває стомлюючу роботу з налаштування, щоб розробники могли зосередитися на написанні бізнес логіки.

Веб-додаток (шарова архітектура) Spring зазвичай включає три рівні:

– Рівень презентації/перегляду (UI) – це крайній рівень, який обробляє презентацію вмісту та взаємодії з користувачем.

– Рівень бізнес-логіки - центральний рівень, який займається логікою програми.

– Рівень доступу до даних – глибинний рівень, який займається отриманням даних із джерел.

Кожен шар залежить від іншого для роботи програми. Іншими словами, рівень презентації спілкується з рівнем бізнес-логіки, який спілкується з рівнем доступу до даних. Залежність – це те, що потрібно кожному шару для виконання своєї функції. Типова програма має тисячі класів і багато залежностей. Без Spring Framework код програми, як правило, тісно пов’язаний (взаємозалежний), що не вважається хорошою практикою програмування. Нещільне зчеплення ідеально, тому що нещільно пов’язані компоненти є незалежними, тобто зміни в одному не вплинуть на роботу інших [12].

3.5 Бібліотека Thymeleaf

Для реалізації інтерфейсу користувача використано бібліотеку Thymeleaf — це бібліотека на основі Java, використовувана для створення веб-додатків. Він забезпечує хорошу підтримку обслуговування XHTML/HTML5 у веб-прикладах (рисунок 3.5) [12].

Мета Thymeleaf — надати стильний і добре сформований спосіб створення шаблонів. Він заснований на тегах і атрибутах XML. Ці теги XML визначають

виконання попередньо визначеної логіки в DOM (об’єктна модель документа) замість явного запису цієї логіки як коду всередині шаблону. Це заміна JSP.

Архітектура Thymeleaf дозволяє швидку обробку шаблонів, що залежить від кешування розібраних файлів. Він використовує найменшу можливу кількість операцій введення-виводу під час виконання.

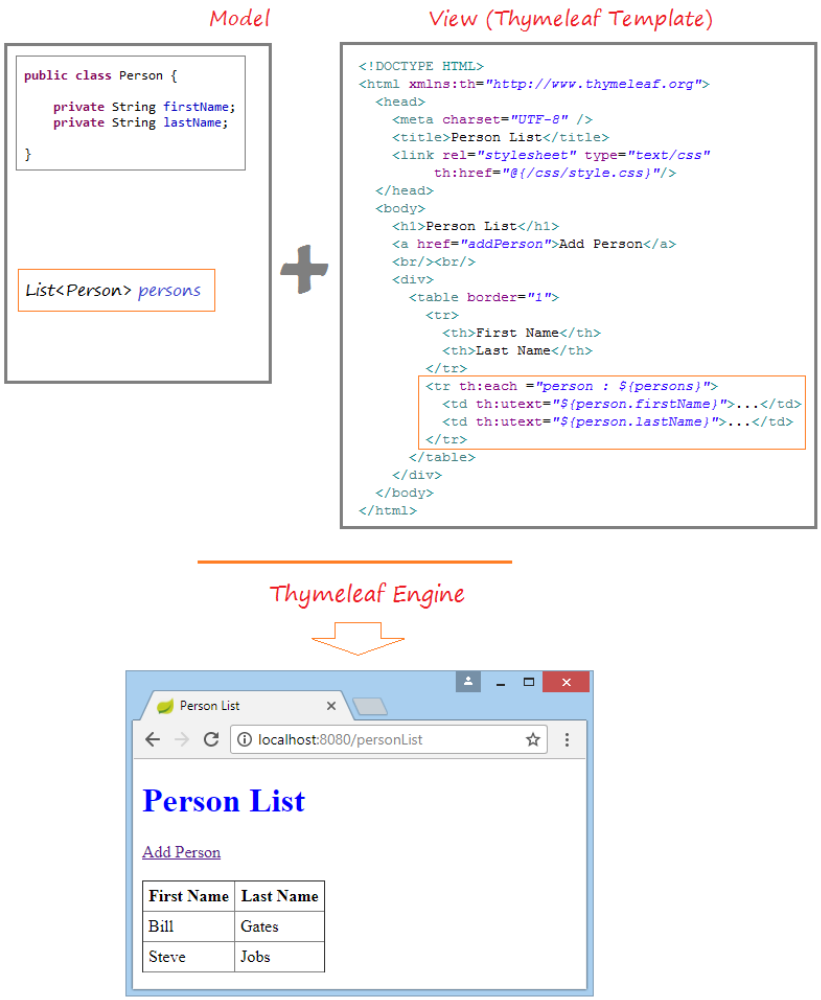

Рисунок 3.5 — Приклад результату взаємодії Thymeleaf з Java кодом

Thymeleaf Engine (Машина Thymeleaf) прочитає шаблонний файл (template file) і комбінує з об'єктами Java, щоб генерувати інший документ

Висновки до розділу 3

В даному розділі було розглянуто інформацію про засоби розробки. Проведено огляд розподіленої файлової системи HDFS і алгоритму обробки MapReduce, які ляжуть в основу для обробки надвеликих масивів даних.

Проведено ознайомлення з мовою програмування Java, фреймворком Spring та бібліотекою для візуалізації користувацького інтерфейсу Thymeleaf, що буде використовуватись під час розробки програмного продукту.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Важливою складовою виконання будь-якої роботи є розробка алгоритму та обміркований поетапний процес реалізації. Тож в цьому розділі описуються необхідні інструменти, компоненти, залежності та багато іншого для створення програмного продукту.

Процес створення системи обробки та передавання надвеликих масивів даних передбачає насамперед три основних етапи: створення і налаштування розподіленого файлового сховища, яке передбачає розвертання сервера Hadoop, реалізація алгоритму обробки даних на основі моделі MapReduce та передача отриманих результатів для подальшого аналізу чи візуалізації. Хоча програмний продукт, який бачитиме користувач, реалізується в основному у другому і третьому етапі, що містить значно більшу частину програмного коду, перший етап є не менш важливий. Дана частина роботи визначається не громісткістю, а складністю, тому перший етап за часом роботи не менший за два інших.

4.1 Встановлення Hadoop і налаштування HDFS

Для початку роботи перш за все необхідно щоб на комп’ютері був встановлений пакет Java 8. Після установки можна перевірити це за допомогою командного рядка командою javac -version (рисунок 4.1).

Далі інсталюємо Hadoop сервер на свій комп’ютер. Для коректної роботи і взаємодії сервера необхідно налаштувати шляхи змінних, для того щоб система могла взаємодіяти між модулями Java та Hadoop (рисунок 4.2).

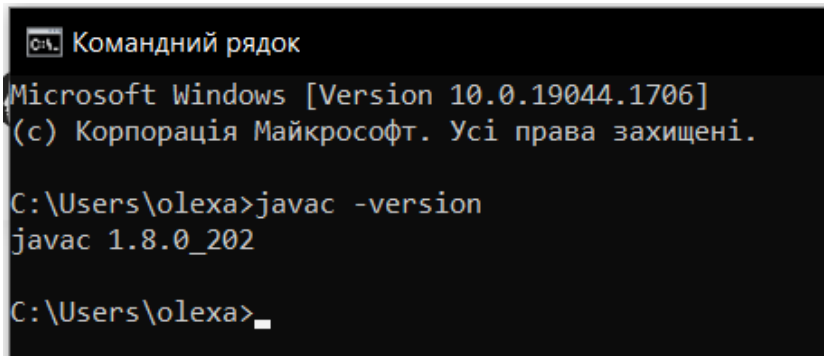

Рисунок 4.1 — Перевірка наявності java на комп’ютері

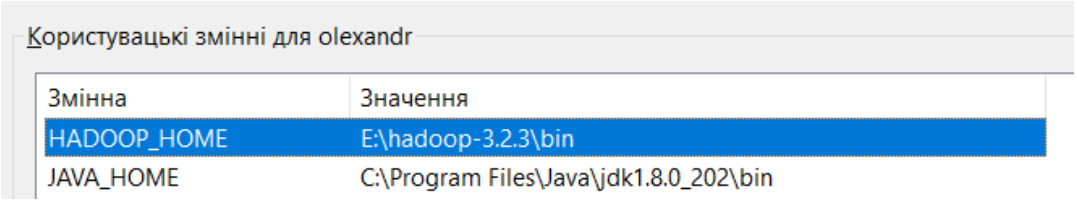

Рисунок 4.2 — Встановлення шляху до модулів Hadoop і Java

Потім потрібно налаштувати властивості сервера для його коректної роботи. Перш за все, у конфігураційному файлі core-site.xml встановимо наступні значення де задаємо порт локального сервера (рисунок 4.3).

По-друге, необхідно сконфігурувати файл hdfs-site.xml, де ми встановлюємо шляхи до компонентів namenode – відповідає за відкриття та закриття файлів, створення та видалення каталогів, управління доступом з боку зовнішніх клієнтів та відповідність між файлами та блоками, дубльованими (реплікованими) на вузлах даних та datanode - відповідає за запис та читання даних, виконання команд від вузла NameNode зі створення, видалення та реплікації блоків, а також періодичне відправлення повідомлення про стан (heartbeats) та обробку запитів на читання та запис, що надходять від клієнтів файлової системи (рисунок 4.4).

core-site.xml: Блокнот

Файл Редагування Формат Вигляд Довідка

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
 <property>
      <name>fs.defaultFS</name>
      <value>hdfs://localhost:9000</value>
   </property>
</configuration>
```

Рисунок 4.3 — Конфігурація файла core-site.xml

hdfs-site.xml: Блокнот

Файл Редагування Формат Вигляд Довідка

```
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
   <name>dfs.namenode.name.dir</name>
   <value>/hadoop-3.2.3/data/namenode</value>
</property>
<property>
   <name>dfs.datanode.data.dir</name>
   <value>/hadoop-3.2.3/data/datanode</value>
</property>
<property>
  <name>dfs.permissions.enabled</name>
  <value>false</value>
</property>
</configuration>
```

Рисунок 4.4 — Конфігурація файла hdfs-site.xml

Далі конфігуруємо файл yarn-site.xml. Yarn – набір системних програм (демонів-потоків), що забезпечують спільне використання, масштабування та надійність роботи розподілених додатків (рисунок 4.5).

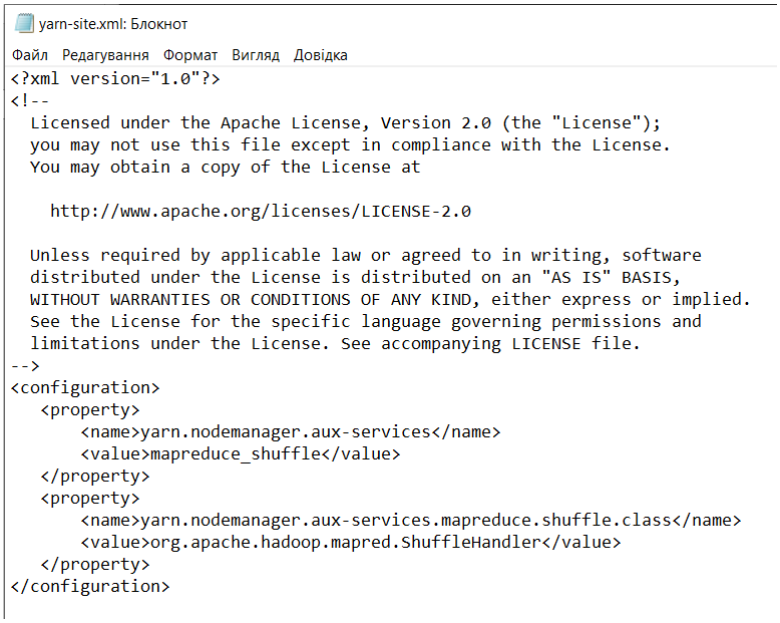

Рисунок 4.5 — Конфігурація файла yarn-site.xml

Після налаштування всіх конфігураційних файлів ми можемо протестувати роботу hdfs. Запускаємо namenode та datanode за допомогою команди start-dfs, запускаємо yarn за допомогою команди start-yarn (рисунок 4.6).

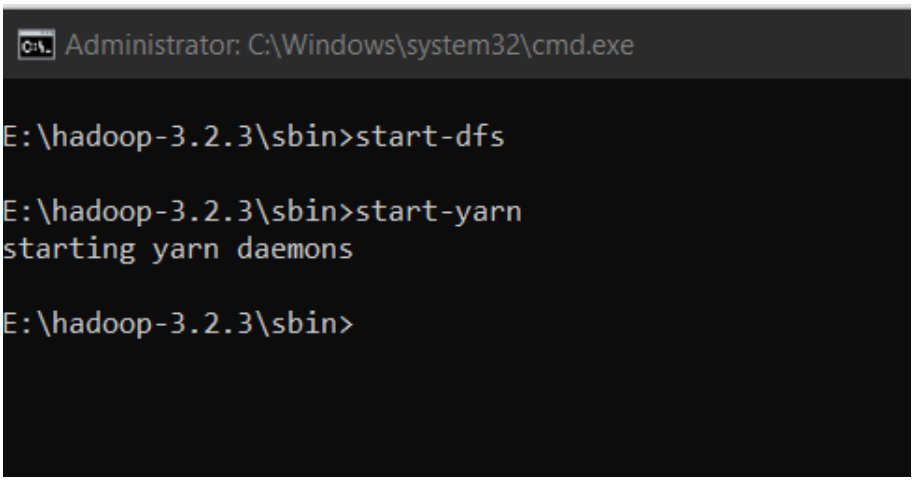

Рисунок 4.6 — Запуск сервера

Після запуску ми можемо спостерігати 4 вікна з сервісними логами, де в разі помилки ми отримаємо відповідне повідомлення. Адміністрування системи відбувається за допомогою спеціальних команд Hadoop (рисунок 4.7).

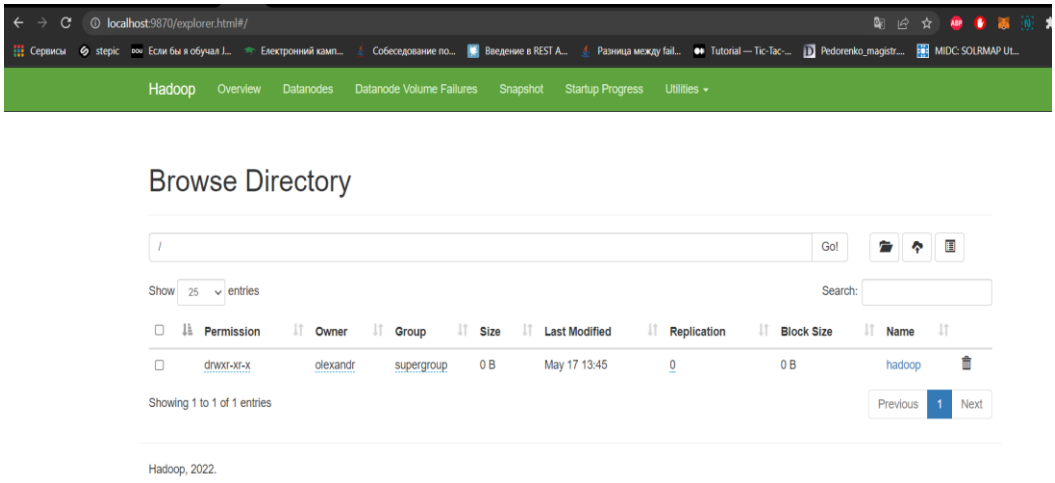

Рисунок 4.7 — Адміністрування hdfs

Можливий варіант використання hdfs через веб інтерфейс, необхідно просто перейти за відповідним посиланням в браузері, яке ми задавали при конфігурації сервера.

4.2 Реалізація алгоритму обробки надвеликих масивів даних

Для організації архітектури зв’язків між класами та написання бізнес логіки було обрано шаблон проектування MVC.

MVC — це скоріше архітектурний шаблон, але не для повного застосування. MVC здебільшого стосується інтерфейсу користувача або рівня взаємодії програми. Модель містить лише чисті дані програми, вона не містить логіки, яка описує, як представити дані користувачеві.

– Представлення надає користувачеві дані моделі. Представлення розуміє, яким чином отримати доступ до даних моделі, але воно не знає, що ці дані означають або що користувач може зробити, щоб маніпулювати ними.

– Контролер існує між виглядом і моделлю. Він прослуховує події, викликані представленням і в залежності від події реагує на неї. Здебільшого викликається якийсь метод який попередньо зазначений на конкретне подію. Оскільки представлення та модель пов’язані через механізм сповіщень, результат цієї дії автоматично відображається у представленні.

Реалізація алгоритму обробки даних буде відбуватись на основі моделі MapReduce. Для обробки за кожним критерієм ми повинні створити клас драйвер. Він відповідає за налаштування завдання MapReduce для запуску Hadoop. Ми вказуємо назви класів Mapper і Reducer з типами даних і відповідними іменами завдань. Для нашої системи зробимо можливість знаходити середнє значення температури, тиску та точки роси за певний період часу: день, тиждень, місяць. Схема роботи такої системи зображено на рисунку 4.8.

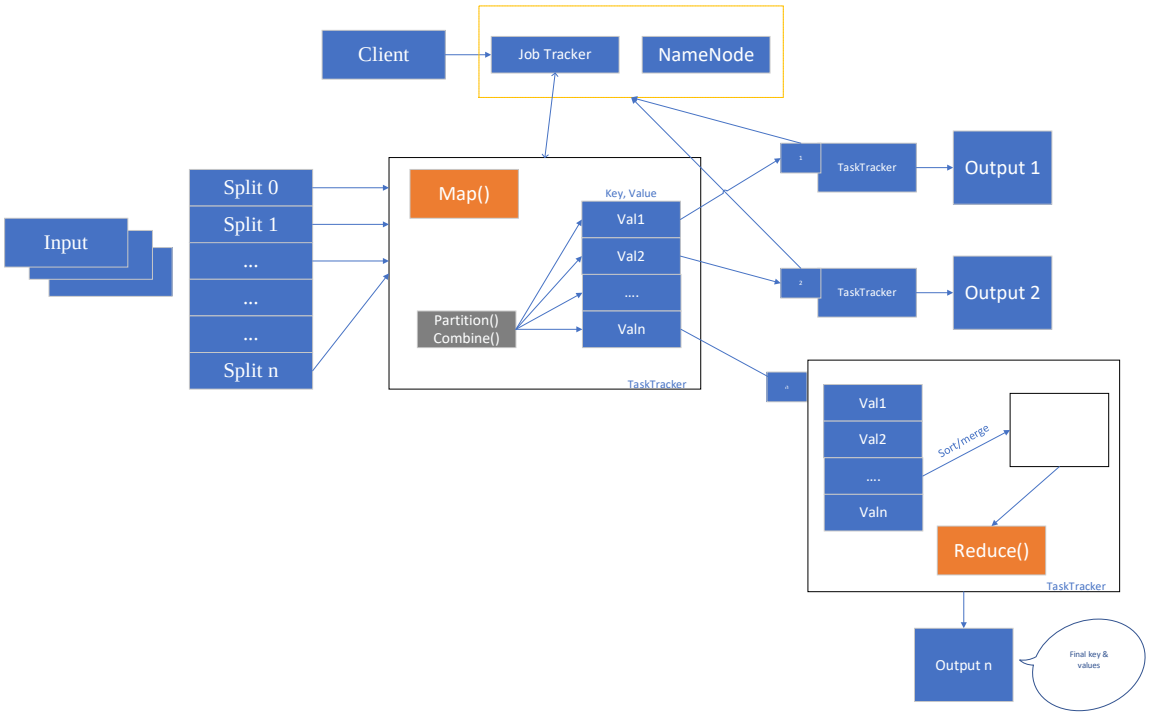

Рисунок 4.8 — Схема роботи програми в залежності від вибору користувача

Приклад методу map() класу TempDayMapper, який отримує вхідні дані в режимі реального часу та передає їх для обробки в парі ключ-значення для подальшої обробки зображено на рисунку 4.9.

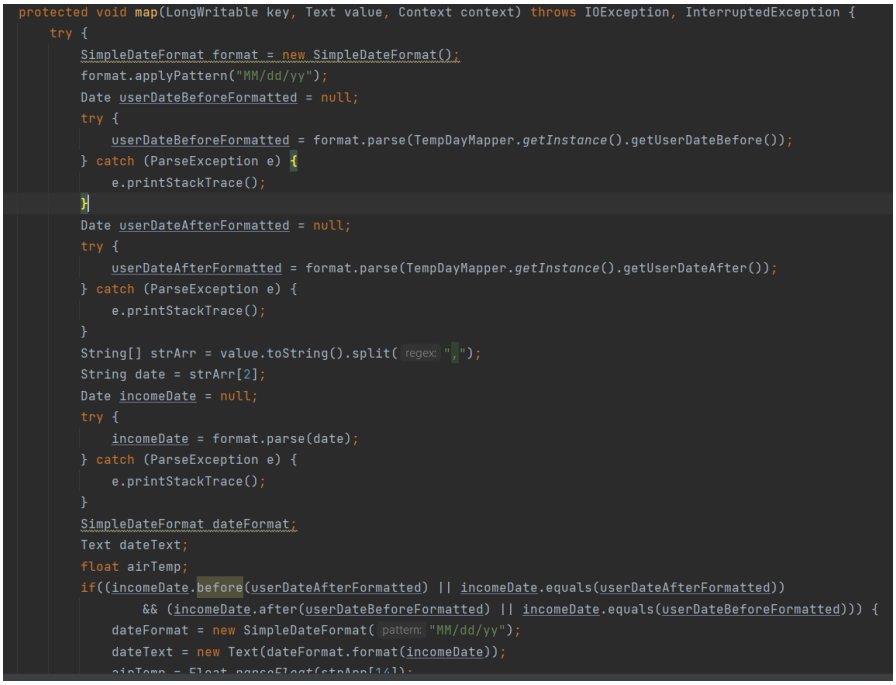

Рисунок 4.9 — Метод map() класу TempDayMapper

Приклад методу reduce() класу TempDayReducer, який отримує дані від метода map() класу TempDayMapper та обробляє їх в залежності від заданої умови зображено на рисунку 4.10.

```java
protected void reduce(Text key, Iterable<FloatWritable> floatTypes, Context context)
        throws IOException, InterruptedException {
    counter = 0;
    sum = 0;
    for (FloatWritable f : floatTypes) {
        sum += f.get();
        counter++;
    }
    hm.put(key.toString(), sum / counter);
    Optional<Map.Entry<String, Double>> first = hm.entrySet().stream().findFirst();
    String date = first.get().getKey();
    YearMonth yearMonthObject = YearMonth.of(Integer.parseInt(date.substring(6, 10)), Integer.parseInt(date.substring(0, 2)));
    int daysInMonth = yearMonthObject.lengthOfMonth();
    if (hm.size() == daysInMonth) {
        ArrayList<Double> listOfValues = new ArrayList<>(hm.values());
        double res = listOfValues.stream().mapToDouble(value -> value).average().orElse( other: 0);
        String str = yearMonthObject.toString().replaceAll( regex: "-", replacement: "/");
        context.write(new Text( string: "Month: " + str + " avg_temp: " + res), nullWritable);
        hm.clear();
    }
}
```

Рисунок 4.10 — Метод reduce() класу TempDayReducer

Діаграма класів для модуля обробки температури за день зображена на рисунку 4.11.

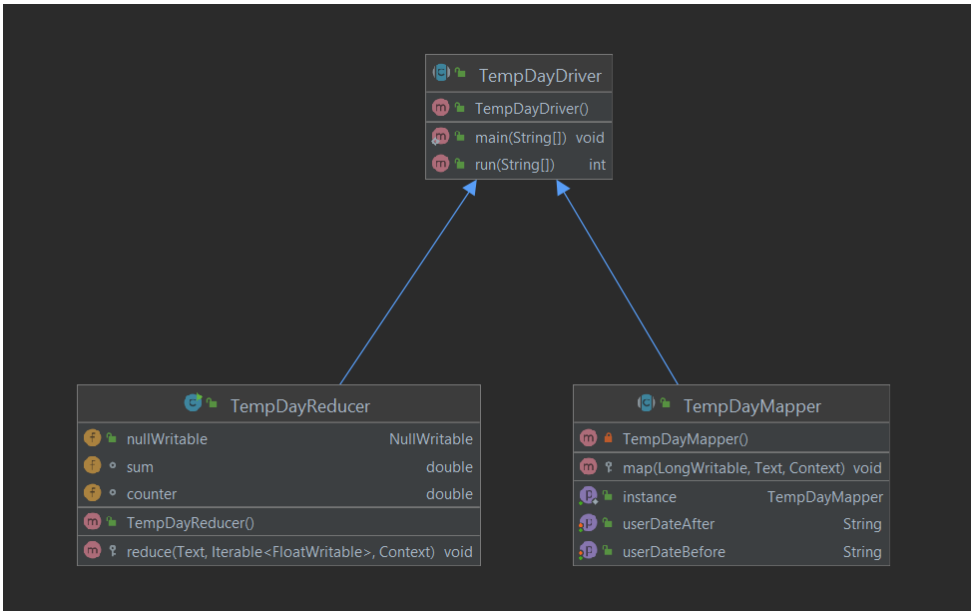

Рисунок 4.11 — Діаграма класів модуля обробки температури за день

4.3 Передавання оброблених даних

Для передавання файлів з результатами обробки з метою їх подальшого аналізу, візуалізації чи простого поширення було прийнято рішення передавати цей файл на Google Drive. Причиною такого рішення є ряд переваг сховища:

– Можливість доступу до файлів звідусіль.
– Можливість редагування файлів.
– Сумісність з більшістю пристроїв.
– Швидкий пошук файлів.
– Можливість перегляду файлів різних типів.
– Легкий обмін.
– Відкрите обговорення.
– Безкоштовне сховище до 15 ГБ.

Тому було прийнято рішення використовувати Google Cloud API для цієї задачі. Перш за все ми додали залежності до нашого проекту (рисунок 4.12).

```
<dependency>
    <groupId>com.google.apis</groupId>
    <artifactId>google-api-services-drive</artifactId>
    <version>v3-rev197-1.25.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.google.oauth-client/google-oauth-client-jetty -->
<dependency>
    <groupId>com.google.oauth-client</groupId>
    <artifactId>google-oauth-client-jetty</artifactId>
    <version>1.33.3</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.google.api-client/google-api-client -->
<dependency>
    <groupId>com.google.api-client</groupId>
    <artifactId>google-api-client</artifactId>
    <version>1.34.0</version>
</dependency>
```

Рисунок 4.12 — Залежності Google Cloud API

Далі налаштували наше API в консолі розробника Google Cloud (рисунок 4.13).

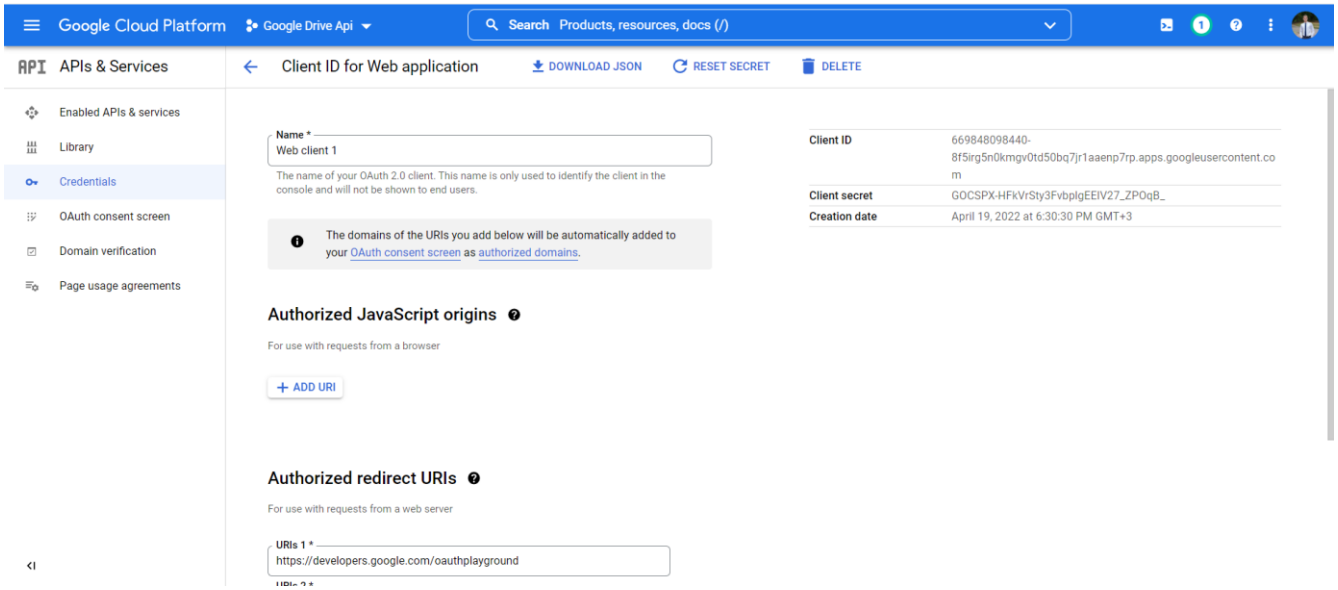

Рисунок 4.13 — Налаштування Google Cloud API

Після завершення налаштування ми отримали унікальний client_id і project_id за допомогою якого вже в нашому сервісі зможемо налаштувати передавання обраних файлів на Google Drive.

Перевіряємо можливість з’єданання за допомогою сервісу OAuth 2.0 Playground від Google (рисунок 4.14).

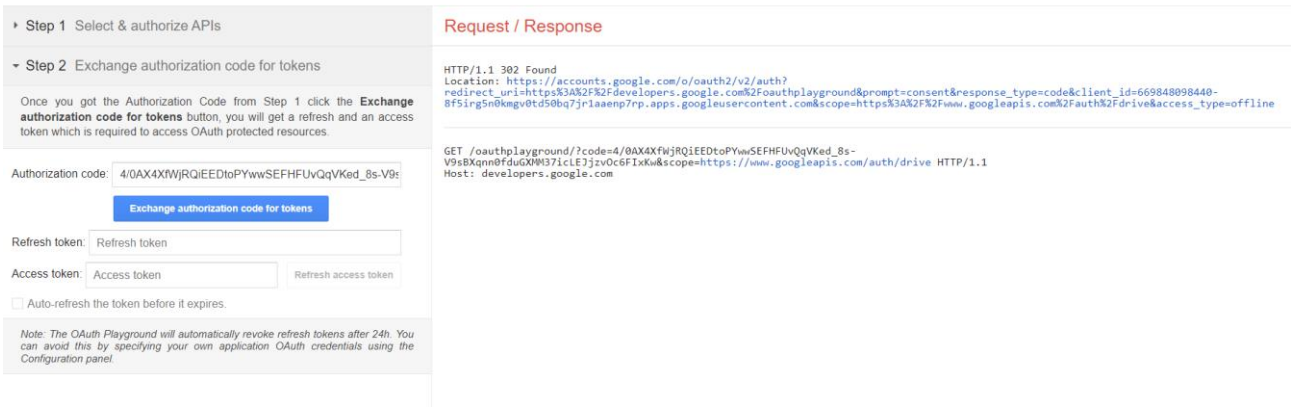

Рисунок 4.14 — Результат успішного підключення до Google Cloud

4.4 Інтерфейс користувача

Функції, можливі для виконання користувачем системи, описані на use-case діаграмі (рисунок 4.15).

Рисунок 4.15 — Use-case діаграма для користувача системи

Інтерфейс користувача розроблено за допомогою фреймворка Thymeleaf, який чудово поєднується і взаємодіє з Java кодом.

4.5 Опис функцій програмного забезпечення системи

На рисунку 4.16 зображено основні функції системи.

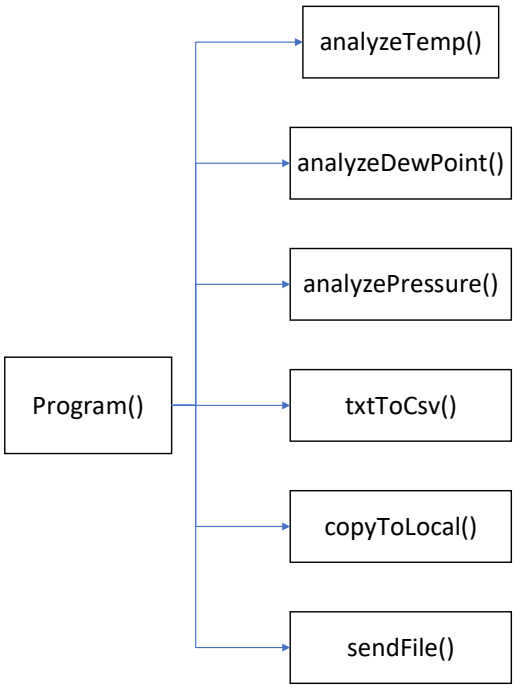

Рисунок 4.16 — Основні функції системи

Опис функцій системи міститься у таблиці 4.1

Таблиця 4.1 Опис функцій системи

№	Назва функції	Вхідні параметри	Вихідні параметри	Призначення
1	2	3	4	5
1	analyzeTemp()	Дані які поступають в залежності від вибору користувача	Оброблені дані за критерієм	Функція обробки надвеликих масивів даних в режимі реального часу за критерієм температура
2	analyzePressure()	Дані які поступають в залежності від вибору користувача	Оброблені дані за критерієм	Функція обробки надвеликих масивів даних в режимі реального часу за критерієм тиск
3	analyzeDewPoint()	Дані які поступають в залежності від вибору користувача	Оброблені дані за критерієм	Функція обробки надвеликих масивів даних в режимі реального часу за критерієм точка роси

Продовження таблиці 4.1

1	2	3	4	5
4	txtToCsv()	Файл з розширенням .txt	Файл з розширенням .csv	Функція конвертації з одного формату в інший
5	copyToLocal()	Файл у hdfs	Файл на комп’ютері	Функція копіювання файлів з hdfs на локальне сховище
6	sendFile()	Файл для передачі		Функція передачі файла на Google Drive
7	Main()			Функція запуску програмного рішення та завантаження конфігурації

В даній таблиці описані основні функції системи необхідні для обробки та передавання надвеликих масивів даних. Для кожної функції описані її вхідні, вихідні параметри та основне призначення.

4.6 Аналіз конфігурації системи

Для вимірювання швидкодії розробленої системи було проведено тестування програмного продукту, при чому набір даних зберігався постійним а кількість вузлів для обробки збільшувалась.

Алгоритм повинен відображати лінійну поведінку, тобто не мати витрат на накладні витрати між вузлами.

Під час експерименту міжвузлові витрати збільшувались, оскільки кількість обчислювальних вузлів також збільшилась. На рисунку 4.17 показано ефективність обробки певного розміру дата сету в залежності від представленої кількості вузлів.

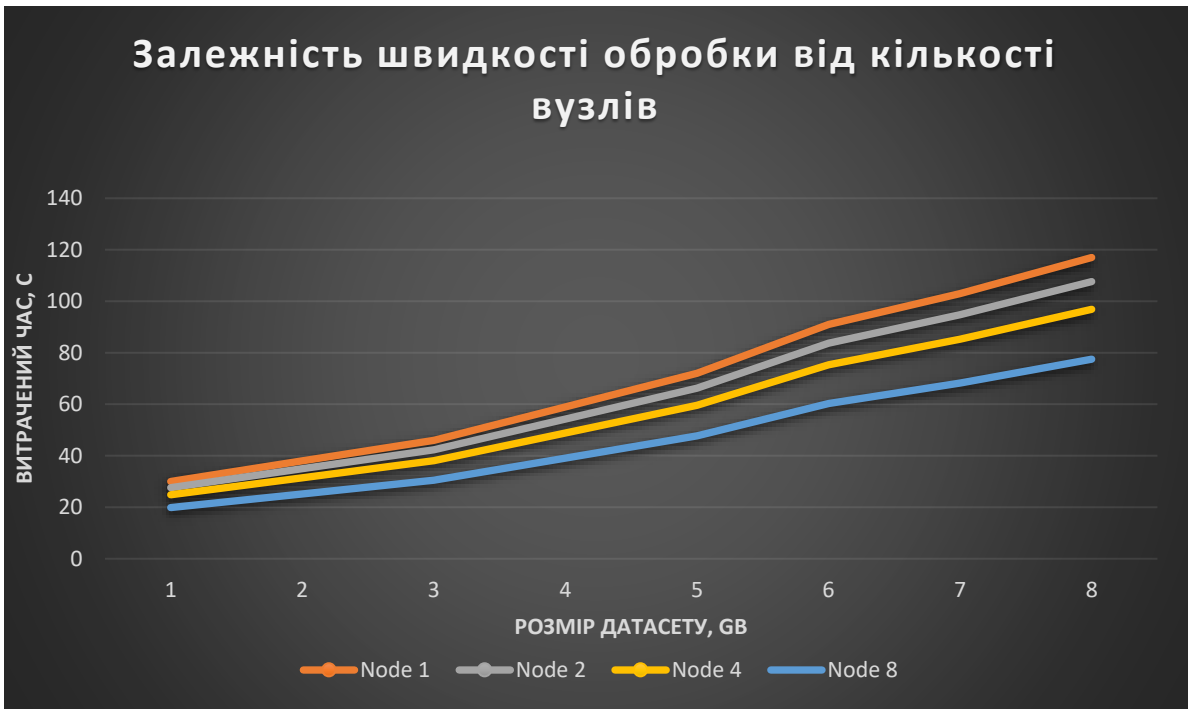

Рисунок 4.17 — Залежність швидкості обробки від кількості вузлів

У ідеальних умовах і без витрат на зв'язування дане рішення повинно виконуватись лінійно, обробляючи великі обсяги даних на одному або більшій кількості вузлів, що ми відповідно бачимо вище.

Висновки до розділу 4

В даному розділі було розглянуто процес реалізації програмної системи: встановлення і налаштування сервера, реалізація алгоритму обробки надвеликих масивів даних, передавання отриманих результатів.

Також в розділі продемонстровано функції користувача у вигляді use-case діаграми і наведено опис основних функцій користувача.

Система була проаналізована на швидкодію в залежності від різних конфігурацій, що було відповідно зображено на графіку.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Користувач повинен дотримуватись усіх вимог роботи з користувацьким інтерфейсом додатку для коректної роботи усіх модулів системи.

5.1 Системні вимоги

Для забезпечення стабільної та коректної роботи розробленої системи оброблення та передавання надвеликих масивів даних пристрій, на якому буде відбуватися весь процес, повинен мати декілька вимог:

– встановлену операційну систему 64-розрядну або 32-розрядну Microsoft Windows 10, Microsoft Windows 8, Microsoft Windows 7, Mac OS або Linux;

– персональний комп’ютер повинен мати процесор не гірший за Intel Core i3, або з тактовою частотою вище 2.4 ГГц;

– з обсягом оперативної пам’яті не менше 8 ГБ, так як є навантаження від роботи одразу декількох серверів;

– дискретну відеокарту з обсягом пам’яті від 4 ГБ;

Також користувачу необхідно мати доступ до інтернету зі середньою швидкістю від 10 Мбіт/с.

5.2 Інструкції щодо встановлення

Система використовує плагін Maven – засіб автоматизації роботи з програмними проектами, завдяки яким запуск сервісів відбувається запуском однієї команди.

Далі необхідно перейти у папку з проектом та виконати команду: $ mvn spring-boot:run

Все, що необхідно потім зробити користувачу, це перейти за посиланням http://localhost:8080/, якщо це локальний комп’ютер, або за адресою віддаленого серверу.

5.3 Взаємодія користувача з системою

Додаток має початковий екран, де у верхній частині є навігаційне меню по видах обробки та можливість передавання даних (рисунок 5.1). У верхній частині вікна зображено логотип та назва системи.

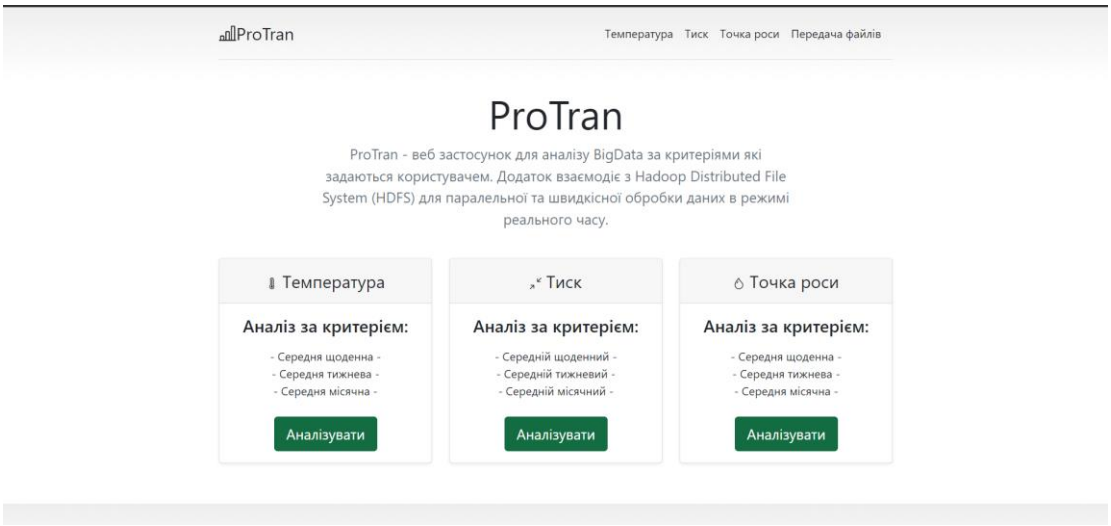

Рисунок 5.1 — Початковий екран системи

Для початку обробки нашого дата сету необхідно обрати один з наявних способів обробки в навігаційному меню або натиснути одну з кнопок «Аналізувати» на головному екрані, попередньо вибравши критерій (рисунок 5.2).

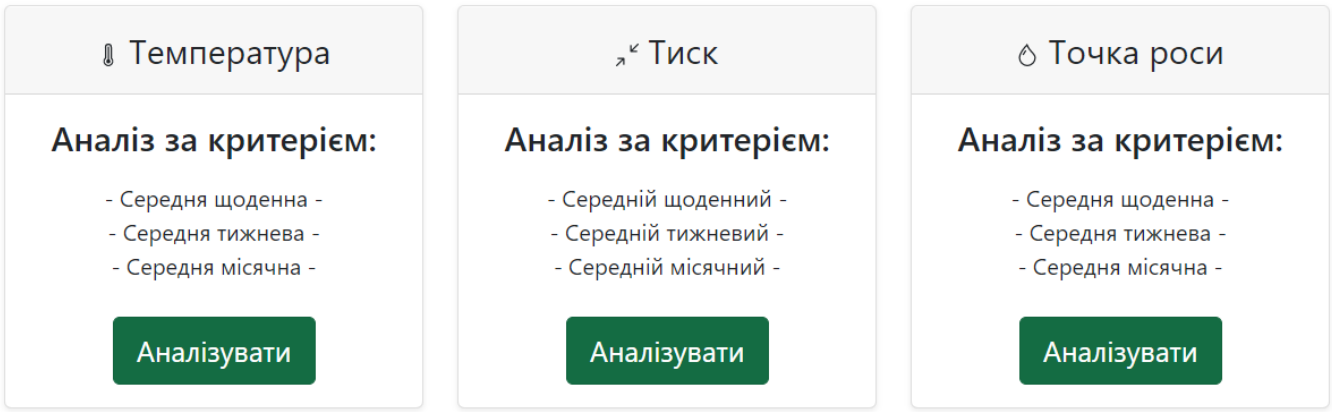

Рисунок 5.2 — Критерії обробки даних

Після вибору одного з критеріїв обробки користувач попадає на нову сторінку (рисунок 5.3), де має можливість обрати інтервали для визначення дати початку і кінця, тобто межі обробки по часу та тип обробки:

– за день;

– за тиждень;

– за місяць.

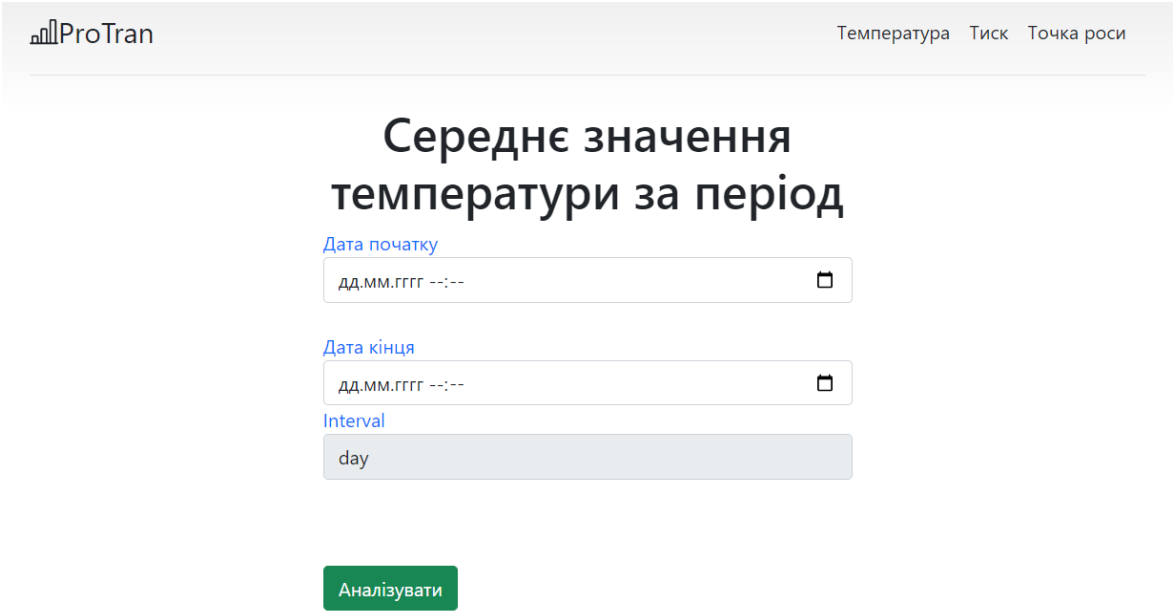

Рисунок 5.3 — Сторінка обробки даних за критерієм температура

Після того як користувач вибере дату початку аналізу та дату кінця, а також інтервал аналізу, він може натиснути на кнопку «Аналізувати», після чого дані почнуть надходити в систему і відповідно до критерію оброблятись (рисунок 5.4).

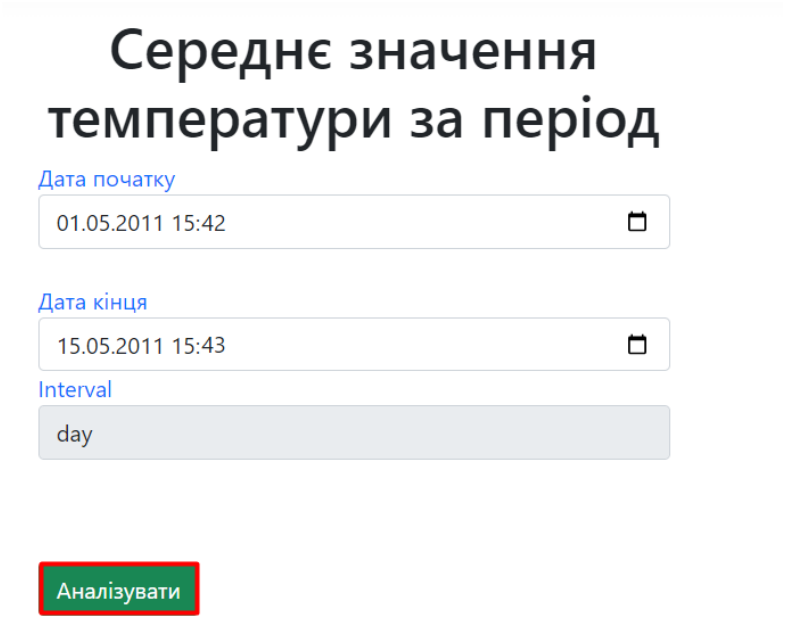

Рисунок 5.4 — Обробка в вибраному діапазоні за температурою по днях

Після закінчення аналізу користувач отримає відповідне повідомлення про успішне виконання обробки (рисунок 5.5).

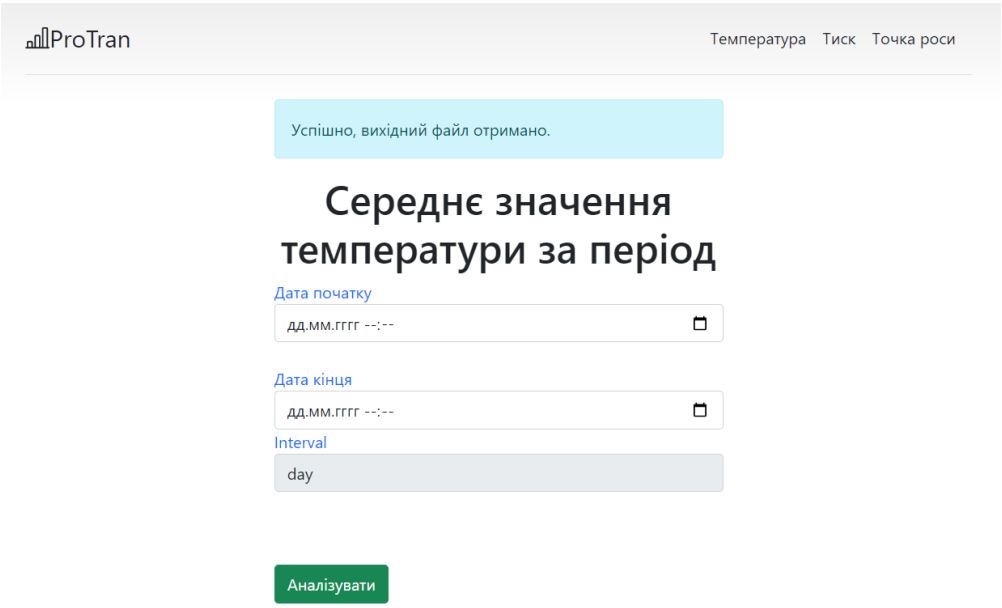

Рисунок 5.5 — Успішна обробка за критерієм температура у вибраному діапазоні

У результаті обробки користувач отримує вихідний файл з обробленими даними у форматі .txt який автоматично конвертується в систему у файл з розширенням .csv (рисунок 5.6).

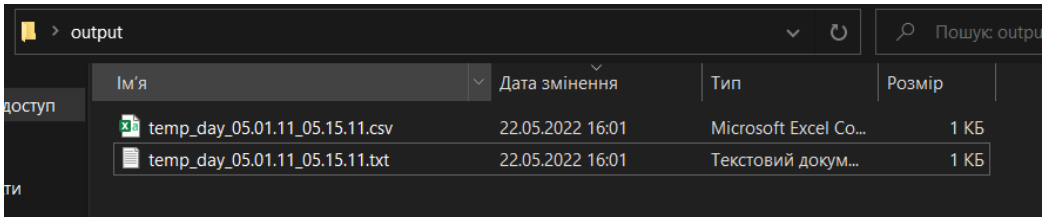

Рисунок 5.6 — Отримання файлів з обробленими даними

Після чого користувач може провести обробку даних за наступним будь-яким вибраним критерієм, наприклад, тиск (рисунок 5.7).

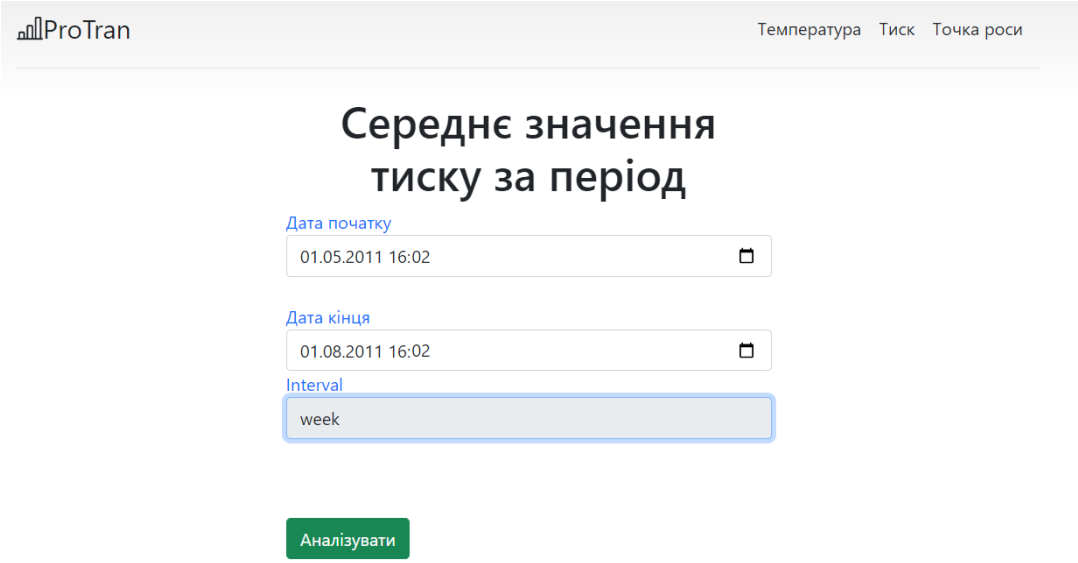

Рисунок 5.7 — Обробка в вибраному діапазоні за тиском по тижнях

Оновлений список вихідних файлів з даними зображено на рисунку 5.8.

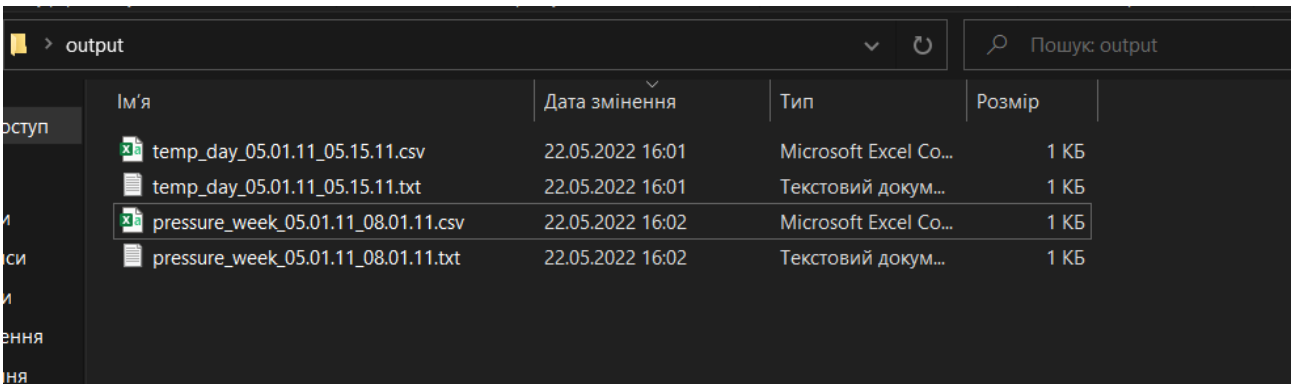

Рисунок 5.8 — Отримання файлів з обробленими даними

Також система має на меті ще один вид обробки за критерієм «точка роси», а також з інтервалом в місяць(рисунок 5.9).

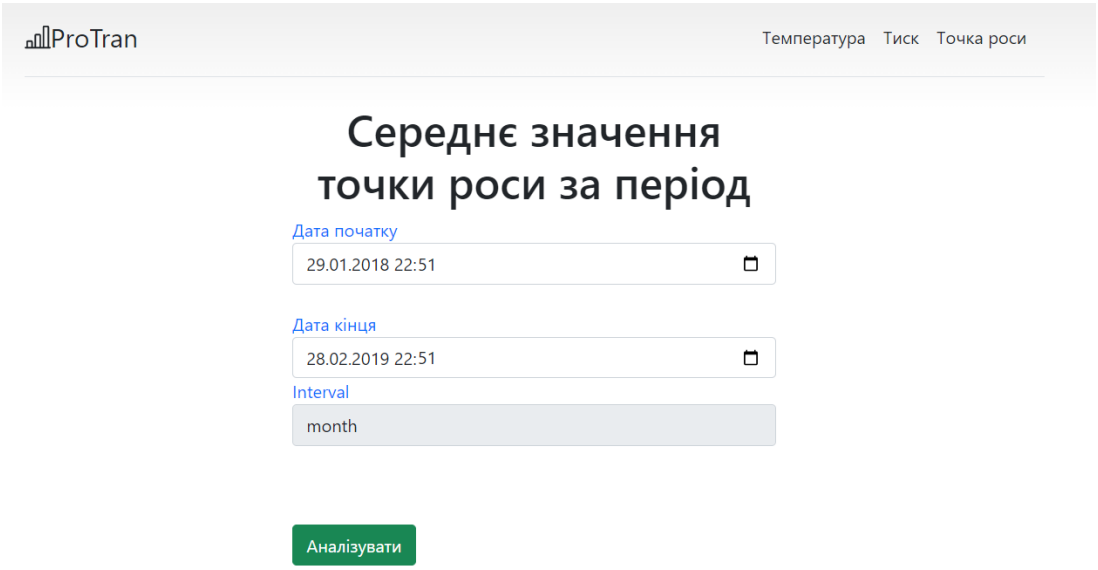

Рисунок 5.9 — Обробка в вибраному діапазоні за точкою роси по місяцях

Для передачі одного з файлів, які ми отримали в результаті обробки необхідно перейти до пункту навігації «Передача файлів», після чого відкриється сторінка для передачі файлів на Google Drive (рисунок 5.10).

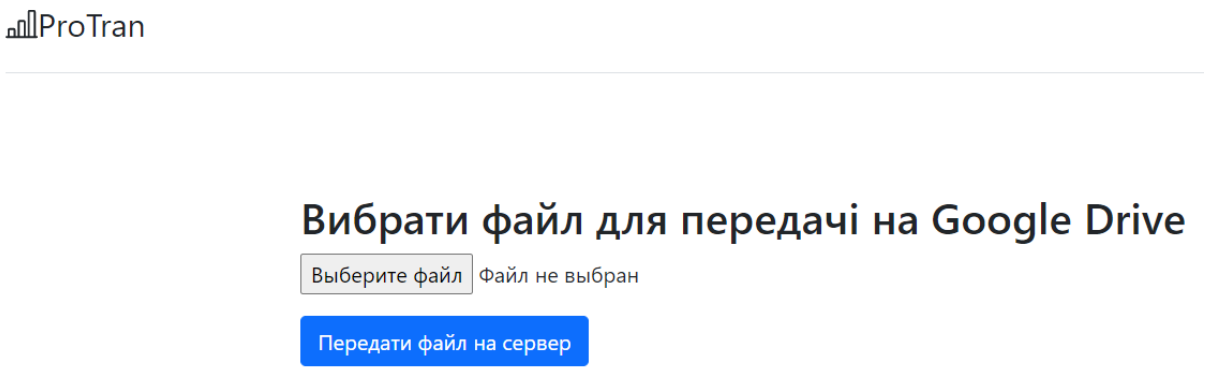

Рисунок 5.10 — Сторінка для передавання оброблених даних

Для того щоб передати файл на Google Drive користувачу необхідно натиснути на кнопку «Вибрати файл» потім у провіднику обрати необхідний файл та натиснути кнопку «Передати файл на сервер». При успішній дії ми отримуємо відповідне повідомлення (рисунок 5.11). Надісланий файл з’явиться на нашому Google Drive (рисунок 5.12).

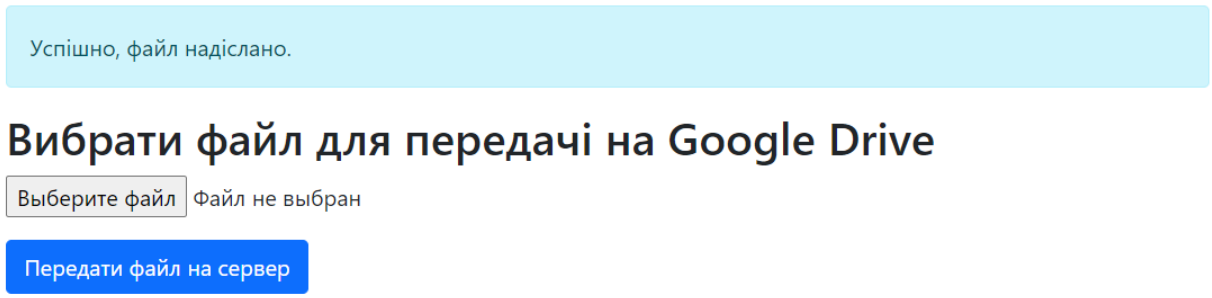

Рисунок 5.11 — Передача файлу даних на Google Drive

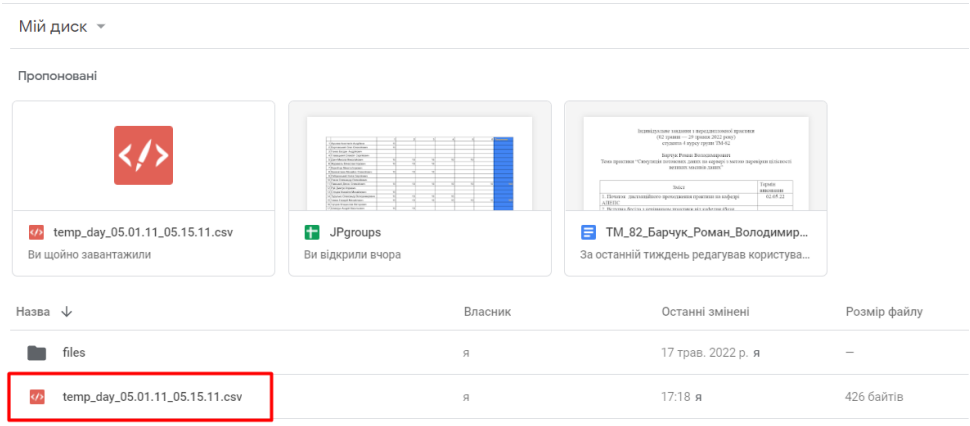

Рисунок 5.12 — Файл з даними на Google Drive

Оброблені дані користувач може візуалізувати, проводити аналіз, передавати на власний розсуд. Так як дані після обробки зберігаються у форматі .csv всі доступні платформи та засоби візуалізації підтримують його. Приклад візуалізації отриманих даних зображено на рисунку 5.13.

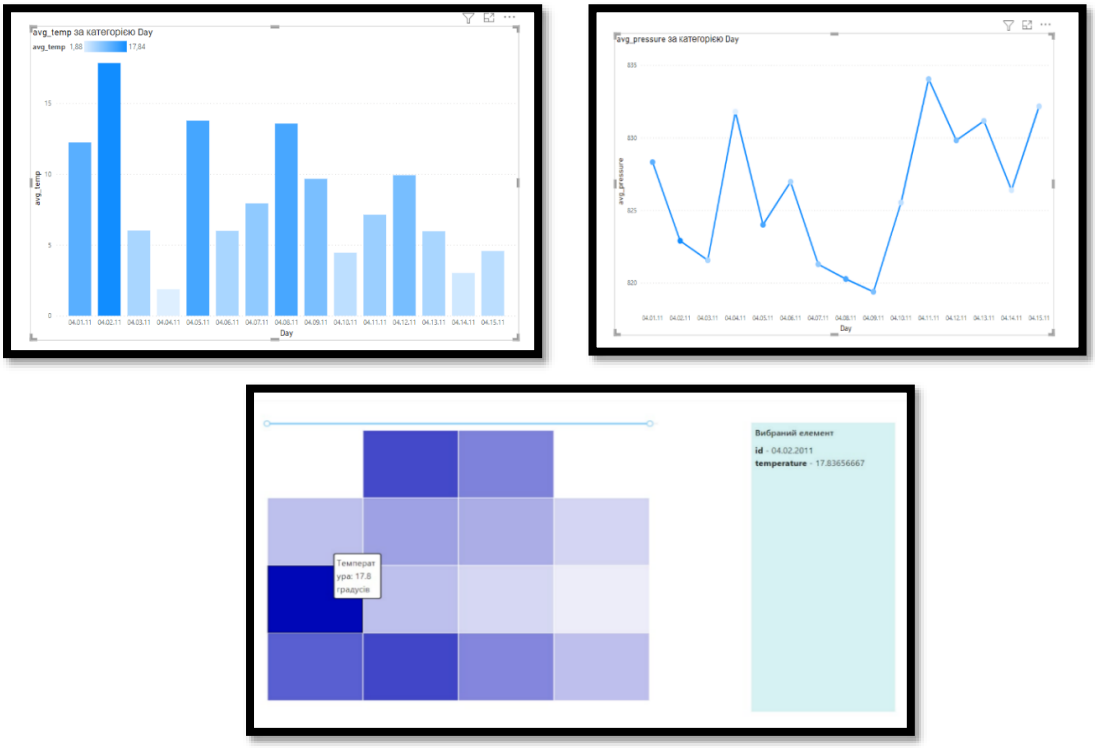

Рисунок 5.13 — Приклад візуалізаці отриманих результатів обробки

Дані відображено у вигляді звичайно стовпчастої діаграми, лінійчастої діаграми та у вигляді карти температур, завдяки чому дані легше аналізувати та сприймати.

Висновки до розділу 5

В даному розділі представлені системні вимоги необхідні для використання системи. Описано дії щодо запуску системи для подальшого користування. Надано інструкції щодо взаємодії користувача системи та продемонстровані варіанти вихідних даних та подальшого передавання їх та візуалізації.

ВИСНОВКИ

В ході виконання даної роботи було розроблено систему за темою дипломної роботи: “Обробка та передавання надвеликих масивів даних в режимі реального часу для елементів сенсорної мережі”, яка повинна обробляти вхідні дані і мати можливість передати отримані результати з метою подальшого розповсюдження, аналізу чи візуалізації.

В ході реалізації завдання було проаналізовано літературу, а також досліджено предметну область. Проаналізовано існуючі алгоритми та підходи для вирішення задачі обробки надвеликих масивів даних у реальному часі. Алгоритми, підходи, які досліджувалися стали базою для подальшого удосконалення.

В результаті чого розроблена система за допомогою останніх доступних на поточний момент інструментів, технологій, які дають можливість максимально розкрити потенціал системи. Була проведена оптимізація процесу розробки і протестовані різні конфігурації системи. В результаті чого система показує гарну швидкодію обробки враховуючи розмір даних. Схематично була описана структура програмного продукту, використовуючи UML-діаграми, процес взаємодії користувача з системою.

Для створення системи використовувалась розподілена файлова система HDFS, алгоритм обробки надвеликих масивів даних MapReduce, мова програмування Java, серверна частина написана за допомогою Spring Framework, а інтерфейс користувача реалізований використовуючи бібліотеку Thymeleaf.

Система спроектована з точки зору можливого подальшого вдосконалення. Тому під час розробки програмного забезпечення враховувалась можливість подальшого розширення системи та додавання нових можливостей обробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Big Data. [Електронний ресурс] - Режим доступу до ресурсу: https://www.it.ua/ru/knowledge-base/technology-innovation/big-data-bolshie-dannye

2. Big Data storage. [Електронний ресурс] - Режим доступу до ресурсу: https://bigdata-madesimple.com/data-storage-which-of-the-3-types-is-best-for-your-business/

3. Data types. [Електронний ресурс] - Режим доступу до ресурсу: https://www.datamation.com/big-data/structured-vs-unstructured-data/

4. Data transfer. [Електронний ресурс] - Режим доступу до ресурсу: https://www.ibm.com/topics/file-transfer

5. Hadoop. Advantages and Disadvantages. [Електронний ресурс] - Режим доступу до ресурсу: http://way2benefits.com/advantages-and-disadvantages-of-hadoop/

6. Apache Spark. Advantages and Disadvantages. [Електронний ресурс] - Режим доступу до ресурсу: https://www.knowledgehut.com/blog/big-data/apache-spark-advantages-disadvantages

7. An introduction to Apache Storm. [Електронний ресурс] - Режим доступу до ресурсу: https://phoenixnap.com/kb/apache-storm

8. What is HDFS? [Електронний ресурс] - Режим доступу до ресурсу: https://phoenixnap.com/kb/what-is-hdfs

9. Understanding MapReduce in Hadoop. [Електронний ресурс] - Режим доступу до ресурсу: https://www.section.io/engineering-education/understanding-map-reduce-in-hadoop/

10. Федорова Н.В., Демченко О.Е. Обробка та передавання надвеликих масивів даних в режимі реального часу для елементів сенсорної мережі. Наукові читання імені Ігоря Недіна : збірка матеріалів (наукових доповідей) VII Міжнародної науково-практичної онлайн-конференції. Київ, 2021. С. 352–356.

11. Schildt H. Java: The Complete Reference, Tenth Edition. New York, 2018. 730 p.

12. Walls C. Spring in Action, Fifth Edition. New York, 2018. 520 p.

13. Wim D. Taming Thymeleaf: Practical guide to building a webapplication with Spring Boot and Thymeleaf. Belgium, 2021. 608 p.

ДОДАТОК А

Обробка та передавання надвеликих масивів даних в режимі реального часу для елементів сенсорної мережі

Код програмного модулю

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТМ82???_???

Аркушів 18

2022

```
package com.protran.bd.controller;

import com.protran.bd.DateTransporter;
import com.protran.bd.Type;
import com.protran.bd.converter.CopyToLocal;
import com.protran.bd.converter.TxtToCsv;
import com.protran.bd.dew_point.day.DewDayDriver;
import com.protran.bd.dew_point.day.DewDayMapper;
import com.protran.bd.dew_point.month.DewMonthDriver;
import com.protran.bd.dew_point.week.DewWeekDriver;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import java.time.format.DateTimeFormatter;

@Controller
public class ControllerDewPoint {
    @GetMapping("/dew_point")
    public String temperaturePage(Model model){
        model.addAttribute("dewTrans", new DateTransporter());
        return "/dew_point";
    }

    @PostMapping("/analyze_dew_point")
    public String analyzeTemperature(@ModelAttribute("dewTrans") DateTransporter,
                                     BindingResult bindingResult) throws Exception {
        if (bindingResult.hasErrors()) {
            return "/dew_point";
        }
        String [] args = new String[]{CONSTANTS.INPUT_FILE_DIR, CONSTANTS.OUTPUT_FILE_DIR};
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM/dd/yy");
        String before = dateTransporter.getDateStart().format(formatter);
        String after = dateTransporter.getDateEnd().format(formatter);
        DewDayMapper = DewDayMapper.getInstance();
        dewDayMapper.setUserDateBefore(before);
        dewDayMapper.setUserDateAfter(after);
        if(dateTransporter.getType().equals(Type.day)){
            DewDayDriver.main(args);
            CopyToLocal.copyToLocalDew(Type.day.name());
            TxtToCsv.converterDew(Type.day.name());
        }else if(dateTransporter.getType().equals(Type.week)){
            DewWeekDriver.main(args);
            CopyToLocal.copyToLocalDew(Type.week.name());
            TxtToCsv.converterDew(Type.week.name());
        }else if(dateTransporter.getType().equals(Type.month)){
            DewMonthDriver.main(args);
            CopyToLocal.copyToLocalDew(Type.month.name());
            TxtToCsv.converterDew(Type.month.name());
        }
        return "redirect:/pressure?success";
    }
}

package com.protran.bd.controller;

import com.protran.bd.DateTransporter;
import com.protran.bd.Type;
import com.protran.bd.converter.CopyToLocal;
import com.protran.bd.converter.TxtToCsv;
```

```
import com.protran.bd.pressure.day.PressureDayDriver;
import com.protran.bd.pressure.day.PressureDayMapper;
import com.protran.bd.pressure.month.PressureMonthDriver;
import com.protran.bd.pressure.week.PressureWeekDriver;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import java.time.format.DateTimeFormatter;

@Controller
public class ControllerPressure {
   @GetMapping("/pressure")
   public String temperaturePage(Model model){
      model.addAttribute("pressTrans", new DateTransporter());
      return "/pressure";
   }

   @PostMapping("/analyze_pressure")
   public String analyzeTemperature(@ModelAttribute("pressTrans") DateTransporter,
                                    BindingResult bindingResult) throws Exception {
      if (bindingResult.hasErrors()) {
         return "/pressure";
      }
      String [] args = new String[]{CONSTANTS.INPUT_FILE_DIR, CONSTANTS.OUTPUT_FILE_DIR};
      DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM/dd/yy");
      String before = dateTransporter.getDateStart().format(formatter);
      String after = dateTransporter.getDateEnd().format(formatter);
      PressureDayMapper = PressureDayMapper.getInstance();
      pressureDayMapper.setUserDateBefore(before);
      pressureDayMapper.setUserDateAfter(after);
      if(dateTransporter.getType().equals(Type.day)){
         PressureDayDriver.main(args);
         CopyToLocal.copyToLocalPress(Type.day.name());
         TxtToCsv.converterPress(Type.day.name());
      }else if(dateTransporter.getType().equals(Type.week)){
         PressureWeekDriver.main(args);
         CopyToLocal.copyToLocalPress(Type.week.name());
         TxtToCsv.converterPress(Type.week.name());
      }else if(dateTransporter.getType().equals(Type.month)){
         PressureMonthDriver.main(args);
         CopyToLocal.copyToLocalPress(Type.month.name());
         TxtToCsv.converterPress(Type.month.name());
      }
      return "redirect:/pressure?success";
   }
}

package com.protran.bd.controller;

import com.protran.bd.DateTransporter;
import com.protran.bd.Type;
import com.protran.bd.converter.CopyToLocal;
import com.protran.bd.converter.TxtToCsv;
import com.protran.bd.pressure.day.PressureDayDriver;
import com.protran.bd.pressure.day.PressureDayMapper;
import com.protran.bd.temperature.day.TempDayDriver;
import com.protran.bd.temperature.day.TempDayMapper;
import com.protran.bd.temperature.month.TempMonthDriver;
import com.protran.bd.temperature.week.TempWeekDriver;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import java.time.format.DateTimeFormatter;

@Controller
public class ControllerTemperature {

    @GetMapping("/temperature")
    public String temperaturePage(Model model){
        model.addAttribute("tempTrans", new DateTransporter());
        return "/temperature";
    }

    @PostMapping("/analyze_temp")
    public String analyzeTemperature(@ModelAttribute("tempTrans") DateTransporter,
                                     BindingResult bindingResult) throws Exception {
        if (bindingResult.hasErrors()) {
            return "/temperature";
        }
        String [] args = new String[]{CONSTANTS.INPUT_FILE_DIR, CONSTANTS.OUTPUT_FILE_DIR};
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM/dd/yy");
        String before = dateTransporter.getDateStart().format(formatter);
        String after = dateTransporter.getDateEnd().format(formatter);
        TempDayMapper = TempDayMapper.getInstance();
        tempDayMapper.setUserDateBefore(before);
        tempDayMapper.setUserDateAfter(after);
        if(dateTransporter.getType().equals(Type.day)){
            TempDayDriver.main(args);
            CopyToLocal.copyToLocalTemp(Type.day.name());
            TxtToCsv.converterTemp(Type.day.name());
        }else if(dateTransporter.getType().equals(Type.week)){
            TempWeekDriver.main(args);
            CopyToLocal.copyToLocalTemp(Type.week.name());
            TxtToCsv.converterTemp(Type.week.name());
        }else if(dateTransporter.getType().equals(Type.month)){
            TempMonthDriver.main(args);
            CopyToLocal.copyToLocalTemp(Type.month.name());
            TxtToCsv.converterTemp(Type.month.name());
        }
        return "redirect:/temperature?success";
    }
} package com.protran.bd.controller;

import com.google.api.client.auth.oauth2.Credential;
import com.google.api.client.extensions.java6.auth.oauth2.AuthorizationCodeInstalledApp;
import com.google.api.client.extensions.jetty.auth.oauth2.LocalServerReceiver;
import com.google.api.client.googleapis.auth.oauth2.GoogleAuthorizationCodeFlow;
import com.google.api.client.googleapis.auth.oauth2.GoogleClientSecrets;
import com.google.api.client.googleapis.javanet.GoogleNetHttpTransport;
import com.google.api.client.http.FileContent;
import com.google.api.client.http.javanet.NetHttpTransport;
import com.google.api.client.json.JsonFactory;
import com.google.api.client.json.gson.GsonFactory;
import com.google.api.client.util.store.FileDataStoreFactory;
import com.google.api.services.drive.Drive;
import com.google.api.services.drive.DriveScopes;
import com.google.api.services.drive.model.File;
import org.springframework.stereotype.Controller;
```

```
import org.springframework.util.StringUtils;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.security.GeneralSecurityException;
import java.util.Collections;
import java.util.List;

@Controller
public class ControllerUpload {
   private final String UPLOAD_DIR = "C:\\Users\\olexa\\Desktop\\output\\";

   private static final String APPLICATION_NAME = "Google Drive Api";
   private static final JsonFactory JSON_FACTORY = GsonFactory.getDefaultInstance();
   private static final String TOKENS_DIRECTORY_PATH = "tokens";
   private static final List<String> SCOPES = Collections.singletonList(DriveScopes.DRIVE);
   private static final String CREDENTIALS_FILE_PATH = "/credit.json";

   @GetMapping("/upload")
   public String homepage() {
      return "upload";
   }

   @PostMapping("/upload")
   public String uploadFile(@RequestParam("file") MultipartFile file, RedirectAttributes attributes) throws
GeneralSecurityException, IOException {
      if (file.isEmpty()) {
         attributes.addFlashAttribute("message", "Please select a file to upload.");
         return "redirect:/";
      }
      String fileName = StringUtils.cleanPath(file.getOriginalFilename());
      Path = Paths.get(UPLOAD_DIR + fileName);
      ControllerUpload.sendFile(path.toString());
      return "redirect:/upload?success";
   }


   private static Credential getCredentials(final NetHttpTransport HTTP_TRANSPORT) throws IOException {
      // Load client secrets.
      InputStream in = ControllerUpload.class.getResourceAsStream(CREDENTIALS_FILE_PATH);
      if (in == null) {
         throw new FileNotFoundException("Resource not found: " + CREDENTIALS_FILE_PATH);
      }
      GoogleClientSecrets clientSecrets = GoogleClientSecrets.load(JSON_FACTORY, new InputStreamReader(in));

      // Build flow and trigger user authorization request.
      GoogleAuthorizationCodeFlow flow = new GoogleAuthorizationCodeFlow.Builder(
            HTTP_TRANSPORT, JSON_FACTORY, clientSecrets, SCOPES)
            .setDataStoreFactory(new FileDataStoreFactory(new java.io.File(TOKENS_DIRECTORY_PATH)))
            .setAccessType("offline")
            .build();
      LocalServerReceiver receiver = new LocalServerReceiver.Builder().setPort(8888).build();
```

```
        Credential = new AuthorizationCodeInstalledApp(flow, receiver).authorize("user");
        return credential;
    }

    public static void sendFile(String arg) throws IOException, GeneralSecurityException {
        final NetHttpTransport HTTP_TRANSPORT = GoogleNetHttpTransport.newTrustedTransport();
        Drive service = new Drive.Builder(HTTP_TRANSPORT, JSON_FACTORY, getCredentials(HTTP_TRANSPORT))
                .setApplicationName(APPLICATION_NAME)
                .build();
        File fileMetadata = new File();
        fileMetadata.setName(arg.substring(30));
        java.io.File filePath = new java.io.File(arg);
        FileContent mediaContent = new FileContent("file/txt", filePath);
        File = service.files().create(fileMetadata, mediaContent)
                .setFields("id")
                .execute();
        System.out.println("File ID: " + file.getId());
    }
}


package com.protran.bd.converter;
import com.protran.bd.dew_point.day.DewDayMapper;
import com.protran.bd.pressure.day.PressureDayMapper;
import com.protran.bd.temperature.day.TempDayMapper;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URI;

public class CopyToLocal {
    public static void copyToLocalTemp(String period) throws IOException {
        String name = TempDayMapper.getInstance().getUserDateBefore() + "_" +
TempDayMapper.getInstance().getUserDateAfter();
        name = name.replaceAll("/", ".");
        String dest = "hdfs://localhost:9000/hadoop/output_temp_" + period + "_" + name + "/part-r-00000";
        String local = "C:\\Users\\olexa\\Desktop\\output\\" + "temp_" + period + "_" + name + ".txt";
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(URI.create(dest),conf);
        FSDataInputStream fsdi = fs.open(new Path(dest));
        OutputStream output = new FileOutputStream(local);
        IOUtils.copyBytes(fsdi,output,4096,true);
    }
    public static void copyToLocalPress(String period) throws IOException {
        String name = PressureDayMapper.getInstance().getUserDateBefore() + "_" +
PressureDayMapper.getInstance().getUserDateAfter();
        name = name.replaceAll("/", ".");
        String dest = "hdfs://localhost:9000/hadoop/output_pressure_" + period + "_" + name + "/part-r-00000";
        String local = "C:\\Users\\olexa\\Desktop\\output\\" + "pressure_" + period + "_" + name + ".txt";
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(URI.create(dest),conf);
        FSDataInputStream fsdi = fs.open(new Path(dest));
        OutputStream output = new FileOutputStream(local);
        IOUtils.copyBytes(fsdi,output,4096,true);
    }
    public static void copyToLocalDew(String period) throws IOException {
        String name = DewDayMapper.getInstance().getUserDateBefore() + "_" +
DewDayMapper.getInstance().getUserDateAfter();
```

```
        name = name.replaceAll("/", ".");
        String dest = "hdfs://localhost:9000/hadoop/output_dew_" + period + "_" + name + "/part-r-00000";
        String local = "C:\\Users\\olexa\\Desktop\\output\\" + "dew_" + period + "_" + name + ".txt";
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(URI.create(dest),conf);
        FSDataInputStream fsdi = fs.open(new Path(dest));
        OutputStream output = new FileOutputStream(local);
        IOUtils.copyBytes(fsdi,output,4096,true);
    }
}


package com.protran.bd.converter;

import com.protran.bd.dew_point.day.DewDayMapper;
import com.protran.bd.pressure.day.PressureDayMapper;
import com.protran.bd.temperature.day.TempDayMapper;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class TxtToCsv {
    public static void converterTemp(String period) throws IOException {
        if (period.equals("day")) {
            FileWriter writer;
            String name = TempDayMapper.getInstance().getUserDateBefore() + "_" +
TempDayMapper.getInstance().getUserDateAfter();
            name = name.replaceAll("/", ".");
            File = new File("C:\\Users\\olexa\\Desktop\\output\\" + "temp_" + period + "_" + name + ".txt");
            Scanner scan = new Scanner(file);
            File file2 = new File("C:\\Users\\olexa\\Desktop\\output\\" + "temp_" + period + "_" + name + ".csv");
            file.createNewFile();
            writer = new FileWriter(file2);
            String csv = "";
            csv = scan.nextLine();
            writer.append(csv.substring(0, 3)).append(";").append(csv.substring(13, 21)).append("\n");
            Scanner scan2 = new Scanner(file);
            while (scan2.hasNext()) {
                csv = scan2.nextLine().replace(".", ",").replace("/", ".")
                        .replace("Day ", "")
                        .replace(" avg_temp ", ";");
                writer.append(csv);
                writer.append("\n");
                writer.flush();
            }
        }
        if (period.equals("week")) {
            FileWriter writer;
            String name = TempDayMapper.getInstance().getUserDateBefore() + "_" +
TempDayMapper.getInstance().getUserDateAfter();
            name = name.replaceAll("/", ".");
            File = new File("C:\\Users\\olexa\\Desktop\\output\\" + "temp_" + period + "_" + name + ".txt");
            Scanner scan = new Scanner(file);
            File file2 = new File("C:\\Users\\olexa\\Desktop\\output\\" + "temp_" + period + "_" + name + ".csv");
            file.createNewFile();
            writer = new FileWriter(file2);
            String csv = "";
            csv = scan.nextLine();
            writer.append(csv.substring(0, 4)).append(";").append(csv.substring(27, 35)).append("\n");
            Scanner scan2 = new Scanner(file);
```

```
            while (scan2.hasNext()) {
                csv = scan2.nextLine().replace(".", ",").replace("/", ".")
                        .replace("(", "")
                        .replace(")", "")
                        .replace("Week: ", "")
                        .replace(" avg_temp: ", "");
                writer.append(csv);
                writer.append("\n");
                writer.flush();
            }
        }
        if (period.equals("month")) {
            FileWriter writer = null;
            String name = TempDayMapper.getInstance().getUserDateBefore() + "_" +
TempDayMapper.getInstance().getUserDateAfter();
            name = name.replaceAll("/", ".");
            File = new File("C:\\Users\\olexa\\Desktop\\output\\" + "temp_" + period + "_" + name + ".txt");
            Scanner scan = new Scanner(file);
            File file2 = new File("C:\\Users\\olexa\\Desktop\\output\\" + "temp_" + period + "_" + name + ".csv");
            file.createNewFile();
            writer = new FileWriter(file2);
            String csv = "";
            csv = scan.nextLine();
            writer.append(csv.substring(0, 5)).append(";").append(csv.substring(15, 23)).append("\n");
            Scanner scan2 = new Scanner(file);
            while (scan2.hasNext()) {
                csv = scan2.nextLine().replace(".", ",").replace("/", ".")
                        .replace("Month: ", "")
                        .replace(" avg_temp: ", ";");
                writer.append(csv);
                writer.append("\n");
                writer.flush();
            }
        }
    }


    public static void converterPress(String period) throws IOException {
        if (period.equals("day")) {
            FileWriter writer;
            String name = PressureDayMapper.getInstance().getUserDateBefore() + "_" +
PressureDayMapper.getInstance().getUserDateAfter();
            name = name.replaceAll("/", ".");
            File = new File("C:\\Users\\olexa\\Desktop\\output\\" + "pressure_" + period + "_" + name + ".txt");
            Scanner scan = new Scanner(file);
            File file2 = new File("C:\\Users\\olexa\\Desktop\\output\\" + "pressure_" + period + "_" + name + ".csv");
            file.createNewFile();
            writer = new FileWriter(file2);
            String csv = "";
            csv = scan.nextLine();
            writer.append(csv.substring(0, 3)).append(";").append(csv.substring(13, 25)).append("\n");
            Scanner scan2 = new Scanner(file);
            while (scan2.hasNext()) {
                csv = scan2.nextLine().replace(".", ",").replace("/", ".")
                        .replace("Day ", "")
                        .replace(" avg_pressure ", ";");
                writer.append(csv);
                writer.append("\n");
                writer.flush();
            }
        }
        if (period.equals("week")) {
            FileWriter writer;
```

```
            String name = PressureDayMapper.getInstance().getUserDateBefore() + "_" +
PressureDayMapper.getInstance().getUserDateAfter();
            name = name.replaceAll("/", ".");
            File = new File("C:\\Users\\olexa\\Desktop\\output\\" + "pressure_" + period + "_" + name + ".txt");
            Scanner scan = new Scanner(file);
            File file2 = new File("C:\\Users\\olexa\\Desktop\\output\\" + "pressure_" + period + "_" + name + ".csv");
            file.createNewFile();
            writer = new FileWriter(file2);
            String csv = "";
            csv = scan.nextLine();
            writer.append(csv.substring(0, 4)).append(";").append(csv.substring(27, 39)).append("\n");
            Scanner scan2 = new Scanner(file);
            while (scan2.hasNext()) {
               csv = scan2.nextLine().replace(".", ",").replace("/", ".")
                     .replace("(", "")
                     .replace(")", "")
                     .replace("Week: ", "")
                     .replace(" avg_pressure: ", "");
               writer.append(csv);
               writer.append("\n");
               writer.flush();
            }
         }
         if (period.equals("month")) {
            FileWriter writer = null;
            String name = PressureDayMapper.getInstance().getUserDateBefore() + "_" +
PressureDayMapper.getInstance().getUserDateAfter();
            name = name.replaceAll("/", ".");
            File = new File("C:\\Users\\olexa\\Desktop\\output\\" + "pressure_" + period + "_" + name + ".txt");
            Scanner scan = new Scanner(file);
            File file2 = new File("C:\\Users\\olexa\\Desktop\\output\\" + "pressure_" + period + "_" + name + ".csv");
            file.createNewFile();
            writer = new FileWriter(file2);
            String csv = "";
            csv = scan.nextLine();
            writer.append(csv.substring(0, 5)).append(";").append(csv.substring(15, 27)).append("\n");
            Scanner scan2 = new Scanner(file);
            while (scan2.hasNext()) {
               csv = scan2.nextLine().replace(".", ",").replace("/", ".")
                     .replace("Month: ", "")
                     .replace(" avg_pressure: ", ";");
               writer.append(csv);
               writer.append("\n");
               writer.flush();
            }
         }
      }

      public static void converterDew(String period) throws IOException {
         if (period.equals("day")) {
            FileWriter writer;
            String name = DewDayMapper.getInstance().getUserDateBefore() + "_" +
DewDayMapper.getInstance().getUserDateAfter();
            name = name.replaceAll("/", ".");
            File = new File("C:\\Users\\olexa\\Desktop\\output\\" + "dew_" + period + "_" + name + ".txt");
            Scanner scan = new Scanner(file);
            File file2 = new File("C:\\Users\\olexa\\Desktop\\output\\" + "dew_" + period + "_" + name + ".csv");
            file.createNewFile();
            writer = new FileWriter(file2);
            String csv = "";
            csv = scan.nextLine();
            writer.append(csv.substring(0, 3)).append(";").append(csv.substring(13, 20)).append("\n");
            Scanner scan2 = new Scanner(file);
```

```
            while (scan2.hasNext()) {
                csv = scan2.nextLine().replace(".", ",").replace("/", ".")
                        .replace("Day ", "")
                        .replace(" avg_dew ", ";");
                writer.append(csv);
                writer.append("\n");
                writer.flush();
            }
        }
        if (period.equals("week")) {
            FileWriter writer;
            String name = DewDayMapper.getInstance().getUserDateBefore() + "_" +
DewDayMapper.getInstance().getUserDateAfter();
            name = name.replaceAll("/", ".");
            File = new File("C:\\Users\\olexa\\Desktop\\output\\" + "dew_" + period + "_" + name + ".txt");
            Scanner scan = new Scanner(file);
            File file2 = new File("C:\\Users\\olexa\\Desktop\\output\\" + "dew_" + period + "_" + name + ".csv");
            file.createNewFile();
            writer = new FileWriter(file2);
            String csv = "";
            csv = scan.nextLine();
            writer.append(csv.substring(0, 4)).append(";").append(csv.substring(27, 34)).append("\n");
            Scanner scan2 = new Scanner(file);
            while (scan2.hasNext()) {
                csv = scan2.nextLine().replace(".", ",").replace("/", ".")
                        .replace("(", "")
                        .replace(")", "")
                        .replace("Week: ", "")
                        .replace(" avg_dew: ", "");
                writer.append(csv);
                writer.append("\n");
                writer.flush();
            }
        }
        if (period.equals("month")) {
            FileWriter writer = null;
            String name = DewDayMapper.getInstance().getUserDateBefore() + "_" +
DewDayMapper.getInstance().getUserDateAfter();
            name = name.replaceAll("/", ".");
            File = new File("C:\\Users\\olexa\\Desktop\\output\\" + "dew_" + period + "_" + name + ".txt");
            Scanner scan = new Scanner(file);
            File file2 = new File("C:\\Users\\olexa\\Desktop\\output\\" + "dew_" + period + "_" + name + ".csv");
            file.createNewFile();
            writer = new FileWriter(file2);
            String csv = "";
            csv = scan.nextLine();
            writer.append(csv.substring(0, 5)).append(";").append(csv.substring(15, 22)).append("\n");
            Scanner scan2 = new Scanner(file);
            while (scan2.hasNext()) {
                csv = scan2.nextLine().replace(".", ",").replace("/", ".")
                        .replace("Month: ", "")
                        .replace(" avg_dew: ", ";");
                writer.append(csv);
                writer.append("\n");
                writer.flush();
            }
        }
    }
}

package com.protran.bd.dew_point.day;

import org.apache.hadoop.conf.Configured;
```

```
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class DewDayDriver extends Configured implements Tool {
   public int run(String[] arg0) {
      try {
         FileSystem fs = FileSystem.get(getConf());

         Job = Job.getInstance(getConf());
         job.setJarByClass(getClass());

         job.setMapperClass(DewDayMapper.class);
         job.setMapOutputKeyClass(Text.class);
         job.setMapOutputValueClass(FloatWritable.class);

         job.setReducerClass(DewDayReducer.class);
         job.setOutputKeyClass(Text.class);
         job.setOutputValueClass(DoubleWritable.class);

         FileInputFormat.setInputPaths(job, new Path(arg0[0]));

         String before = DewDayMapper.getInstance().getUserDateBefore().replaceAll("/", ".");
         String after = DewDayMapper.getInstance().getUserDateAfter().replaceAll("/", ".");
        /* if (fs.exists(new Path(arg0[1] + " day " + before + " - " + after)))
            fs.delete(new Path(arg0[1] + " day " + before + " - " + after), true);*/

         FileOutputFormat.setOutputPath(job, new Path(arg0[1] + "_dew_day_" + before + "_" + after));

         return job.waitForCompletion(true) ? 0 : 1;
      } catch (Exception e) {
         e.printStackTrace();
      }

      return 0;
   }

   public static void main(String[] args) throws Exception {
      ToolRunner.run(new DewDayDriver(), args);
   }
}

package com.protran.bd.dew_point.day;

import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class DewDayMapper extends Mapper<LongWritable, Text, Text, FloatWritable> {
   private String userDateBefore;
```

```
private String userDateAfter;
private final static DewDayMapper INSTANCE = new DewDayMapper();

private DewDayMapper() {}

public static DewDayMapper getInstance() {
   return INSTANCE;
}

public String getUserDateBefore() {
   return userDateBefore;
}

public void setUserDateBefore(String userDateBefore) {
   this.userDateBefore = userDateBefore;
}

public String getUserDateAfter() {
   return userDateAfter;
}

public void setUserDateAfter(String userDateAfter) {
   this.userDateAfter = userDateAfter;
}

@Override
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
   try {
      //MM/DD/YYYY
      /* String userDateBefore = "03/01/11";
      String userDateAfter = "05/01/11";*/
      SimpleDateFormat format = new SimpleDateFormat();
      format.applyPattern("MM/dd/yy");
      Date userDateBeforeFormatted = null;
      try {
         userDateBeforeFormatted = format.parse(DewDayMapper.getInstance().getUserDateBefore());
      } catch (ParseException e) {
         e.printStackTrace();
      }
      Date userDateAfterFormatted = null;
      try {
         userDateAfterFormatted = format.parse(DewDayMapper.getInstance().getUserDateAfter());
      } catch (ParseException e) {
         e.printStackTrace();
      }
      String[] strArr = value.toString().split(",");
      String date = strArr[2];
      Date incomeDate = null;
      try {
         incomeDate = format.parse(date);
      } catch (ParseException e) {
         e.printStackTrace();
      }
      SimpleDateFormat dateFormat;
      Text dateText;
      float airTemp;
      if((incomeDate.before(userDateAfterFormatted) || incomeDate.equals(userDateAfterFormatted))
            && (incomeDate.after(userDateBeforeFormatted) || incomeDate.equals(userDateBeforeFormatted))) {
         dateFormat = new SimpleDateFormat("MM/dd/yy");
         dateText = new Text(dateFormat.format(incomeDate));
         airTemp = Float.parseFloat(strArr[14]);
         context.write(dateText, new FloatWritable(airTemp));
      }
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    /*
    public static LocalDateTime setDate(LocalDateTime localDateTime){
        return localDateTime;
    }*/
}


package com.protran.bd.dew_point.day;

import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class DewDayReducer extends Reducer<Text, FloatWritable, Text, NullWritable> {

    public NullWritable = NullWritable.get();
    double sum;
    double counter;

    @Override
    protected void reduce(Text key, Iterable<FloatWritable> floatTypes, Context context)
            throws IOException, InterruptedException {
        counter = 0;
        sum = 0;
        for (FloatWritable f : floatTypes) {
            sum += f.get();
            counter++;
        }
        //   String result = key.toString().substring(3,6) + key.toString().substring(0,3) + key.toString().substring(6);
        context.write(new Text("Day " + key + " avg_dew " + sum / counter), nullWritable);
    }

}

package com.protran.bd.dew_point.week;

import com.protran.bd.dew_point.day.DewDayMapper;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class DewWeekDriver extends Configured implements Tool
{
    public int run(String[] arg0) throws Exception
    {
        try
        {
            FileSystem fs = FileSystem.get(getConf());
```

```
            Job = Job.getInstance(getConf());
            job.setJarByClass(getClass());

            job.setMapperClass(DewWeekMapper.class);
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(FloatWritable.class);

            job.setReducerClass(DewWeekReducer.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(DoubleWritable.class);
            FileInputFormat.setInputPaths(job, new Path(arg0[0]));

            String before = DewDayMapper.getInstance().getUserDateBefore().replaceAll("/", ".");
            String after = DewDayMapper.getInstance().getUserDateAfter().replaceAll("/", ".");
           /* if (fs.exists(new Path(arg0[1] + " week " + before + " - " + after)))
               fs.delete(new Path(arg0[1] + " week " + before + " - " + after), true);*/

            FileOutputFormat.setOutputPath(job, new Path(arg0[1] + "_dew_week_" + before + "_" + after));

            return job.waitForCompletion(true) ? 0 : 1;
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }

         return 0;
      }

      public static void main(String[] args) throws Exception
      {
         ToolRunner.run(new DewWeekDriver(), args);
      }
   }

   package com.protran.bd.dew_point.week;

   import com.protran.bd.dew_point.day.DewDayMapper;
   import org.apache.hadoop.io.FloatWritable;
   import org.apache.hadoop.io.LongWritable;
   import org.apache.hadoop.io.Text;
   import org.apache.hadoop.mapreduce.Mapper;

   import java.io.IOException;
   import java.text.ParseException;
   import java.text.SimpleDateFormat;
   import java.time.Instant;
   import java.time.LocalDateTime;
   import java.time.ZoneId;
   import java.time.temporal.ChronoUnit;
   import java.util.Calendar;
   import java.util.Date;

   public class DewWeekMapper extends Mapper<LongWritable, Text, Text, FloatWritable> {
      @Override
      protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
         try {

           /*  //MM/DD/YYYY
            String userDateBefore = "03/01/11";
            String userDateAfter = "05/12/11";*/
            SimpleDateFormat format = new SimpleDateFormat();
```

```
            format.applyPattern("MM/dd/yy");
            Date userDateBeforeFormatted = null;
            try {
               userDateBeforeFormatted = format.parse(DewDayMapper.getInstance().getUserDateBefore());
            } catch (ParseException e) {
               e.printStackTrace();
            }
            Date userDateAfterFormatted = null;
            try {
               userDateAfterFormatted = format.parse(DewDayMapper.getInstance().getUserDateAfter());
            } catch (ParseException e) {
               e.printStackTrace();
            }
            String[] strArr = value.toString().split(",");
            String date = strArr[2];
            Date incomeDate = null;
            try {
               incomeDate = format.parse(date);
            } catch (ParseException e) {
               e.printStackTrace();
            }
            Calendar cal = Calendar.getInstance();
            cal.setTime(userDateBeforeFormatted);
            Calendar cal2 = Calendar.getInstance();
            cal2.setTime(userDateAfterFormatted);
            long fullWeeks = getFullWeeks(cal,cal2);
            cal.add(Calendar.WEEK_OF_MONTH, (int) fullWeeks);
            Date untilDate = cal.getTime();
            SimpleDateFormat dateFormat;
            Text dateText;
            float airTemp;
            if((incomeDate.before(untilDate) || incomeDate.equals(untilDate))
                  && (incomeDate.after(userDateBeforeFormatted) || incomeDate.equals(userDateBeforeFormatted))) {
               dateFormat = new SimpleDateFormat("MM/dd/yy");
               dateText = new Text(dateFormat.format(incomeDate));
               airTemp = Float.parseFloat(strArr[14]);
               context.write(dateText, new FloatWritable(airTemp));
            }
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
      public static long getFullWeeks(Calendar d1, Calendar d2){

         Instant d1i = Instant.ofEpochMilli(d1.getTimeInMillis());
         Instant d2i = Instant.ofEpochMilli(d2.getTimeInMillis());

         LocalDateTime startDate = LocalDateTime.ofInstant(d1i, ZoneId.systemDefault());
         LocalDateTime endDate = LocalDateTime.ofInstant(d2i, ZoneId.systemDefault());

         return ChronoUnit.WEEKS.between(startDate, endDate);
      }
   }

   package com.protran.bd.dew_point.week;

   import org.apache.hadoop.io.FloatWritable;
   import org.apache.hadoop.io.NullWritable;
   import org.apache.hadoop.io.Text;
   import org.apache.hadoop.mapreduce.Reducer;

   import java.io.IOException;
   import java.util.*;
```

```
public class DewWeekReducer extends Reducer<Text, FloatWritable, Text, NullWritable> {
   Map<String, Double> hm = new LinkedHashMap<>();
   public NullWritable = NullWritable.get();
   double sum;
   int counter;

   @Override
   protected void reduce(Text key, Iterable<FloatWritable> floatTypes, Context context)
         throws IOException, InterruptedException {
      counter = 0;
      sum = 0;
      for (FloatWritable f : floatTypes) {
         sum += f.get();
         counter++;
      }
      hm.put(key.toString(), sum / counter);
      if (hm.size() == 7) {
         ArrayList<Double> listOfValues = new ArrayList<>(hm.values());
         double res = listOfValues.stream().mapToDouble(value -> value).average().orElse(0);
         Optional<Map.Entry<String, Double>> first = hm.entrySet().stream().findFirst();
         List<Map.Entry<String,Double>> entryList =
               new ArrayList<>(hm.entrySet());
         Map.Entry<String, Double> lastEntry =
               entryList.get(entryList.size()-1);
         context.write(new Text("Week:" + " (" + first.get().getKey() + "-" + lastEntry.getKey() +")" +"; avg_dew: " + res),
nullWritable);
         hm.clear();
      }
   }
}


package com.protran.bd.dew_point.month;

import com.protran.bd.dew_point.day.DewDayMapper;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class DewMonthDriver extends Configured implements Tool
{
   public int run(String[] arg0) throws Exception
   {
      try
      {
         FileSystem fs = FileSystem.get(getConf());

         Job = Job.getInstance(getConf());
         job.setJarByClass(getClass());

         job.setMapperClass(DewMonthMapper.class);
         job.setMapOutputKeyClass(Text.class);
         job.setMapOutputValueClass(FloatWritable.class);
```

```
            job.setReducerClass(DewMonthReducer.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(DoubleWritable.class);

            FileInputFormat.setInputPaths(job, new Path(arg0[0]));

            String before = DewDayMapper.getInstance().getUserDateBefore().replaceAll("/", ".");
            String after = DewDayMapper.getInstance().getUserDateAfter().replaceAll("/", ".");
           /* if (fs.exists(new Path(arg0[1] + " month " + before + " - " + after)))
               fs.delete(new Path(arg0[1] + " month " + before + " - " + after), true);*/

            FileOutputFormat.setOutputPath(job, new Path(arg0[1] + "_dew_month_" + before + "_" + after));

            return job.waitForCompletion(true) ? 0 : 1;
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }

         return 0;
      }

      public static void main(String[] args) throws Exception
      {
         ToolRunner.run(new DewMonthDriver(), args);
      }
   }


   package com.protran.bd.dew_point.month;

   import com.protran.bd.dew_point.day.DewDayMapper;
   import org.apache.hadoop.io.FloatWritable;
   import org.apache.hadoop.io.LongWritable;
   import org.apache.hadoop.io.Text;
   import org.apache.hadoop.mapreduce.Mapper;

   import java.io.IOException;
   import java.text.ParseException;
   import java.text.SimpleDateFormat;
   import java.time.Instant;
   import java.time.LocalDateTime;
   import java.time.ZoneId;
   import java.time.temporal.ChronoUnit;
   import java.util.Calendar;
   import java.util.Date;

   public class DewMonthMapper extends Mapper<LongWritable, Text, Text, FloatWritable> {
      @Override
      protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
         try {
            SimpleDateFormat format = new SimpleDateFormat();
            format.applyPattern("MM/dd/yy");
            Date userDateBeforeFormatted = null;
            try {
               userDateBeforeFormatted = format.parse(DewDayMapper.getInstance().getUserDateBefore());
            } catch (ParseException e) {
               e.printStackTrace();
            }
            Date userDateAfterFormatted = null;
            try {
               userDateAfterFormatted = format.parse(DewDayMapper.getInstance().getUserDateAfter());
```

```
            } catch (ParseException e) {
               e.printStackTrace();
            }
            String[] strArr = value.toString().split(",");
            String date = strArr[2];
            Date incomeDate = null;
            try {
               incomeDate = format.parse(date);
            } catch (ParseException e) {
               e.printStackTrace();
            }
            Calendar cal = Calendar.getInstance();
            cal.setTime(userDateBeforeFormatted);
            Calendar cal2 = Calendar.getInstance();
            cal2.setTime(userDateAfterFormatted);
            long fullMonth = getFullMonth(cal,cal2);
            cal.add(Calendar.MONTH, (int) fullMonth);
            Date untilDate = cal.getTime();
           /* System.out.println(userDateBeforeFormatted);
            System.out.println(fullMonth);
            System.out.println(untilDate);*/
            SimpleDateFormat dateFormat;
            Text dateText;
            float airTemp;
            if((incomeDate.before(untilDate) || incomeDate.equals(untilDate))
                  && (incomeDate.after(userDateBeforeFormatted) || incomeDate.equals(userDateBeforeFormatted))) {
               dateFormat = new SimpleDateFormat("MM/dd/yyyy");
               dateText = new Text(dateFormat.format(incomeDate));
               airTemp = Float.parseFloat(strArr[14]);
               context.write(dateText, new FloatWritable(airTemp));
            }
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
      public static long getFullMonth(Calendar d1, Calendar d2){

         Instant d1i = Instant.ofEpochMilli(d1.getTimeInMillis());
         Instant d2i = Instant.ofEpochMilli(d2.getTimeInMillis());

         LocalDateTime startDate = LocalDateTime.ofInstant(d1i, ZoneId.systemDefault());
         LocalDateTime endDate = LocalDateTime.ofInstant(d2i, ZoneId.systemDefault());

         return ChronoUnit.MONTHS.between(startDate, endDate);
      }
   }
```

```
package com.protran.bd.dew_point.month;

import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.time.YearMonth;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Optional;

public class DewMonthReducer extends Reducer<Text, FloatWritable, Text, NullWritable> {
```

```
    Map<String, Double> hm = new LinkedHashMap<>();
    public NullWritable = NullWritable.get();
    double sum;
    int counter;


    @Override
    protected void reduce(Text key, Iterable<FloatWritable> floatTypes, Context context)
            throws IOException, InterruptedException {
        counter = 0;
        sum = 0;
        for (FloatWritable f : floatTypes) {
            sum += f.get();
            counter++;
        }

        hm.put(key.toString(), sum / counter);
       /* System.out.println(hm);
        Thread.sleep(3000);*/
        Optional<Map.Entry<String, Double>> first = hm.entrySet().stream().findFirst();
        String date = first.get().getKey();
        YearMonth yearMonthObject = YearMonth.of(Integer.parseInt(date.substring(6, 10)), Integer.parseInt(date.substring(0,
2)));
        int daysInMonth = yearMonthObject.lengthOfMonth();
       /* System.out.println(key);
        System.out.println(yearMonthObject);
        System.out.println(daysInMonth);
        Thread.sleep(300);*/
        if (hm.size() == daysInMonth) {
            ArrayList<Double> listOfValues = new ArrayList<>(hm.values());
            double res = listOfValues.stream().mapToDouble(value -> value).average().orElse(0);
           /* Optional<Map.Entry<String, Double>> first1 = hm.entrySet().stream().findFirst();
            List<Map.Entry<String,Double>> entryList =
                    new ArrayList<>(hm.entrySet());
            Map.Entry<String, Double> lastEntry =
                    entryList.get(entryList.size()-1);*/
            String str = yearMonthObject.toString().replaceAll("-", "/");
            context.write(new Text("Month: " + str + " avg_dew: " + res), nullWritable);
            hm.clear();
        }
    }
}
```

ДОДАТОК Б

Обробка та передавання надвеликих масивів даних в режимі реального часу для елементів сенсорної мережі

Апробації наукових досліджень

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТМ82???_???

Аркушів 8

2022

Національний університет “Києво-Могилянська академія”

СТАЛИЙ РОЗВИТОК — XXI СТОЛІТТЯ

Дискусії 2021

Збірка матеріалів (наукових доповідей)
VII Міжнародної науково-практичної онлайн-конференції
“Сталий розвиток — XXI століття (наукові читання імені Ігоря Неділа)”

Київ, Україна
2021

УДК 66.012:658.567.1:368.075.8
ББК 65.9(4УКР)-98

Рекомендовано до друку Вченою радою
Національного університету "Києво-Могилянська академія"

Рецензенти:
Горошкова Л.А., д-р.екон.наук, проф. кафедри екології НУ "Києво-Могилянська академія";
Меньшов О.І., д-р.геол.наук., с.н.с. Навчально-науковного інституту "Інститут геології" Київського національного університету ім. Тараса Шевченка;
Черноусенко О.Ю., д-р.техн.наук, проф., зав. кафедри теплоенергетики Національного технічного університету України "Київський політехнічний інститут ім. Ігоря Сікорського".

Сталий розвиток — XXI століття. Дискусії 2021: матеріали VII Міжнародної науково-практичної конференції / Національний університет "Києво-Могилянська академія" / за ред. проф. Хлобистова Є.В. — Київ, 2021. — 527 с. — Електронне видання. ISBN: 978-617-7668-33-5

Збірка матеріалів конференції висвітлює широке коло теоретичних і прикладних проблем соціально-економічного, техніко-технологічного, інформаційно-аналітичного, соціально-філософського та освітнього забезпечення переходу України на шлях сталого розвитку, з урахуванням сучасних трансформаційних процесів. Особливу увагу приділено проблемам моделювання суспільно-економічних і екологічних процесів для ефективного територіального й корпоративного управління, державної політики і самоврядування.

Науковий редактор: д.е.н., проф. Хлобистов Є.В.

Наведені у виданні наукові доповіді були обговорені на VII Міжнародній науково-практичній онлайн-конференції "Сталий розвиток — XXI століття (наукові читання імені Ігоря Неділа)", яка відбулася 2-3 грудня 2021 року в м. Києві.

Збережено авторську орфографію, пунктуацію і стилістику.
Відповідальність за зміст матеріалів несуть автори.

ISBN: 978-617-7668-33-5

4.24. Обробка надвеликих масивів даних в режимі реального часу для елементів Інтернету речей[683]

Вступ. У сучасному світі кількість інформації зростає з року в рік. Про це свідчить і статистика від аналітиків компанії IBS, яка провела дослідження й оцінила об'єм світових даних. Їх кількість до 2003 року становила до 5 ексабайтів даних (1 ексабайт ≈ 1 млрд гігабайтів) у 2008 році — 0,18 зеттабайта (1 зеттабайт ≈ 1024 ексабайта), така ж кількість даних вироблялась кожні 10 днів у 2015 році, і щодня у 2020 році[684].

Великі обсяги даних прийнято називати Big Data — широке розмаїття масивів даних, які не можуть бути належним чином оброблені традиційними додатками через свій величезний обсяг чи складний зміст.

Складність аналізу надвеликих масивів даних полягає у специфіці їхнього збору, поділу, зберігання, передачі, візуалізації та збереження конфіденційності інформації. Під обробкою великих даних часто розуміється застосування про-

---

[682] Ensuring Data Integrity with Hash Codes | Microsoft Docs [Електронний ресурс] — Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/standard/security/ensuring-data-integrity-with-hash-codes

[683] **Автори Федорова Н.В., Демченко О.Е.**

[684] Big Data. [Електронний ресурс] — Режим доступу до ресурсу:https://www.it.ua/ru/knowledge-base/technology-innovation/big-data-bolshie-dannye

гнозної аналітики чи інших передових методів з метою вилучення з множини даних певної корисної інформації[685].

Основна частина. Ідея Інтернету речей (Internet of Things — IoT) полягає в автоматизації процесів повсякденного життя та забезпечення простоти, комфорту та ефективності.

Процес складається з чотирьох етапів:

— велика кількість неструктурованих даних генерується пристроями IoT, які збираються в системі великих даних. Факторами, які в основному залежать, є швидкість, різноманітність та обсяг;

— у системі великих даних, яка в основному є спільною розподіленою базою даних, величезна кількість даних зберігається у файлах великих даних;

— аналіз збережених великих даних IoT за допомогою аналітичних інструментів;

— створення звітів щодо проаналізованих даних[686].

Методи обробки надвеликих масивів даних: — Data Mining — навчання асоціативним правилам, класифікація, кластерний та регресійний аналіз; — краудсорсинг — категоризація та збагачення даних народними силами, тобто із добровільною допомогою сторонніх осіб; — змішування та інтеграція різних даних, таких як цифрова обробка сигналів та обробка природної мови; — машинне навчання (Machine Learning), включаючи штучні нейронні мережі, мережевий аналіз, методи оптимізації та генетичні алгоритми; — розпізнавання образів; — прогнозна аналітика; — імітаційне моделювання; — просторовий та статистичний аналіз; — візуалізація аналітичних даних — малюнки, графіки, діаграми, таблиці.

Основні платформи обробки надвеликих масивів даних — Apache Hadoop з технологією MapReduce та Apache Spark.

MapReduce — це модель розподілених обчислень від компанії Google, яка використовується в технологіях Big Data для паралельних обчислень над дуже великими (до кількох петабайт) наборами даних у комп'ютерних кластерах, та фреймворк для обчислення розподілених завдань на вузлах (node) кластеру.

Метод MapReduce передбачає обробку даних у 3 етапи.

— Map — вхідні дані спочатку поділяються на дрібніші блоки. Потім кожен блок призначається для перетворювача обробки. Результат записується у форматі "ключ-значення" у тимчасове сховище;

— Shuffle — фаза споживає вихідні дані етапу Map. Її завдання — об'єднати відповідні записи з вихідних даних попередньої фази, таким чином, щоб усі дані одного ключа лежали в одному робочому вузлі;

---

[685] Сучасні методи обробки великих даних у великомасштабних системах. [Електронний ресурс] - Режим доступу до ресурсу: http://repo.ssau.ru/bitstream/Matematicheskie-modeli-sovremennyh-ekonomicheskih-processov/SOVREMENNYE-METODY-OBRABOTKI-BOLShIH-DANNYH-V-KRUPNOMASShTABNYH-SISTEMAH-66841/1/%D0%9A%D0%B0%D1%87%D0%B0%D0%BB%D0%BE%D0%B2_%D0%9C%D0%B8%D1%88%D1%83%D1%81%D1%82%D0%B8%D0%BD_%D0%A4%D0%B0%D1%80%D1%85%D0%B0%D0%B4%D0%BE%D0%B2_%D1%81%D1%82%D0%B0%D1%82%D1%8C%D1%8F.pdf

[686] Is It Possible For IoT To Help The Big Data? [Електронний ресурс] - Режим доступу до ресурсу: https://medium.datadriveninvestor.com/is-it-possible-for-iot-to-help-the-big-data-467ec4b6ffb0

— Reduce — на цьому етапі збираються вихідні значення етапу Shuffle. На цьому етапі підбивається підсумок всього набору даних, візуалізація роботи алгоритму зображена на рис. 1.

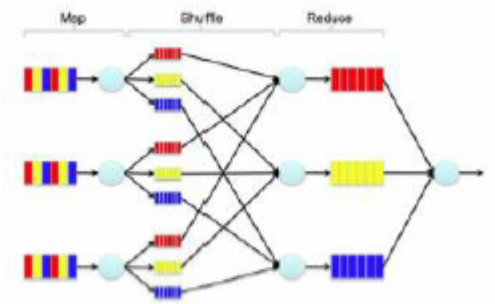

Рисунок 1. Схема функціонування MapReduce-задачі

Лідером у застосуванні парадигми MapReduce, на сьогодні, є технології Apache Hadoop MapReduce та Apache Spark. Ці ж платформи використовуються для організації розподіленої обробки великих обсягів даних.

Apache Hadoop MapReduce — це вільна платформа, для організації обробки великих обсягів даних (виміряється у петабайтах) з використанням парадигми MapReduce.

Спочатку проєкт розроблено на Java в рамках обчислювальної парадигми MapReduce, коли програма поділяється на велику кількість однакових елементарних завдань, що виконуються на розподілених комп'ютерах (вузлах) кластера і зводяться в єдиний результат.

Проєкт складається з основних 4-х модулів:

— Hadoop Common — набір інфраструктурних програмних бібліотек та утиліт, які використовуються в інших рішеннях та родинних проєктах, зокрема, для керування розподіленими файлами та створення необхідної інфраструктури;

— HDFS — розподілена файлова система, Hadoop Distributed File System — технологія зберігання файлів різних серверах даних (вузлах, DataNodes), адреси яких перебувають у спеціальному сервері імен (майстра, NameNode);

— YARN — система планування завдань та управління кластером (Yet Another Resource Negotiator. Фактично, YARN є інтерфейсом між апаратними ресурсами кластера та додатками, що використовують його потужності для обчислень та обробки даних;

— Hadoop MapReduce — платформа програмування та виконання розподілених MapReduce-обчислень з використанням великої кількості комп'ютерів, що утворюють кластер.

Схема функціонування Apache Hadoop зображена на рис. 2.

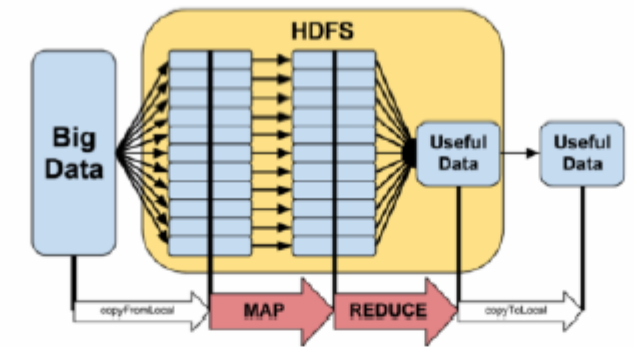

Рисунок 2. Схема функціонування Apache Hadoop

Apache Spark — це Big Data фреймворк з відкритим вихідним кодом для розподіленої пакетної та потокової обробки неструктурованих та слабоструктурованих даних, що входить до екосистеми проєктів Hadoop. У порівняні з попереднім, Spark забезпечує більшу продуктивність при обробці даних у пам'яті.

Класичний MapReduce, компонент Apache Hadoop для обробки даних, проводить обчислення в два етапи. Поки всі процеси етапу Map не закінчаться, процеси Reduce не розпочнуться. Таким чином, технологія MapReduce добре підходить для задач розподілених обчислень у пакетному режимі, але через затримки не може використовуватися для потокової обробки в режимі реального часу. Для розв'язання цієї проблеми було створено Apache Spark.

На відміну від класичного обробника ядра Apache Hadoop з дворівневою концепцією MapReduce на базі дискового сховища, Spark використовує спеціалізовані примітиви для рекурентної обробки оперативної пам'яті. Завдяки цьому багато обчислювальних завдань реалізуються в Spark значно швидше. Окрім цього платформа Apache Spark містить Apache Streaming для роботи з асинхронними стрімами, бібліотеку Mlib для машинного аналізу та GraphX. Структурна модель Spark API зображена на рис. 3[687].

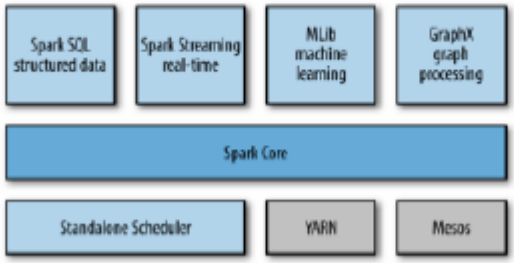

Рисунок 3. Структурна модель Spark API

[687] Большие данные (Big Data) [Електронний ресурс] — Режим доступу до ресурсу: https://www.bigdataschool.ru/wiki/%D0%B1%D0%BE%D0%BB%D1%8C%D1%88%D0%B8%D0%B5-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5-big-data

Висновок. Для обробки великих масивів даних потрібно визначити структуру даних для кращого визначення ключових факторів та розробити алгоритм для аналізу великих обсягів даних за прийнятний час і з допустимою точністю результату відповідно до критерію аналізу. Для отримання статистичних даних, їх потрібно обробити, інтерпретувати та в подальшому застосувати один з методів обробки надвеликих масивів. Реалізацію візуалізації великих масивів даних та результатів обробки можливо здійснити у зручному форматі з використанням різних інструментів і платформ.