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

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Обробка Big Data хмарних обчислень на основі операційних систем»

Виконала:
студентка IV курсу, групи КІ-01
Олещенко Євгенія Олегівна _____

Керівник:
доцент кафедри ШІ, к.т.н.,
доцент Коваленко Анатолій Єпіфанович _____

Консультант з економічного розділу:
професор, д.е.н., проф. кафедри ЕК ФММ,
Шевчук Олена Анатоліївна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ, Кравець П.В. _____

Рецензент:
професор, д.т.н., Бідюк Петро Іванович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студентка _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«31» січня 2024 р.

ЗАВДАННЯ
на дипломну роботу студенту
Олещенко Євгенії Олегівні

1. Тема роботи «Обробка Big Data хмарних обчислень на основі операційних систем», керівник роботи Коваленко Анатолій Єпіфанович, к.т.н., доцент кафедри ШІ, затверджені наказом по університету від «31»травня 2024 р. № 2240-С

2. Термін подання студентом роботи «10» червня 2024 року.

3. Вихідні дані до роботи: файл з аналізом повторюваності слів в вхідному файлі, зазначено слова та кількість разів що вони зустрічаються в тексті, а також час який програма витратила на реалізацію вихідного файлу.

4. Зміст роботи: дослідження Big Data та архітектури хмарних обчислень, його методи, характеристики, безпекову складову, моделі, передумови та проблеми; дослідження моделей обробки розподілених та хмарних обчислень, MPP та MapRecuce; архітектура, програмування, класи та обчислення MapRecuce; засіб розробки в хмарному середовищі Hadoop, бібліотеки та реалізія програми обробки Big Data хмарних обчислень за допомогою моделі MapRecuce в Hadoop; аналіз отриманих результатів; функціонально-вартісний аналіз програмного продукту.

5. Перелік ілюстративного матеріалу: презентація до захисту роботи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук Олена Анатоліївна, професор, д. е. н.		

7. Дата видачі завдання «05» лютого 2024 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Вибір теми і постановка дослідження.	15.04.2024 22.04.2024	Виконано
2	Аналіз актуальності дослідження.	22.04.2024 29.04.2024	Виконано
3	Збір та аналіз літератури.	29.04.2024 10.05.2024	Виконано
4	Формування задачі дослідження.	10.05.2024 15.05.2024	Виконано
5	Розробка програмної частини.	15.05.2024 25.05.2024	Виконано
6	Аналіз та структурізація результатів.	25.05.2024 30.05.2024	Виконано
7	Робота над економічною частиною продукту.	30.05.2024 10.06.2024	Виконано
8	Оформлення пояснювальної записки.	30.05.2024 10.06.2024	Виконано
9	Оформлення презентації для демонстрації.	30.05.2024 10.06.2024	Виконано

Студент Євгенія ОЛЕЩЕНКО

Керівник Анатолій КОВАЛЕНКО

РЕФЕРАТ

Дипломна робота: 94 с., 17 рис., 7 табл., 35 посилань, 1 додаток.

BIG DATA, ХМАРНІ ОБЧИСЛЕННЯ, MAPREDUCE, HADOOP, ХМАРНЕ СЕРЕДОВИЩЕ, IAAS, PAAS, SAAS, АРХІТЕКТУРА ХМАРНИХ ОБЧИСЛЕНЬ, БЕЗПЕКА, ОПТИМІЗАЦІЯ, АНАЛІЗ ВЕЛИКИХ ОБСЯГІВ ІНФОРМАЦІЇ.

Об’єкт дослідження – система обробки та аналізу великих обсягів інформації.

Предмет дослідження – використання хмарного середовища на базі операційної системи для запровадження моделі MapReduce аналізувати великі обсяги інформації.

Мета роботи – дослідження та аналіз використання хмарних обчислень для управління та аналізу великих обсягів даних (Big Data), включаючи основні підходи, класифікацію стратегій, платформи керування Big Data, розподілені файлові системи, зберігання великих даних, застосування MapReduce та оптимізацію.

Актуальність – дослідження застосування хмарних обчислень для управління та аналізу Big Data. Хмарні обчислення надають можливість доступу до обчислювальних ресурсів на вимогу, що сприяє швидкому економічному розвитку та зменшенню технологічних бар'єрів. У світлі зростаючого значення обробки великих обсягів інформації та розвитку хмарних технологій, дослідження, що фокусується на використанні хмарних обчислень для управління Big Data, є надзвичайно актуальним і важливим.

Результат роботи – створена програма точного підрахунку повторюваності слів у заданому текстовому файлі з подальшим зберіганням даних у вихідному файлі. Додана функціональність засікання часу роботи програми. Код був скомпільований в окремому середовищі розробки Visual Studio Code 2, а потім розгорнутий у кластері Hadoop, для отримавання файлу розширення .jar та подальшого його використання. Реалізація та запуск програми

відбувалися в терміналі локального комп’ютера. Для реалізації програми було використано ноутбук Macbook Pro 3-inch, M1, 2020 на базі операційної системи macOS Sonoma версії 14.4.1 (23E224).

ABSTRACT

Master's thesis: 94 p., 17 figures, 7 tables, 35 references, 1 appendix.

BIG DATA, CLOUD COMPUTING, MAPREDUCE, HADOOP, CLOUD ENVIRONMENT, IAAS, PAAS, SAAS, CLOUD COMPUTING ARCHITECTURE, SECURITY, OPTIMIZATION, BIG DATA ANALYSIS.

The object of the study is the system for processing and analyzing large volumes of data.

The subject is the use of a cloud environment based on an operating system to implement the MapReduce model for analyzing large volumes of data.

The purpose is to research and analyze the use of cloud computing for managing and analyzing large datasets (Big Data), including the main approaches, classification strategies, Big Data management platforms, distributed file systems, Big Data storage, the application of MapReduce, and optimization.

The relevance is explores the application of cloud computing for managing and analyzing Big Data. Cloud computing provides on-demand access to computational resources, contributing to rapid economic development and reducing technological barriers. Given the increasing importance of processing large volumes of information and the development of cloud technologies, research focused on using cloud computing for managing Big Data is highly relevant and significant.

The results is a program was created to accurately count the frequency of words in a given text file, with the results stored in an output file. Functionality for timing the program's execution was added. The code was compiled in a separate development environment, Visual Studio Code 2, and then deployed to a Hadoop cluster to obtain a .jar file for further use. The implementation and execution of the program took place in the terminal of a local computer. The program was developed using a MacBook Pro 3-inch, M1, 2020, running on macOS Sonoma version 14.4.1 (23E224).

ЗМІСТ

ВСТУП

Зі швидким розвитком програм соціальних мереж, семантичниго веб-аналізу та біоінформаційного мережевого аналізу, кількість даних для обробки та аналізу стрімко зростає. За останній час Big Data привертають велику увагу наукових кіл, промислового сектору та уряду, адже вони відкривають низку можливостей у різних галузях. Однак з такими можливостями виникають і численні проблеми, насамперед пов’язані з обробкою, аналізом і зберіганням великих обсягів даних для подальшого використання. Найефективніше управління та аналіз великомасштабних даних становлять цікаву, але критичну проблему.

За останні два десятиліття безперервний потік зростання обчислювальної потужності у різних секторах призвело до надзвичайного потоку даних, що повинні оброблятись. Сучасні технології, такі як мережеві та хмарні обчислення, використовуються для забезпечення доступу до великої кількості обчислювальної потужності через агрегування ресурсів і надання єдиного перегляду системи. Хмарні обчислення стають однією з найпотужніших архітектур для виконання великомасштабних і складних обчислень та аналізу, революціонізуючи спосіб абстрагування та використання обчислювальної інфраструктури.

Досліджуванні у цій роботі хмарні обчислення асоціюються з новою парадигмою забезпечення обчислювальної інфраструктури та методами обробки Big Data для всіх видів ресурсів. Одним із рішень проблеми обробки великомасштабних даних є саме хмарні обчислення, адже вони підтримують масивні засоби зберігання та обчислення для обробки Big Data. Управління та обробка Big Data у хмарних обчисленнях досліджує проблеми підтримки обробки великих даних і хмарних платформ як пропоноване рішення.

Big Data обчислення стають доступнішими і зрозумілішими з часом для комп’ютерів. Мультимедійні платформи що використовують інтелектуальний

аналіз даних дозволяють кожному легко досягти цих цілей з мінімальними зусиллями (з точки зору програмного забезпечення, процесора та мережі). Незважаючи на це, підтримка та обробка великомасштабних наборів даних зазвичай є недосяжною для малого бізнесу, що все частіше створює проблеми та перешкоди навіть для великих компаній та інститутів.

РОЗДІЛ 1 BIG DATA ТА ХМАРНІ ОБЧИСЛЕННЯ

1.1 Історія Big Data

Людина змогла не тільки перемогти природне правило виграшу найсильнішого, а й підкорити собі цілий світ завдяки розуму. Наші пращури не лише були спостерігачами за навколишнім середовищем, але й навчалися від нього. Жага до знань ніколи не покидала людство: коли в природі все, що можна було вже дослідити та знайти, було вивчено, людство почало записувати та досліджувати власні дії задля збільшення ресурсів і багатства виду.

Так, наприклад, у 18000 році до н.е. населення давньої Африки підраховувало набуту їжу та інші запаси, вони робили відповідні позначки на паличках і кістках тварин, щоб мати змогу передбачувати та підраховувати об’єми здобичі наперед. Подібні математичні дії робили й у Месопотамії, там занотовували кількість забитої дичини, зберігаючи маркерування глиною. Це і було першим відомим нам і нині збереженням інформації для подальшого аналізу та використання – першим кроком до Big Data обчислень.

Іншими прикладами можна також виокремити деталізовані дані про армію Римської імперії. Big Data там використовували для кращого перерозподілу військових та утворення першої великої бібліотеки в Александрії.

З розвитком галузі математики відкривалися нові шляхи використання зібраних даних та їх подальшого аналізу. У 1663 році Джон Граунт, якого ще називають «батьком статистики», опублікував книгу «Natural and Political Observations Made Upon the Bills of Mortality». У ній Джон аналізував документи про померлих, щоб дослідити та виявити ознаки початку епідемії бубонної чуми. Саме цю роботу прийнято вважати першим прикладом використання методів дослідження статистичних даних і, фактично, першим прикладом застосування Big Data на практиці [1].

Велика кількість інформації для обробки з’явилася лише в 20 столітті н. е. з визначною подією винаходу комп’ютера. Завдяки такій новітній технології збір

та обробка великих масивів даних стали можливими без кропіткої роботи людства. Так, першою серйозною задачею у цій сфері була система зібрання податків громадян для надання справедливих пенсій та виплат, цей проєкт було замовлено американським урядом у 1937 році та реалізовано компанією IBM що створила для цього величезну перфокартну машину.

Першими прикладами використання обробки Big Data були дії що стосувалися саме дешифрування. Таким винаходом була британська машина «Colossus», створена у 1943 році. Це була машина, що була здатна розшифровувати повідомлення німецьких загарбників набагато швидше за людську логіку. Після завершення протистояння розвідка з Америки використовувала вже виключно автоматичну обробку Big Data для аналізу розвідувальних даних. На той час збір та обробка інформації були присутні винятково у великих бізнесах, і то переважно тільки для збереження даних без конкретного подальшого застосування. Розвиток сфери та подальших досліджень був обмежений тодішніми технологіями.

Зберігання інформації було найпростішою проблемою. Навіть з примітивними методами того часу поєднуючи багато малих сховищ можна було створити одне велике. Зростаюча кількість потреб у обсязі зберігання покращувала характеристики кожного елемента: ємність, швидкість і розмір. Так, наприклад, на перфокартах того часу можна було вмістити лише 0.08 кілобайтів, а вже в 1982 році з’явилися компакт-диски з обсягом до 7 мільйонів кілобайтів та жорсткі диски на 10 мегабайтів.

Подолати проблему швидкості обчислень наші предки змогли завдяки суперкомп’ютерам. Саме у 1976 році Сеймур Крей створив перший суперкомп’ютер, що мав назву Cray-1. Його обчислювальна потужність була 133 мегафлопсів. Подальші створені суперкомп’ютери дали можливість аналізувати набагато більше даних ніж було технологічно до цього можливо. На сьогодення, найсерйозніші проєкти у сфері Big Data обчислень використовують суперкомп’ютери.

Останньою проблемою що залишалася був збір інформації, але і вона проіснувала не довго, адже була вирішена у 20 столітті: вже в 1989 році було створено концепт «всесвітньої павутини», яка дозволяла передавати дані між комп'ютерами та серверами в цілому. Зростаюча кількість комп'ютерів у мережі збільшувала кількість інформації у геометричній прогресії. Таким чином, сучасний збір даних здійснюється не тільки з комп'ютерів, але й з пристроїв Internet of Things. Статистично також слід додати, що кількість даних, згенерованих людиною, зросла в 10 разів за минуле десятиліття, а даних від сенсорів – у 50 разів. У 2020 році загальний обсяг даних досяг 44 зетабайтів, а до 2025 року прогнозується збільшення аж до 175 зетабайтів.

Але технології не стоять на місці та невпинно розвиваються, надаючи можливість нашим дослідникам та різним сферам математичного аналізу та прогнозування прагматично використовувати набуті знання та відкривати нові методи обробки та аналізу Big Data.

1.2 Визначення терміну Big Data

Не існує лише одного визначення терміну "Big Data" через різні перспективи та підходи щодо розуміння цього поняття. В незалежності від джерела його визначення, більшість експертів у галузі Big Data обчислень погодяться, що великі дані не можна обмежити лише їх обсягом. Необхідним є також враховування багатьох інших аспектів, що можуть залежати від застосування або джерела цих даних, вони допомагають створити повне визначення поняття "Big Data" [2].

Найбільш загальноприйнятим визначенням «Big Data» можна вважати наступне: величезні обсяги експоненційно зростаючих, різноманітних даних. Даг Лейні з компанії Gartner запропонував модель 3V (Volume, Variety, Velocity) для опису великих даних. Проте, Big Data, як технологія, отримала значний розвиток

лише після появи так званих «відкритих технологій», таких як NoSQL та Hadoop, вони у свою чергу стали ефективними рішеннями для зберігання та обробки досліджуваних великих даних. Відповідно до цього, визначення "Big Data" було розширене до даних, що не можуть бути збережені, керовані та оброблені традиційними системами та технологіями.

Для уточнення технічної характеристики точності визначень, Шон Конноллі ввів додаткові терміни: "транзакції", "взаємодії" та "спостереження". Термін "Транзакції" – описують дані, які вже були зібрані та проаналізовані, "взаємодії" – охоплюють дані, зібрані від об’єктів та людей, а "спостереження" – включають дані зібрані автоматично. Баррі Девлін також додав до вище згаданого визначення, що описуючи Big Data можна їх узагальнити як машинно-генеровані дані, дані від людей та дані, що обробляються процесами.

Якщо говорити з точки зору їх бізнес-ефективності, то транзакційні дані мають обмежене значення та використання, адже вони зазвичай є застарілими на моменті їх аналізу та обробки системою. Натомість нові дані необхідно аналізувати дуже оперативно для отримання чітких прогнозів і своєчасних втручань за потреби.

Окрім вищезгаданих варіацій понять, існують також і інші погляди на тлумачення поняття «Big Data». До прикладу, Метт Асслет вбачає у них насамперед можливість дослідити потенціал зібраної інформації, яка раніше ігнорувалася через обмежені можливості традиційних винайдених систем обчислення. Інші ж дослідники вважають це висловлювання новим терміном для вже існуючих технологій, таких як сучасна бізнес-аналітика. Незалежно від обраного тлумачення, Big Data відкривають двері до безкордонних можливостей. Щоб їх використання було ефективним необхідно розробляти нові або ж модифікувати вже існуючі інструменти та технології обробки Big Data обчислень [3].

Говорячи про сучасні технології, такі як хмарні обчислення, можна стверджувати що вони суттєво спрощують обробку Big Data. Хмарні платформи, такі що засновані на операційних системах, дозволяють агрегувати всі

обчислювальні ресурси та забезпечують єдиний перегляд системи, що робить обробку великих обсягів даних більш доступною та ефективною для використання. За допомогою хмарних обчислень для обробки Big Data ми можемо зберігати та обробляти величезні масиви інформації, що не можуть бути опрацьовані традиційними системами обробки на аналізу.

Хмарні обчислення Big Data стали важливою архітектурою для виконання складних обрахунків, у той же час революціонізуючи спосіб залучання обчислювальної інфраструктури. Такі технології надають нові можливості для обробки Big Data завдяки своїй гнучкості, масштабованості та ефективності. Вони підтримують величезні засоби зберігання та обробки, що є найважливішим для обробки великих обсягів різноманітних даних у реальному часі.

Отже, Big Data відкривають нові вершини для досліджень і бізнесу, а хмарні обчислення на основі операційних систем є важливим інструментом для їх обробки та аналізу. Для найефективнішого використання цих технологій треба постійно вдосконалювати вже існуючі інструменти/системи та розробляти нові методи обробки та аналізу Big Data.

1.3 Архітектура хмарних обчислень

Хмарні обчислення є однією з найобговорюваніших і найбільш використовуваних технологій сьогодення. Вони надають компаніям віртуалізовані сервіси та зберігальні ресурси, доступні в будь-який час, що додає сучасності будь-якій установі. Незалежно від масштабу, будь-яка компанія використовує хмарні обчислення для зберігання інформації та доступу до неї через мережу Інтернет з будь-якої точки нашої планети. Для їх використання важливо глибоко розуміти, як працюють хмарні обчислення і їх внутрішню структуру.

Дизайн будь-якої хмарної системи повинен враховувати такі ключові обмеження, як відкритість, масштабованість, безпеку та інтелектуальний моніторинг. Поточні дослідження обмежень допомагають у розробці нових функцій та стратегій для хмарних систем [4].

Архітектура хмарних обчислень поділяється на дві частини: фронтенд, бекенд.

Дизайн хмарних обчислень використовує архітектури, орієнтовані на сервіси (SOA) і архітектури, керовані подіями (EDA). Архітектура хмарних обчислень включає інфраструктуру клієнта, додатки, сервіси, середовище виконання, зберігання, інфраструктуру, організацію та безпеку.

1.3.1 Фронтенд

Фронтенд хмарних обчислень відноситься до сторони клієнтського використання, включаючи всі інструменти, необхідні їм для доступу до хмарних сервісів. Наприклад, для доступу ми можемо використовувати веб-браузер.

Клієнтська інфраструктура – це компоненти, що включають у собі інтерфейси самого користувача та додатки що необхідні для доступу до хмарних обчислень. Використовуючи хмарні обчислення, можна виконувати складні обчислення в великих масштабах без необхідності оплачувати та мати дорогу апаратну та програмну інфраструктуру чи техніку.

Завдяки соціальним мережам, Інтернету речей та мультимедійним платформам, компанії збирають все більші і більші обсяги даних. Це явище, відоме як "Big Data", швидко набуває популярності в академічній, урядовій та комерційній сферах. Big Data характеризуються великим обсягом, неможливістю зберігання в стандартних реляційних базах даних та високою швидкістю створення, збору та аналізу цих даних [5].

1.3.2 Бекенд

Бекенд відноситься до хмарної інфраструктури постачальника послуг, яка зберігає ресурси, забезпечує їх стан та надає захист. Це включає великі обсяги зберігання, віртуальні додатки та комп'ютери, системи управління трафіком та моделі розгортання.

Додатком називається будь-яке програмне забезпечення або платформа, що доступна клієнтам. Таким чином, кожна послуга налаштовується відповідно до потреб відповідного клієнта.

1. Середовище виконання забезпечує платформу для виконання та середовище виконання віртуальної машини.
2. Інфраструктура – це сервери, сховища, мережеві пристрої, програмне забезпечення віртуалізації та інше хмарне апаратне і програмне забезпечення.
3. Зберігання – гнучке та масштабоване управління збереженими даними.
4. Управління – управління додатками, зберіганням, середовищем виконання та іншими заходами безпеки.
5. Інтернет – зв'язок що є необхідним для взаємодії між фронтендом і бекендом.
6. SaaS (Software as a Service), PaaS (Platform as a Service) та IaaS (Infrastructure as a Service) – найбільш поширені моделі хмарних сервісів.
7. Безпека – впровадження різних технік безпеки для захисту ресурсів, систем, даних та інфраструктури.

Хмарні обчислення являються одним із найважливіших нововведень у сучасних інформаційних технологіях та бізнес-додатках. Вони забезпечують віртуалізовані ресурси, паралельну обробку та підвищений рівень безпеки.

Хмарні обчислення також спрощують управління, знижують витрати на утримання інфраструктури та розширюють можливості доступу для користувачів. З розвитком хмарних додатків, кількість даних, які вони генерують та використовують, значно зросла.

Першими користувачами рішень для великих даних на базі хмари стали IBM Bluemix, Microsoft Azure та Amazon AWS. Віртуалізація, одна з технологій, що лежить в основі хмарних обчислень, є базою для різних функцій платформи, таких як доступ, зберігання, аналіз і управління даними. Віртуалізація дозволяє ефективно використовувати ресурси, ізолюючи апаратне забезпечення, що робить комп'ютери більш ефективними, корисними та масштабованими.

1.3.3 Хмарні обчислення та великі дані

Хмарні обчислення та великі дані дозволяють операторам виконувати розподілені запити по численних наборах даних і швидко отримувати результати за допомогою стандартних обчислювальних технік. Хмарні обчислення базуються на платформі Hadoop, яка дозволяє обробляти дані, розподілені по багатьох комп'ютерах. Інформація зберігається за допомогою технології розподіленого зберігання, створеної на основі хмарних обчислень [6].

1.3.4 Безпека хмарних обчислень

Як одна з найперспективніших та найскладніших комп'ютерних технологій, безпека хмарних обчислень забезпечує широкий спектр варіантів комунікації для користувачів. З підвищенням інновацій у сфері даних необхідно посилити безпеку. Дані в центрах обробки інформації зберігаються та керуються, але можуть бути вразливі до різних атак. Тому необхідно мати безпечний спосіб змінювати дані на серверах, зберігаючи конфіденційність клієнтів. Дані в хмарі захищаються різними методами шифрування.

1.3.5 Характеристики хмарних обчислень

Хмарна система має різні сервіси, які використовують ресурси по-різному, а динамічні вимоги до ресурсів часто змінюються. Системи хмарних обчислень повинні гарантувати, що динамічно надані ресурси надійні і не заважають ефективному використанню, навіть якщо системи сервісу виходять з ладу. У великих системах важко зібрати точні дані про помилки через їх розмір і складність [7].

1.3.6 Вплив архітектури, орієнтованої на сервіси (SOA), на організацію

Управління та використання апаратних і програмних ресурсів у хмарній системі є критичним. Архітектура, орієнтована на сервіси (SOA), дозволяє бізнесам ділитися своїми фізичними та неприхованими IT-інфраструктурами. Це дозволяє зменшити початкові інвестиції та операційні витрати. Хмарні

обчислення впливають на різні аспекти діяльності, включаючи е-комерцію. Окрім моделей і методів розгортання, ця структура включає три основні категорії сервісів: платформа як сервіс (PaaS), програмне забезпечення як сервіс (SaaS) та інфраструктура як сервіс (IaaS) [8].

1.3.7 Аспекти хмарних обчислень

Завдяки розподіленому характеру хмарних систем вони можуть забезпечувати високу доступність і масштабованість. Однак складність управління такою системою вимагає ретельного проєктування та впровадження механізмів контролю.

Резюмуючи, хмарні обчислення є потужною технологією, що дозволяє компаніям ефективно використовувати ресурси та масштабувати свої операції. Інтеграція великих даних, безпеки та архітектури, орієнтованої на сервіси, робить хмарні обчислення ключовим елементом сучасних IT-інфраструктур.

1.3.8 Моделі хмарних обчислень

Існують три моделі хмарних сервісів: від віртуального обладнання до користувачів, орієнтованих на конкретні додатки. Технологічний шар, відомий також як інфраструктура як сервіс (IaaS), надає користувачам віддалений доступ до сховищ і серверів, які можуть використовуватися для аналізу даних. Клієнти можуть збільшувати або зменшувати обсяг свого віртуального сховища за потреби через Інтернет. Хмарне сховище просте у використанні і коштує значно менше, ніж традиційний жорсткий диск. Крім того, цей метод є дуже безпечним. Шар додатків, або платформа як сервіс (PaaS), дозволяє розробникам створювати

додатки. PaaS дозволяє поєднувати дані з різних внутрішніх та зовнішніх офісів, знижуючи витрати на володіння. Це лише деякі з послуг SaaS (програмне забезпечення як сервіс), які широко використовуються в будівельній галузі.

Багато додатків для масштабних експериментів організовуються в хмарі, і очікується, що їх кількість зростатиме, оскільки обчислювальні ресурси на локальних серверах стають дефіцитними, а капітальні витрати знижуються. Обсяг інформації, що створюється і розповсюджується експериментами, продовжує зростати. Користувачі можуть використовувати хмарні ресурси і встановлювати свої додатки, якщо постачальники хмарних сервісів інтегрують паралельні фреймворки обробки даних у свої пропозиції. За визначенням, основна концепція хмарних обчислень полягає в можливості отримати доступ до мережі на вимогу, використовуючи різні попередньо налаштовані обчислювальні ресурси, які можуть бути швидко надані та звільнені з мінімальними зусиллями з боку управління або взаємодії з постачальниками послуг. Використання хмарних обчислень надає такі переваги, як швидший розвиток економіки і зниження технологічних перешкод. Завдяки хмарним обчисленням організації можуть зосередитися на своїх основних операціях, не турбуючись про побічні проблеми, такі як організація, адаптивність і доступність ресурсів. Завдяки утилітарній моделі хмарних обчислень та величезній кількості обчислень, інфраструктур і сховищ, які зараз доступні, дослідники можуть проводити свої тести в дуже сприятливих умовах. Найпоширенішими моделями сервісів у хмарі є PaaS, SaaS та IaaS [9].

1.3.9 Приклади бізнес-моделей на базі хмари

Хмарні послуги можуть бути налаштовані як публічні, приватні, комунальні та гібридні. Моделі розгортання відрізняються в залежності від того, хто і як може використовувати сервіс та як вони можуть отримати до нього

доступ. Клієнти підключаються до хмарного сервісу через Інтернет, використовуючи публічну хмару. Дані одного клієнта обробляються разом з даними інших клієнтів у мультиорендній системі. Публічні хмарні додатки можуть бути розміщені в одному центрі обробки даних або розподілені.

Постачальник хмарних послуг відповідає за обслуговування та управління всією IT-інфраструктурою. Для більшості малих підприємств публічна хмара є найкращим варіантом. Лише одна організація може використовувати приватну хмару. Хмарна інфраструктура розміщується на місці, і доступ до неї можливий тільки через корпоративну внутрішню мережу. Приватна хмара є ідеальним варіантом для зберігання конфіденційних даних на спеціалізованій системі, наприклад, національній інфраструктурі. Великі корпорації та урядові агентства повинні використовувати цю стратегію для зберігання конфіденційних даних.

Деякі організації спільно використовують хмарну інфраструктуру в моделі комунальної хмари. Лише учасники організацій мають доступ до комунальної хмари. У гібридному хмарному розгортанні можуть співіснувати кілька типів розгортання в одному середовищі. Приватні хмари можуть використовуватися для фінансової звітності, тоді як публічні хмари можуть використовуватися для телематики та інших неконфіденційних даних.

1.3.10 Проблемна область

Велика хмарна платформа використовує великі дані та хмарні обчислення. Зі збільшенням кількості та складності даних і схем виникають проблеми з управлінням даними, їх обміном, конфіденційністю, збоями програмного забезпечення та безпекою. Хмарні сервіси даних дозволяють повільно вирішувати ці проблеми, покращуючи швидкість хмарних обчислень. Хмарний сервіс даних може зберігати дані паралельно, дозволяючи одночасно ділитися ними. У цій ситуації постачальник послуг повинен підтримувати цілісність,

конфіденційність і захист даних. Зберігання, отримання та обмін даними є основними причинами занепокоєння щодо хмарних обчислень. Багато дослідників запропонували різні фреймворки, стратегії та моделі для вирішення проблем програмного забезпечення. Дослідження повинно бути ретельно вивчене. В кінцевому підсумку необхідно розробити фреймворк і рекомендації для існуючих досліджень.

1.3.11 Передумови

Основні послуги хмарних обчислень забезпечують функціональність для коректної роботи на стороні клієнта. Тому усі збої програмного забезпечення та питання безпеки стосуються цих послуг. Певні обмеження цих послуг слід враховувати при обговоренні збоїв програмного забезпечення та питань безпеки.

Швидке розширення платформ хмарних обчислень за розміром і обчислювальною потужністю призвело до значних проблем з управлінням і підтримкою систем: стає все важче виявляти несправності в апаратному та програмному забезпеченні. Коли операція може керувати збоями, потрібне постійне планування надійності. Великомасштабні хмарні системи мають бути сплановані та впроваджені для забезпечення високої надійності системи, гнучкості в роботі та автоматичного відновлення після збоїв у рамках цієї стратегії (незалежно від місця розташування). Все більше людей звикають зберігати свої комерційні та організаційні дані у хмарі.

Операційна та сервісна досконалість, міцна безпека, стратегія довгострокового виживання та конкурентні переваги є деякими з бізнес-цілей, які вимагають хмарні обчислення. Від автоматизації ручної праці до фундаментальних змін у бізнес-моделях, які стали можливими завдяки Інтернету, багато технологічних досягнень привели до цифрової трансформації

підприємств. Потрібно створити таку структуру для вирішення збоїв програмного забезпечення та питань безпеки.

Основні області, в яких розробляється модель, включають аналіз великих даних, хмарні обчислення, мобільні технології та інтерфейси програмування додатків (API).

Необхідно створити таку модель, яка б включала характеристики великих даних, платформу хмарних обчислень для знань, зв’язків та великих даних на першому етапі, який називається "Big Data Підприємства". Аналітика розглядається як спосіб покращення результатів компаній. Три основні послуги хмарних обчислень сприяють потоку своїх послуг у хмарі. Це SaaS, IaaS та PaaS. Безпека даних повинна підтримуватися на рівні програмного забезпечення як сервера. Кожна послуга потребує частого обслуговування, щоб зменшити ймовірність збоїв та підвищити безпеку і конфіденційність даних. Аналогічно, у PaaS та IaaS дані потрібно захищати. Всі послуги обговорені нижче з їх основними обмеженнями (підрозділи 1.3.12–1.3.14).

З іншого боку, потрібна єдина інфраструктура, яка могла б обробляти та аналізувати широкий спектр великих даних, згенерованих сервісами. Це пояснюється тим, що обсяг зібраних даних постійно зростає.

Великі дані, згенеровані сервісами, можуть покращити продуктивність системи різними способами. В результаті, для підвищення продуктивності та зниження витрат, організації звертаються до “Big Data Як Послуга” (BDaaS – Big Data as a Service, також відомих як інфраструктура як послуга, платформа як послуга або програмне забезпечення як послуга) для надання загальних сервісів великих даних (таких як доступ до згенерованих сервісами великих даних та результатів аналітики даних). Все це включено до BDaaS.

1.3.12 SaaS: Програмне забезпечення як послуга

Власники бізнесу, які хочуть використовувати хмарні технології, надають перевагу SaaS (Програмне забезпечення як послуга, часто відоме як "хмарні додатки"). SaaS дозволяє клієнтам отримувати доступ до додатків, хостованих третьою стороною через Інтернет. Власникам бізнесу не потрібно завантажувати чи встановлювати нічого на своїх комп'ютерах для використання більшості програм SaaS. Основними мінусами цієї програми виділяють:

- безпека даних. Центри обробки даних на бекенді додатків SaaS можуть мати справу з великими обсягами даних для виконання основних функцій програмного забезпечення. Рішення SaaS, розміщене в публічній хмарі, може поставити під загрозу безпеку та угоди, а також значні витрати на переміщення масивних робочих навантажень даних, якщо передається конфіденційна інформація компанії;
- відсутність підтримки інтеграції. Багато підприємств вимагають глибоких інтеграцій з локальним програмним забезпеченням, інформацією та сервісами. Організації можуть бути змушені використовувати внутрішні ресурси для створення та управління інтеграціями, якщо постачальник SaaS надає лише мінімальну підтримку. Чим складніші інтеграції, тим менш корисний додаток SaaS або інші залежні сервіси;
- інтероперабельність. Якщо додаток SaaS не побудований для інтеграції з іншими додатками та сервісами, це може бути серйозною проблемою. Сервіси SaaS можуть бути не завжди придатними для організацій, які потребують створення або оптимізації своїх систем інтеграції;
- залежність від постачальника. Підписатися на сервіс може бути легко завдяки постачальнику, але вийти з сервісу може бути складно. Наприклад, інформація може бути теоретично або економічно неконвертованою між додатками SaaS від різних

постачальників без значних витрат або необхідності в інженерній переробці. Це може бути пов'язано з кількома причинами. Не кожен постачальник використовує стандартні протоколи та інструменти прикладного програмного інтерфейсу, а структури можуть бути релевантними для певних бізнес-активностей;

- кастомізація. Додатки SaaS дозволяють дуже мало персоналізації. Без універсального рішення споживачі можуть бути обмежені можливостями, продуктивністю та іншими інтеграціями постачальника. Широкий діапазон кастомізації є перевагою використання локальних наборів розробки програмного забезпечення (SDK);
- обмеження функцій. Організації, які хочуть додаткові функції від програм SaaS, можуть змушені відмовитися від таких речей, як безпека, вартість, продуктивність та інші бізнес-стандарти. Перехід до інших постачальників або сервісів для задоволення нових вимог до функцій може бути неможливим через залежність від постачальника, цінові міркування або питання безпеки. Ці фактори можуть зробити перехід неможливим;
- оскільки постачальник контролює сервіс SaaS, відповідальність за підтримку продуктивності та безпеки сервісу лежить на постачальнику. Навіть при наявності відповідних угод про рівень обслуговування (SLA), які захищають від незапланованого та запланованого обслуговування, кібератак та збоїв мережі, продуктивність додатку SaaS може бути негативною;
- відсутність контролю. При використанні рішень SaaS необхідно відмовитися від деякого контролю на користь стороннього постачальника послуг. Ці обмеження стосуються програми та її даних і управління (у плані версій, оновлень і зовнішнього вигляду). Через особливості та функціональність, запропоновані

сервісом SaaS, користувачам може знадобитися переосмислити існуючі підходи до захисту та управління даними.

1.3.13 PaaS: Платформа як послуга

Більш типове використання терміну вказує на надання послуг хмарних платформ, а не хмарних обчислень. PaaS надає каркас для створення власних додатків. Організація або сторонній постачальник може керувати серверами, сховищами та мережею, але додатки залишаються у відповідальності розробників.

Обмеження та питання, пов'язані з PaaS:

- безпека даних. Адміністрації можуть використовувати результати PaaS для запуску своїх додатків та сервісів; проте, збереження даних на серверах, що належать стороннім постачальникам, викликає питання безпеки. Наприклад, альтернативи забезпечення безпеки організації, швидше за все, обмежені, якщо клієнт має певні обмеження хостингу;
- інтеграції. Дані центри на місці та хмари на відстані можуть бути складніше підключити до пропозицій PaaS, обмежуючи додатки та сервіси. Може бути складно підключити легасі IT-системи до хмари, якщо не всі вони готові для хмарних технологій;
- залежність від постачальника. Рішення щодо певного рішення PaaS може вже не бути важливими через зміни в бізнесі та технологіях. Може бути неможливо перейти до іншого постачальника PaaS без негативного впливу на фінансовий стан компанії, якщо постачальник не надає політику міграції, яка була б легкою для слідування;

- кастомізація легасі систем. Якщо організації використовують стару версію програми чи сервісу, PaaS може не бути ідеальним вибором для організації. Легасі системи можуть потребувати певних оновлень та налаштувань для роботи з PaaS. Через потенціал кастомізації для створення складних IT-систем, PaaS є витратою, яку слід уникати;
- проблеми часу виконання. Існують ситуації, коли рішення PaaS можуть не бути найкращим варіантом для мов та фреймворків, які обирають організації, та обмежень, пов'язаних з конкретними додатками та сервісами. Залежно від платформи, клієнти можуть визначати свої залежності, або це може бути неможливим;
- обмеження з точки зору функціонування. У зв'язку з обмеженнями платформи у можливостях, доступних для кінцевих користувачів, рішення PaaS можуть не бути найкращим варіантом для хмарних операцій, які включають автоматизацію управління. Постачальники PaaS намагаються зробити свої послуги більш легкими у керуванні, масштабовані та ефективні, звільняючи кінцевих користувачів від відповідальності за щоденні операції.

1.3.14 IaaS: Інфраструктура як сервіс

Якщо організація шукає хмарні обчислювальні послуги, які є високо масштабованими та автоматизованими, то вони повинні використовувати інфраструктуру як сервіс (IaaS). Хмарні обчислювальні послуги, такі як інфраструктура як сервіс (IaaS), цілком самообслуговуванні. Організації повинні бути уважними до різниць між моделями хмари, щоб прийняти найкраще рішення та знайти оптимальну модель для розв'язання питань безпеки. Хмарна послуга може задовольнити цілі організацій, чи то вони шукають опції зберігання, просту платформу для створення власних додатків або повне управління інфраструктурою компанії без її підтримки. Перехід організацій та технологій до хмари наближається незалежно від вподобань організацій [10].

1.4 Питання відмов програмного забезпечення

Хмара складається з декількох програм, які працюють паралельно, щоб гарантувати її надійність. Усі програми виконують свою функцію та мають певну відповідальність. Якщо будь-який компонент хмари перестає працювати з будь-якої причини, такої як часткова або повна відмова програмного забезпечення, це впливає на загальну продуктивність хмарних обчислень [11].

Часткова відмова програмного забезпечення – якщо будь-який компонент програмного забезпечення перестає працювати. Повна відмова програмного забезпечення – коли повна програма не відповідає. У хмарі балансування навантаження має кілька переваг, зокрема тоді, коли обсяг трафіку збільшується, сервери можуть перезавантажиться, що призводить до відключень. З іншого боку, балансування навантаження хмари ефективно розподіляє навантаження по численних серверах та мережах. Навантаження передається на інший серверний

центр, коли один з серверів вмирає або стає непридатним для використання. В результаті сайт залишається функціональним навіть під час періодів великого трафіку. Компаніям потрібні правильні машини для керування навантаженням, оскільки хмарні обчислення та обмін даними та інформацією через Інтернет стають популярнішими. Низька вартість балансувальників хмари робить їх чудовим варіантом. Це допомагає їм швидше виводити матеріал, бути адаптивними та завжди доступними.

1.5 Розподіленні та хмарні обчислення

Розподіленими обчисленнями називають стратегію підвищення продуктивності якогось окремого комп'ютера за рахунок його об'єднання в кластер з іншими пристроями та системами, які називаються воркерами. Такий кластер має значну обчислювальну потужність, що приблизно дорівнює сумарній потужності кожного воркера якщо розглядати їх окремо. Цей термін також може застосовуватися для опису роботи багатоядерного процесора, у якому вмонтовані ядра виконують задане завдання спільно. У порівнянні з централізованими обчисленнями, у якому єдиний максимально оснащений воркер, розподілені обчислення переважають у своїй масштабованості та надійності. Масштабованість дозволяє просто розширювати систему, додавши нові воркери, тоді як надійність полягає в тому, що воркери можуть замінювати один одного у разі виникнення проблем.

Хмарні обчислення, у свою чергу, є логічним розвитком концепції розподілених обчислень. Вони дозволяють компаніям не заморочуватися на створенні власної інфраструктури для кластеру або його технічному обслуговуванні, а замість цього орендувати комп'ютерні ресурси в Інтернеті, коли це необхідно. Такий підхід дозволяє зберігати ресурси та зосереджуватися на бізнесових завданнях, замість інфраструктурних питань [12].

1.6 Massively parallel processing (MPP)

MPP (масово паралельна обробка) – це узгоджена обробка програми кількома процесорами, які працюють над різними частинами програми, кожен із яких використовує власну операційну систему та пам'ять. Зазвичай такі процесори спілкуються за допомогою інтерфейсу обміну повідомленнями. В деяких реалізаціях до 200 або більше процесорів можуть працювати над одним застосунком. Спеціальна організація шляхів передачі даних, відома як "інтерконект", дозволяє відправляти повідомлення між процесорами. Зазвичай налаштування MPP є складнішим і вимагає розподілу спільної бази даних між процесорами та призначення завдань кожному з них. Система MPP також відома як "слабкозв'язана" або "з повним розподілом ресурсів".

Система MPP вважається кращою за симетрично паралельну систему (SMP) для додатків, які дозволяють паралельний пошук у багатьох базах даних. До таких додатків належать системи підтримки прийняття рішень та сховища даних [13].

1.7 MapReduce

MapReduce – це модель веб-програмування для масштабованої обробки даних у великих кластерах на великих наборах даних. Ця модель переважно використовується у додатках для веб-пошуку та хмарних обчислень. Користувач задає функцію Map для генерації набору проміжних пар "ключ/значення". Потім користувач застосовує функцію Reduce для об'єднання всіх проміжних значень з однаковим ключем. MapReduce добре маштабується для досягнення високого

рівня паралелізму на різних рівнях завдань. Типовий процес обчислень MapReduce може обробляти терабайти даних на десятках тисяч або більше клієнтських машин. Сотні програм MapReduce можуть виконуватися одночасно; насправді, тисячі завдань MapReduce щоденно виконуються на кластерах Google.

MapReduce – це програмна платформа, яка підтримує паралельні та розподілені обчислення на великих наборах даних. Ця платформа абстрагує потік даних під час виконання паралельної програми на розподіленій обчислювальній системі, надаючи користувачам два інтерфейси у вигляді двох функцій: Map і Reduce. Користувачі можуть перевизначати ці дві функції для взаємодії з потоком даних та маніпулювання ним під час виконання своїх програм. У цій платформі "значення" у парі (ключ, значення) є фактичними даними, а "ключ" використовується контролером MapReduce для управління потоком даних. Програмна платформа MapReduce надає шар абстракції з потоком даних та контролю, приховуючи реалізацію всіх кроків потоку даних, таких як розподіл даних, маппінг, синхронізація, комунікація та планування. Незважаючи на те, що потік даних у таких платформах заздалегідь визначений, шар абстракції надає два добре визначені інтерфейси у вигляді двох функцій: Map і Reduce, які можуть бути перевизначені користувачем для досягнення конкретних цілей [14].

Отже, користувач спочатку перевизначає функції Map і Reduce, а потім викликає надану функцію MapReduce (Spec, & Results) з бібліотеки для запуску потоку даних. Функція MapReduce (Spec, & Results) приймає важливий параметр – об'єкт специфікації (Spec). Цей об'єкт спочатку ініціалізується в програмі користувача, а потім користувач пише код для його заповнення іменами вхідних та вихідних файлів, а також іншими опціональними параметрами налаштування. Цей об'єкт також заповнюється іменами функцій Map і Reduce для ідентифікації цих користувацьких функцій у бібліотеці MapReduce.

Загальна структура програми користувача, що містить функції Map, Reduce та Main:

Map Function (... .) {

... ... }

Reduce Function (... .) {

... ... }

Main Function (... .) {

Initialize Spec object

... ...

MapReduce (Spec, & Results)

}

Map і Reduce є двома основними підпрограмами, які будуть викликані для реалізації бажаної функції, що виконується у головній програмі [15].

1.8 Постановка задачі

Задачею дипломної роботи є розробка програми для аналізу текстів великого маштабу. Крім того, буде проведено дослідження та аналіз використання хмарних обчислень для управління та аналізу великих обсягів даних (Big Data) з врахуванням основних підходів, класифікації стратегій, платформ керування Big Data, розподілених файлових систем, зберігання великих даних, застосування MapReduce та оптимізації процесів аналізу.

Висновки до розділу 1

Історія розвитку Big Data демонструє прагнення людства до знань та прогресу, що сприяє збільшенню ресурсів та добробуту. Це свідчить про довготривалу і постійну еволюцію інтересу до обробки великих обсягів даних.

Обробка великомасштабних наборів даних може бути складною для бізнесу, але сучасні технології Big Data та хмарні обчислення значно полегшують цей процес, дозволяючи досягати поставлених цілей з мінімальними зусиллями. Це підкреслює необхідність використання передових технологій для оптимізації обробки даних у бізнес-середовищі.

Big Data набуває все більшої популярності у різних сферах завдяки своїм можливостям обробляти великі обсяги даних з високою швидкістю. Це вказує на те, що використання Big Data стає ключовим елементом для ефективного управління даними в сучасному світі.

Хмарні обчислення значно спрощують роботу з Big Data, забезпечуючи ефективне та доступне зберігання і обробку великих масивів інформації. Це підкреслює важливість впровадження хмарних технологій для оптимізації роботи з великими обсягами даних.

Таким чином, розглянутий розділ дипломної роботи надає важливі відомості про історичний розвиток та сучасний стан Big Data і хмарних обчислень, акцентуючи увагу на їх важливості та перевагах у сучасному світі.

РОЗДІЛ 2 МОДЕЛЬ MAPREDUCE

MapReduce та HDFS є основними складовими Hadoop, які роблять його потужним і ефективним. MapReduce – це модель програмування, яка використовується для ефективної паралельної обробки великих наборів даних у розподіленому середовищі. Дані спочатку розподіляються, а потім об'єднуються для отримання кінцевого результату. Бібліотеки MapReduce написані багатьма мовами програмування з різними оптимізаціями.

Hadoop – це відкрита платформа для розробки та запуску розподілених додатків, що обробляють великі набори даних. Вона здобула велику популярність як у промисловості, так і в наукових колах. Hadoop застосовується для веб-індексації, наукового моделювання, аналізу соціальних мереж, виявлення шахрайства, рекомендаційних систем, таргетування реклами, аналізу загроз, моделювання ризиків тощо. Hadoop є ключовим елементом інфраструктури хмарних обчислень і використовується такими компаніями, як Yahoo, Facebook, IBM, LinkedIn та Twitter.

Мета MapReduce у Hadoop полягає в тому, щоб розподілити кожне завдання і зменшити їх до еквівалентних задач для зниження навантаження на кластерну мережу і зменшення обчислювальної потужності. Завдання MapReduce поділяється на дві основні фази: Map Phase і Reduce Phase [16].

2.1 Архітектура MapReduce

Компоненти архітектури MapReduce зображено на рис. 2.1:

- клієнт: клієнт MapReduce подає завдання для обробки. Може бути багато клієнтів, які постійно відправляють завдання до менеджера Hadoop MapReduce;

- завдання: завдання MapReduce складається з багатьох менших підзадач, які клієнт хоче виконати;
- головний вузол Hadoop MapReduce: він ділить завдання на окремі частини;
- частини завдання: це підзадачі, які утворюються після розподілу основного завдання, результат усіх частин завдання об'єднується для отримання кінцевого результату;
- вхідні дані: набір даних, який подається для обробки;
- вихідні дані: кінцевий результат після обробки.

У MapReduce клієнт подає завдання певного розміру до головного вузла Hadoop MapReduce. Головний вузол розподіляє це завдання на подальші еквівалентні частини. Ці частини завдання стають доступними для завдань Map і Reduce. Програма для завдань Map і Reduce створюється відповідно до вимог конкретного випадку використання. Розробник пише логіку для виконання цих вимог. Вхідні дані подаються до завдання Map, яке генерує проміжні пари ключ-значення як результат. Ці ключ-значення передаються до завдання Reduce, а кінцевий результат зберігається у HDFS. Кількість завдань Map і Reduce може варіюватися залежно від потреби. Алгоритм для Map і Reduce розробляється з мінімальною часовою та просторовою складністю [17].

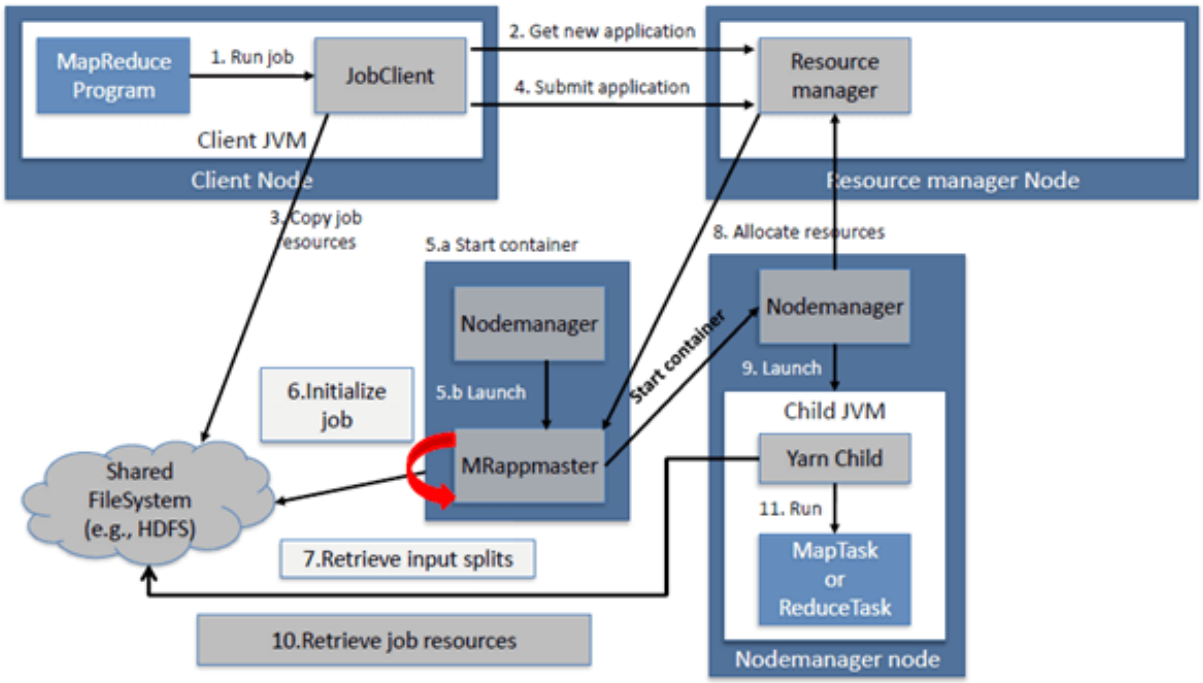


Рисунок 2.1 – Архітектура MapReduce[1]

2.2 Фази MapReduce

1) Map: основна мета фази Map – перетворити вхідні дані на пари ключ-значення. Вхід до фази Map може бути парою ключ-значення, де ключ – це ідентифікатор, а значення – це фактичне значення. Функція Map() виконується в своїй пам'яті над кожною парою ключ-значення і генерує проміжні пари ключ-значення, які слугують вхідними для функції Reduce [18].
2) Reduce: проміжні пари ключ-значення, які є вхідними для Reducer, сортуються і передаються до функції Reduce(). Reducer агрегує або

[1] Джерело зображення: https://static.javatpoint.com/hadooppages/images/mapreduce-introduction1.png

групує дані на основі пари ключ-значення згідно з алгоритмом, написаним розробником [19].

2.3 Взаємодія Job Tracker і Task Tracker з MapReduce

Завдання Job Tracker – керувати всіма ресурсами і завданнями в кластері, а також розкладати кожну карту на Task Tracker, який працює на тому ж вузлі даних, оскільки в кластері може бути сотні вузлів даних.

Task Tracker – це фактичні виконавці, які працюють за інструкціями Job Tracker. Task Tracker розгортається на кожному вузлі в кластері, виконуючи завдання Map і Reduce за інструкціями Job Tracker.

Існує також важливий компонент архітектури MapReduce, відомий як Job History Server. Це демон-процес, який зберігає історичну інформацію про завдання або застосунок, такі як логи, які генеруються під час або після виконання завдання, що зберігаються на Job History Server [20].

2.4 Програмування MapReduce

Хоча інтерфейси Map і Reduce значно спростили створення розподілених додатків у Hadoop, важко висловити широкий спектр логіки в парадигмі програмування MapReduce. Ітеративні процеси є прикладом логіки, яка не підходить для MapReduce. Зазвичай дані не зберігаються в пам'яті, а ітеративна логіка обробляється шляхом ланцюгового зв'язування додатків MapReduce, що призводить до збільшення складності.

Завдання MapReduce зберігають мало даних у пам'яті, оскільки не мають концепції розподіленої пам'яті для користувацьких даних. Дані повинні бути зчитані і записані в HDFS. Більш складні додатки MapReduce вимагають ланцюгового зв'язування менших завдань MapReduce. Оскільки дані не можуть

передаватися між цими завданнями, необхідне спільне використання даних через HDFS, що створює вузьке місце у процесі [21].

2.5 Орієнтованість на Java

MapReduce базується на Java, тому найбільш ефективний спосіб написання додатків для нього – використання Java. Код повинен бути скомпільований в окремому середовищі розробки, а потім розгорнутий у кластері Hadoop. Такий стиль розробки не широко прийнятий аналітиками даних чи науковцями з даних, які звикли до інших технологій, таких як SQL або інтерпретовані мови, як-от Python.

MapReduce може викликати логіку Map/Reduce, написану іншими мовами, такими як C, Python або Shell Scripting, однак це робиться шляхом запуску системного процесу для виконання цих програм. Ця операція створює накладні витрати, що впливають на продуктивність завдання [22].

Переваги MapReduce:

- масштабованість;
- гнучкість;
- безпека та автентифікація;
- швидка обробка даних;
- дуже проста модель програмування;
- доступність і стійкість.

2.6 Класи Mapper, Reducer та Job

У MapReduce клас Mapper відповідає за перетворення вхідних пар ключ-значення у набір проміжних пар ключ-значення. Він трансформує вхідні записи

у проміжні записи. Ці проміжні записи, пов'язані з певним ключем, передаються до Reducer для отримання кінцевого результату.

Методи класу Mapper:

- `void cleanup(Context context)` – цей метод викликається тільки один раз наприкінці завдання;
- `void map(KEYIN key, VALUEIN value, Context context)` – цей метод може бути викликаний тільки один раз для кожної пари ключ-значення у вхідному наборі;
- `void run(Context context)` – цей метод можна перевизначити для контролю виконання Mapper;
- `void setup(Context context)` – цей метод викликається тільки один раз на початку завдання.

У MapReduce клас Reducer відповідає за зменшення набору проміжних значень. Його реалізації можуть отримати доступ до конфігурації завдання через метод JobContext.getConfiguration().

Методи класу:

- `void cleanup(Context context)` – цей метод викликається тільки один раз наприкінці завдання;
- `void map(KEYIN key, Iterable<VALUEIN> values, Context context)` – цей метод викликається тільки один раз для кожного ключа;
- `void run(Context context)` – цей метод можна використовувати для контролю завдань Reducer;
- `void setup(Context context)` – цей метод викликається тільки один раз на початку завдання.

Клас Job використовується для налаштування завдання та його подання. Він також контролює виконання та перевіряє стан завдання. Після подання завдання метод set викликає IllegalStateException.

Методи класу Job:

- `Counters getCounters()` – використовується для отримання лічильників для завдання;
- `long getFinishTime()` – використовується для отримання часу завершення завдання;
- `Job getInstance()` – використовується для створення нового завдання без кластера;
- `Job getInstance(Configuration conf)` – використовується для створення нового завдання без кластера з наданою конфігурацією;
- `Job getInstance(Configuration conf, String jobName)` – використовується для створення нового завдання без кластера з наданою конфігурацією та ім'ям завдання;
- `String getJobFile()` – використовується для отримання шляху до конфігурації поданого завдання;
- `String getJobName()` – використовується для отримання вказаного користувачем імені завдання;
- `JobPriority getPriority()` – використовується для отримання функції планування завдання;
- `void setJarByClass(Class<?> c)` – використовується для встановлення jar файлу, надаючи ім'я класу з розширенням .class;
- `void setJobName(String name)` – використовується для встановлення вказаного користувачем імені завдання;
- `void setMapOutputKeyClass(Class<?> class)` – використовується для встановлення класу ключа для вихідних даних map;
- `void setMapOutputValueClass(Class<?> class)` – використовується для встановлення класу значення для вихідних даних map;
- `void setMapperClass(Class<? extends Mapper> class)` – використовується для встановлення Mapper для завдання;
- `void setNumReduceTasks(int tasks)` – використовується для встановлення кількості завдань reduce;

- `void setReducerClass(Class<? extends Reducer> class)` – використовується для встановлення Reducer для завдання [23].

2.7 Обчислення MapReduce

MapReduce розроблена таким чином, щоб продовжувати роботу навіть у разі системних збоїв. Коли завдання виконується, MapReduce відстежує прогрес кожного сервера, що бере участь у завданні. Якщо один з них затримується з відповіддю або виходить з ладу до завершення роботи, MapReduce автоматично запускає інший екземпляр цього завдання на іншому сервері, що має копію даних. Складність механізму обробки помилок повністю прихована від програміста. MapReduce ґрунтується на операціях map і reduce у функціональних мовах, таких як Lisp. Ця модель абстрагує обчислювальні проблеми через дві функції: map і reduce. Усі проблеми, сформульовані таким чином, можуть автоматично паралелізуватися. В основному, модель MapReduce дозволяє користувачам писати компоненти map/reduce у функціональному стилі. Ці компоненти потім компонуються як граф потоку даних, що явно визначає їх паралелізм. Нарешті, система виконання MapReduce планує ці компоненти для розподілених ресурсів, вирішуючи безліч складних проблем: паралелізація, мережеве спілкування та відмовостійкість [24].

Функція map приймає пару ключ/значення як вхідні дані та продукує список пар ключ/значення як вихідні дані. Тип вихідних ключа та значення може відрізнятися від вхідних:

map :: (key1; value1) -> list(key2; value2)… (1)

Функція reduce приймає ключ і пов'язаний список значень як вхідні дані та генерує список нових значень як вихідні дані:

reduce :: (key2; list(value2)) -> list(value3)… (2)

Додаток MapReduce виконується паралельно у два етапи. У першій фазі всі операції map можуть виконуватися незалежно одна від одної. У другій фазі кожна операція reduce може залежати від виходів, згенерованих будь-якою

кількістю операцій map. Усі операції reduce також можуть виконуватися незалежно, подібно до операцій map [25].

Використовуватися MapReduce може до прикладу у Google:

- створення індексу для Google Search;
- кластеризація статей для Google News;
- статистичний машинний переклад.

У Facebook: оптимізація реклами, виявлення спаму.

Висновки до розділу 2

Модель програмування MapReduce використовується для ефективної паралельної обробки великих обсягів даних у розподілених системах. Це підтверджує необхідність застосування MapReduce для оптимізації обробки великих наборів даних у сучасних технологіях.

MapReduce включає дві ключові фази: Map і Reduce. Фаза Map займається перетворенням вхідних даних у пари ключ-значення, тоді як фаза Reduce виконує агрегування або групування даних за цими парами ключ-значення. Це підкреслює важливість кожної фази у процесі обробки даних у MapReduce.

Класи Mapper і Reducer є критично важливими у MapReduce. Mapper перетворює вхідні пари ключ-значення у проміжні пари, які потім обробляються класом Reducer для отримання кінцевого результату. Це акцентує увагу на необхідності правильної роботи цих класів для успішної обробки даних.

Таким чином, розглянутий розділ дипломної роботи надає важливі знання про модель програмування MapReduce, її фази та класи, підкреслюючи їх значущість для ефективної обробки великих обсягів даних у розподілених середовищах.

РОЗДІЛ 3 ОГЛЯД ЗАСОБІВ РОЗРОБКИ ТА РЕЗУЛЬТАТИ РОБОТИ

Для прикладу використання MapReduce на практиці, ми розробили програму, яка використовує фреймворк Apache Hadoop для обробки великих обсягів даних та підрахунку частоти зустрічання слів у текстових файлах. Програма реалізована мовою програмування Java.

3.1 Apache Hadoop

Hadoop MapReduce - це програмна платформа для створення додатків, яка дозволяє обробляти великі обсяги даних (декілька терабайтів) паралельно на великих кластерах (тисячі вузлів) за допомогою надійного, стійкого до відмов способу.

Зазвичай завдання MapReduce розбиває вхідний набір даних на незалежні частини, які обробляються вузлами Map в повній паралельності. Потім результати роботи вузлів Map сортуються і передаються вузлам Reduce. Типово як вхід, так і вихід завдання зберігаються в файловій системі. Платформа відповідає за розподіл завдань, їхнє моніторинг та повторне виконання в разі відмови [26].

3.2 Вибір мови програмування, бібліотек та підготовка

Hadoop – це фреймворк, який здебільшого написаний на мові програмування Java, з деяким власним кодом на C та утилітами командного рядка, написаними як сценарії оболонки [27].

MapReduce теж базується на мові програмування Java, тому для найбільш ефективного способу написання програми для дипломної роботи було вирішено обрати використання мови програмування Java.

Для початку роботи з даною програмою було інстальовано мову програмування Java та фреймворк Hadoop [28].

Використані бібліотеки моделі MapReduce:

- org.apache.hadoop.conf.Configuration: ця бібліотека використовується для налаштування конфігурації Hadoop, таких як шляхи до файлів, параметри кластера тощо;
- org.apache.hadoop.fs.Path: дозволяє працювати з шляхами файлів у Hadoop Distributed File System (HDFS) або локальній файловій системі;
- org.apache.hadoop.io.IntWritable: дана бібліотека являється клас-обгорткою для цілих чисел у контексті Hadoop MapReduce. використовується для передачі цілих чисел у маппері та редукторі;
- org.apache.hadoop.io.LongWritable: бібліотека подібна до IntWritable, але для довгих цілих чисел;
- org.apache.hadoop.io.Text: дана бібліотека представляє рядок даних у контексті Hadoop MapReduce. використовується для передачі текстових даних у маппері та редукторі;
- org.apache.hadoop.mapreduce.Job: дозволяє створювати та налаштовувати завдання MapReduce, такі як вказування класів маппера та редуктора, встановлення ключів та значень виведення тощо;
- org.apache.hadoop.mapreduce.Mapper: використовується для написання мапперів у Hadoop MapReduce. Використовується для обробки вхідних даних та виведення проміжних ключів та значень;
- org.apache.hadoop.mapreduce.Reducer: використовується для написання редукторів у Hadoop MapReduce. Залучена для обробки

проміжних даних, згрупованих за ключем, та виведення результатів;

- org.apache.hadoop.mapreduce.lib.input.FileInputFormat: ця бібліотека надає інтерфейс для встановлення вхідних шляхів для MapReduce задачі;
- org.apache.hadoop.mapreduce.lib.output.FileOutputFormat: ця бібліотека надає інтерфейс для встановлення вихідних шляхів для MapReduce задачі;
- org.apache.hadoop.util.GenericOptionsParser: ця бібліотека використовується для обробки загальних опцій командного рядка Hadoop, таких як вхідні та вихідні шляхи [29].

3.3 Вибір змінних та методів

Conf – це змінна, яка створює об'єкт конфігурації для використання в програмі.

GenericOptionsParser – допоміжний клас Hadoop для обробки загальних опцій командного рядка [30].

OtherArgs – масив рядків, який містить інші аргументи командного рядка, крім опцій, які вже оброблені GenericOptionsParser. Використовується для отримання додаткових аргументів командного рядка, які не відносяться до опцій конфігурації Hadoop.

Job – це об'єкт, який представляє конкретне завдання MapReduce у Hadoop.

Job – змінна, яка створює об'єкт роботи для виконання конкретного завдання MapReduce.

Job.setJarByClass – метод, який вказує Hadoop, який клас містить код для запуску на кластері [31].

Word_counter.class – це клас, який містить метод main і відповідає за запуск програми MapReduce.

Job.setMapperClass – метод, який вказує, який клас використовувати як маппер у завданні MapReduce.

Token_Mapper.class – це внутрішній клас програми, який містить логіку мапперу.

Job.setCombinerClass – метод, який вказує, який клас використовувати як комбінатор (після мапперу, перед редуктором) у завданні MapReduce.

IntSumReducer.class – це внутрішній клас програми, який містить логіку комбінатора.

Job.setReducerClass – метод, який вказує, який клас використовувати як редуктор у завданні MapReduce.

IntSumReducer.class – це внутрішній клас програми, який містить логіку редуктора.

Job.setOutputKeyClass – метод, який вказує тип ключа вихідних даних [32].

Text.class – це клас Hadoop, що представляє рядковий ключ.

Text – це клас Hadoop, що представляє рядок даних у контексті MapReduce.

WordToken – це змінна для представлення окремого слова у тексті.

IntWritable – клас Hadoop, що представляє ціле число для використання у MapReduce.

One – це статична змінна, яка містить ціле число 1, яке використовується для підрахунку кількості входжень кожного слова.

Map – метод маппера, який приймає ключ типу LongWritable, значення типу Text і об'єкт Context.

Key – це ключ рядка у контексті Hadoop MapReduce.

Value – це значення рядка у контексті Hadoop MapReduce.

Result – це змінна, яка містить підсумкове значення після обробки в редукторі.

Reduce – метод редуктора, який приймає ключ типу Text, ітератор значень типу IntWritable та об'єкт Context.

Key – ключ для групування значень у редукторі.

Values – ітератор значень для конкретного ключа у редукторі.

Context – об'єкт, який дозволяє редуктору виводити пари ключ-значення та взаємодіяти з фреймворком Hadoop.

Job.setOutputValueClass – метод, який вказує тип значення вихідних даних.

IntWritable.class – це клас Hadoop, що представляє ціле число для запису у вихідні дані.

FileInputFormat.addInputPath – метод для додавання вхідного шляху до завдання MapReduce.

New Path(otherArgs[i]) – створення об'єкта шляху для кожного вхідного шляху, який передається як аргумент.

FileOutputFormat.setOutputPath – метод для встановлення вихідного шляху результатів MapReduce завдання [33].

New Path(otherArgs[otherArgs.length – 1]) - створення об'єкта шляху для вихідного шляху, який передається як останній аргумент командного рядка.

Job.waitForCompletion(true) – метод, який запускає MapReduce задачу та очікує її завершення.

System.exit – метод, який виводить результат роботи MapReduce задачі у консоль та завершує програму з кодом виходу 0 у випадку успішного завершення, або 1 у випадку помилки.

Token_Mapper – внутрішній статичний клас, який розширює клас Mapper і визначає маппер для задачі MapReduce.

Job.getInstance() – для створення об'єкта конфігурації Hadoop.

<LongWritable, Text, Text, IntWritable> – це типи ключа та значення для вхідних та вихідних даних маппера: вхідний ключ (LongWritable), вхідний текст (Text), вихідний текст (Text), вихідне ціле число (IntWritable).

IntSumReducer – внутрішній статичний клас, який розширює клас Reducer і визначає редуктор для задачі MapReduce [35].

<Text, IntWritable, Text, IntWritable> – це типи ключа та значення для вхідних та вихідних даних редуктора: вхідний текст (Text), вхідне ціле число (IntWritable), вихідний текст (Text), вихідне ціле число (IntWritable) [34].

Main – основний метод програми, який викликається при запуску програми.

String[] та args – це аргументи командного рядка, які можуть містити шляхи вхідних та вихідних файлів для MapReduce задачі.

Tokens – масив рядків, який містить токени (слова) з вхідного тексту.

Value.toString() – перетворення значення з вхідного тексту у рядок.

.split(" ") – розділення рядка на токени (слова) за допомогою пробілів.

For (String word : tokens) – цикл, який проходиться по кожному слову в масиві токенів.

Sum – змінна, яка використовується у редукторі для підрахунку суми значень, що відповідають кожному унікальному слову.

For (IntWritable val : values) sum += val.get() – цей цикл проходиться по кожному значенню для конкретного ключа у редукторі і додає його до суми sum.

Result.set(sum) – встановлює підсумкове значення sum у змінну result, яка потім виводиться у контекст разом з ключем у вихідній парі.

Для проведення експериментів над текстами різного розміру було додано лічильник часу.

Long startTime – ця змінна використовується для запам'ятовування часу початку виконання програми.

Long endTime – ця змінна використовується для запам'ятовування часу завершення виконання програми.

long executionTime – ця змінна використовується для обчислення часу виконання програми (різниці між endTime і startTime).

boolean jobCompleted – ця змінна використовується для позначення того, чи завершилося виконання завдання MapReduce успішно.

Для вимірювання часу виконання програми використовується системний час в мілісекундах.

long executionTime = endTime - startTime; - обчислює загальний час виконання програми як різницю між часами початку і завершення.

Результат виконання програми (job.waitForCompletion(true)) записується у змінну jobCompleted.

При успішному виконанні програми (значення true у jobCompleted), програма виводить повідомлення "Job completed successfully in [час виконання] milliseconds" разом з часом виконання.

При невдачі або перериванні програми (значення false у jobCompleted), програма виводить повідомлення "Job failed or was interrupted".

1) Щоб запустити програму, потрібно підготувати текстовий файл у якому потрібно підрахувати кількість входжень кожного слова.
2) Далі треба скомпілювати програму за допомогою команди `javac Word_counter.java`.
3) Після компіляції треба запустити програму за допомогою команди `java Word_counter <input-path> <output-path>`, де `<input-path>` - шлях до вхідного текстового файлу, а `<output-path>` - шлях до вихідної директорії, де буде збережений результат підрахунку. Після завершення обчислень результат буде збережений у вказаній вихідній директорії.
4) Щоб переглянути результат треба зайти у вихідну директорію `<output-path>` і знайти результат у вигляді файлу (під назвою `part-r-00000`). Відкривати цей файл слід за допомогою текстового редактора або командою `cat` у терміналі для перегляду результату.
5) У вихідному файлі кожний рядок буде містити слово та кількість його входжень у вихідний текст.

Отже, логіка програми полягає в наступному порядку дій:

- фіксується час початку виконання програми;
- виконується програма MapReduce для підрахунку кількості слів у тексті;

- фіксується час завершення виконання програми та обчислюється загальний час виконання;
- виводиться результат виконання разом з часом виконання у термінал;
- програма завершується з відповідним кодом виходу в залежності від успішності виконання.

3.4 Результати експериментів

Для експерименту було взято 3 вірші Тараса Шевченка різної довжини.

1) «Іван Підкова» (рис. 3.1) у вхідному файлі ivan.txt.
2) «Тополя» (рис. 3.2) у вхідному файлі topolya.txt.
3) «Якби-то ти, Богдане п’яний» (рис. 3.3) у вхідному файлі bogdan.txt.

Рисунок 3.1 – Вірш «Іван Підкова».

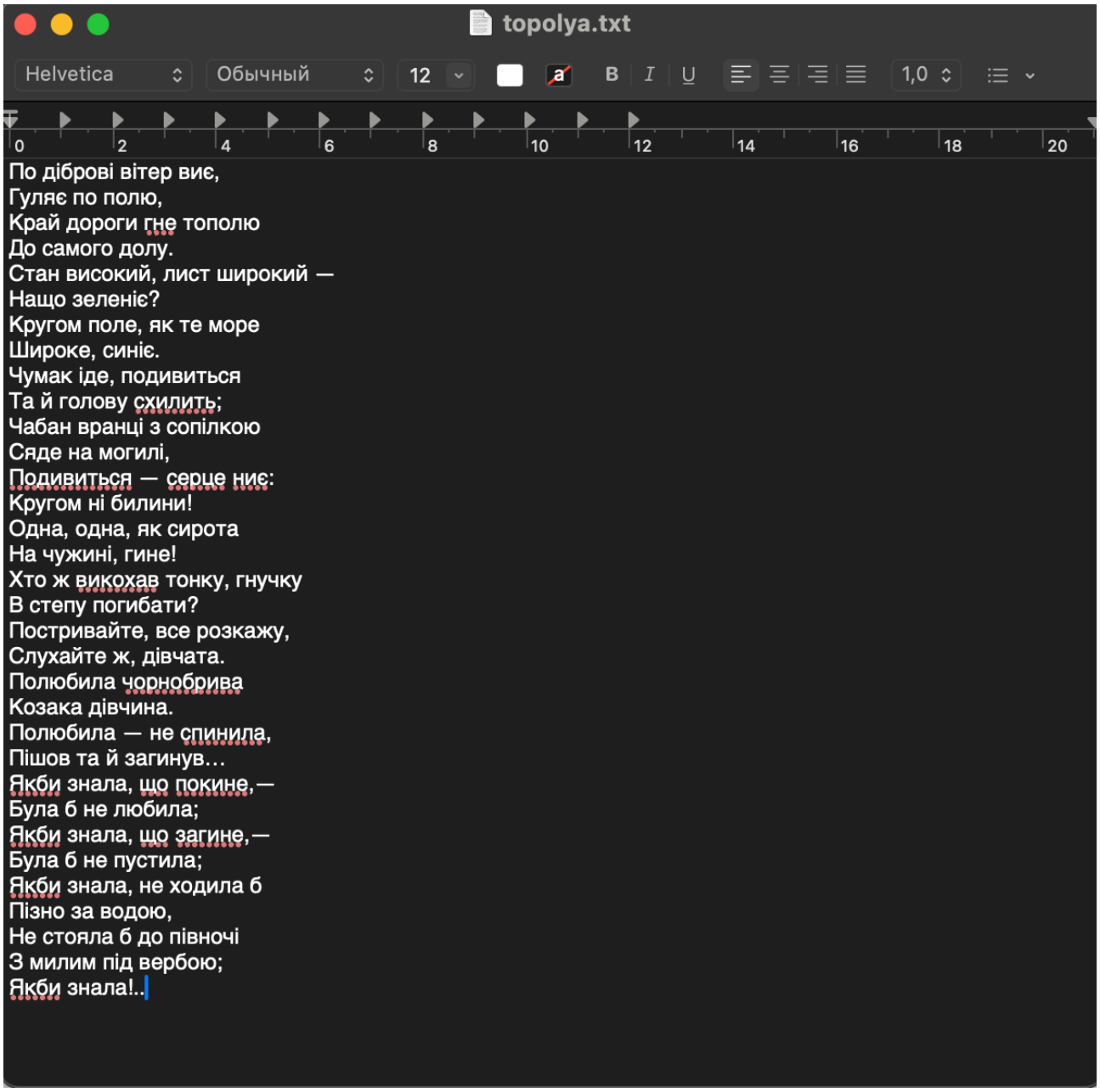


Рисунок 3.2 – Вірш «Тополя».

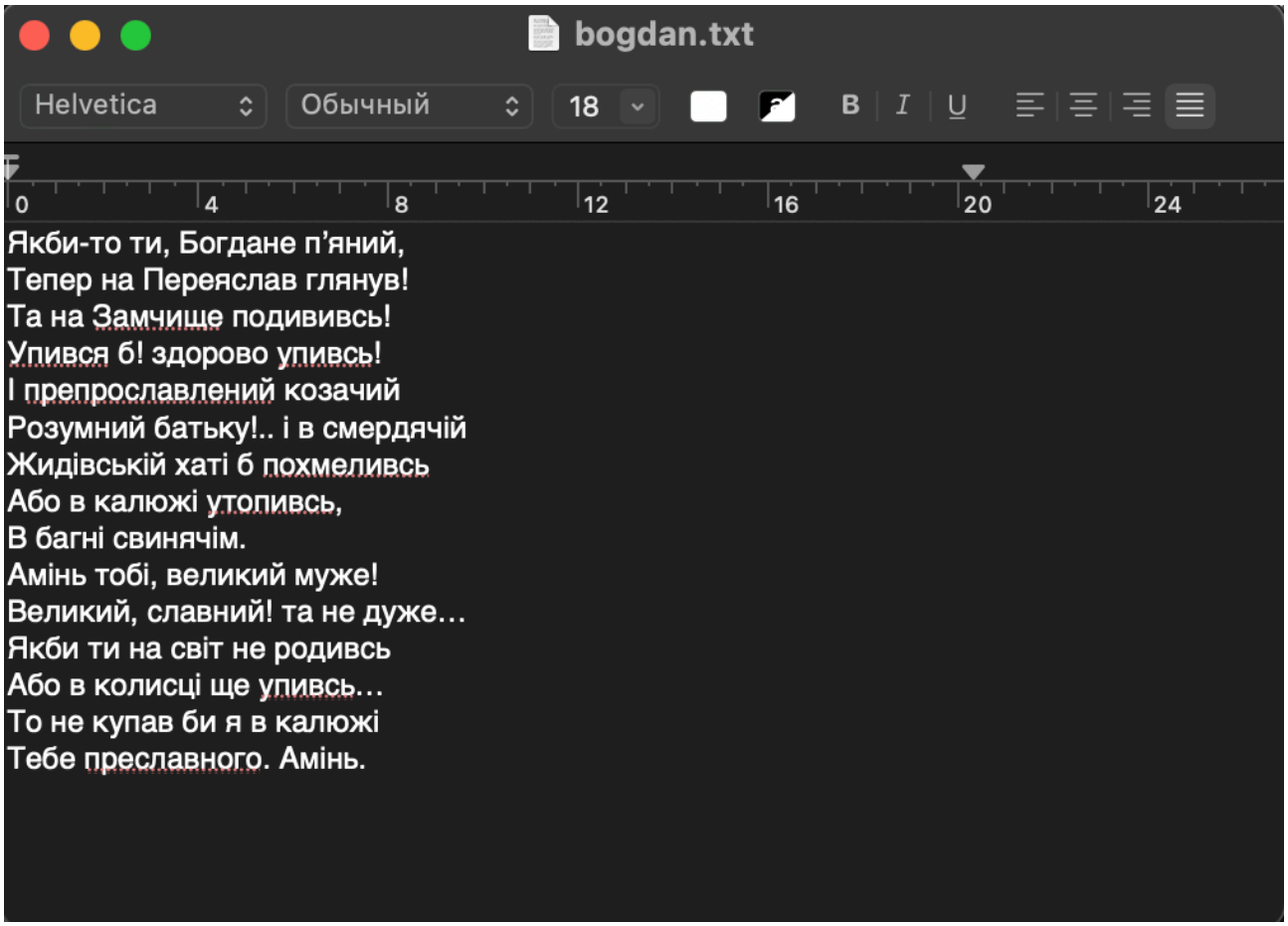


Рисунок 3.3 – Вірш «Якби-то ти, Богдане п’яний».

Результати проведення експерименту.

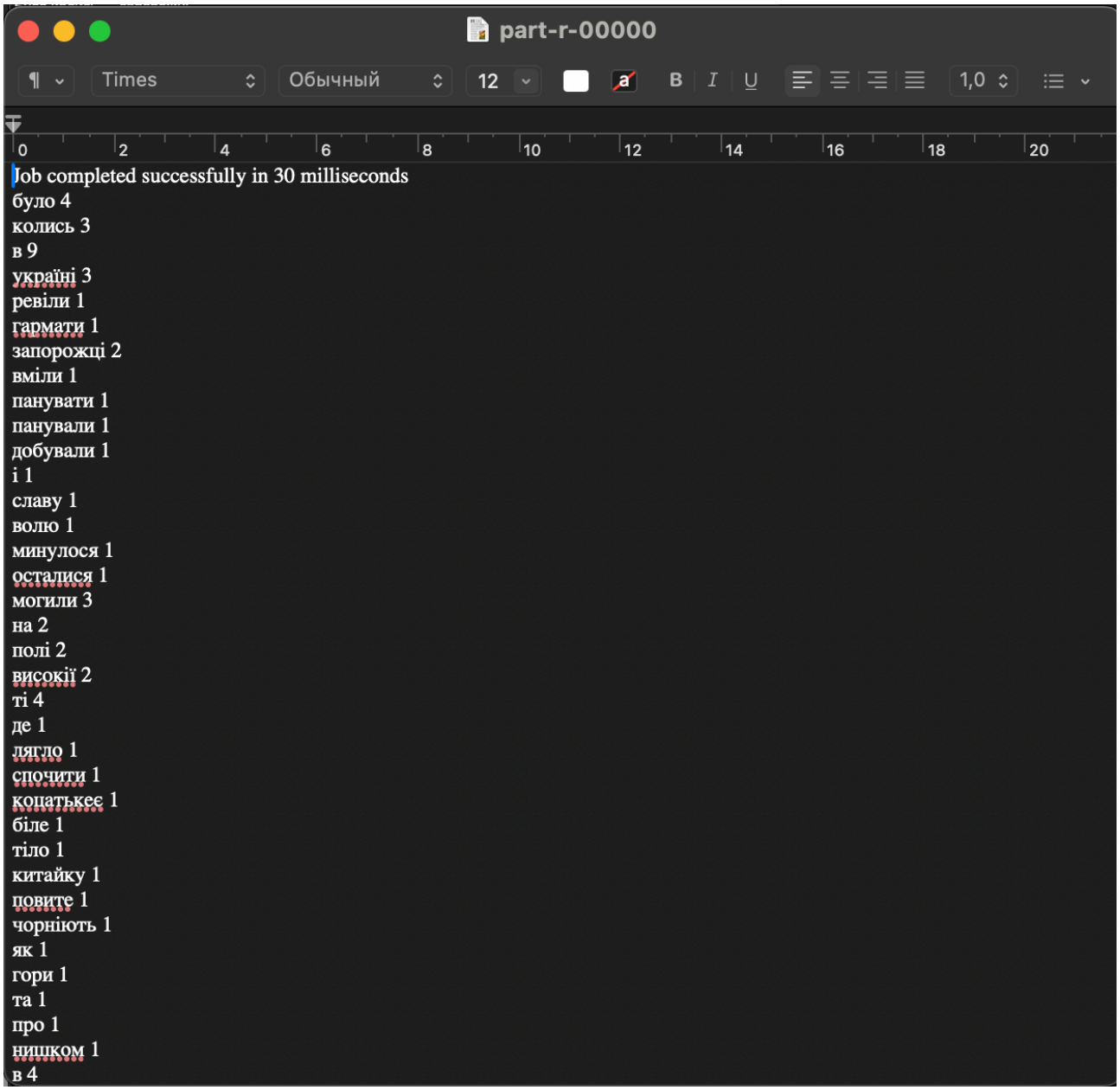
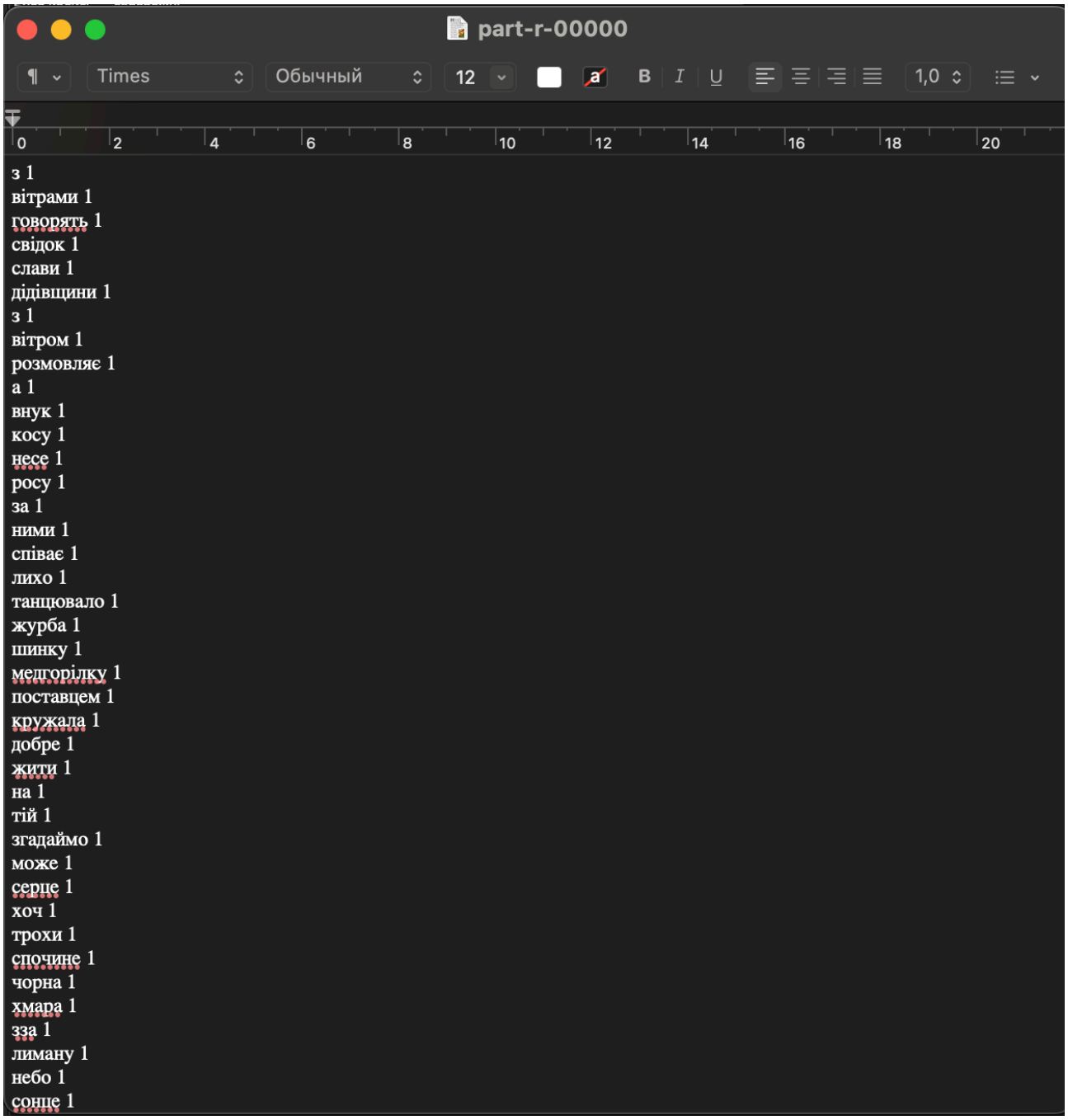


Рисунок 3.4 – Аналіз віршу «Іван Підкова» частина 1.


part-r-00000
Times
Обычный
12
B
I
U
1,0
0
2
4
6
8
10
12
14
16
18
20
з 1
вітрами 1
говорять 1
свідок 1
слави 1
дідівщини 1
з 1
вітром 1
розмовляє 1
а 1
внук 1
косу 1
несе 1
росу 1
за 1
ними 1
співає 1
лихо 1
танцювало 1
журба 1
шинку 1
медгорілку 1
поставцем 1
кружала 1
добре 1
жити 1
на 1
тій 1
згадаймо 1
може 1
серце 1
хоч 1
трохи 1
спочине 1
чорна 1
хмара 1
зза 1
лиману 1
небо 1
сонце 1

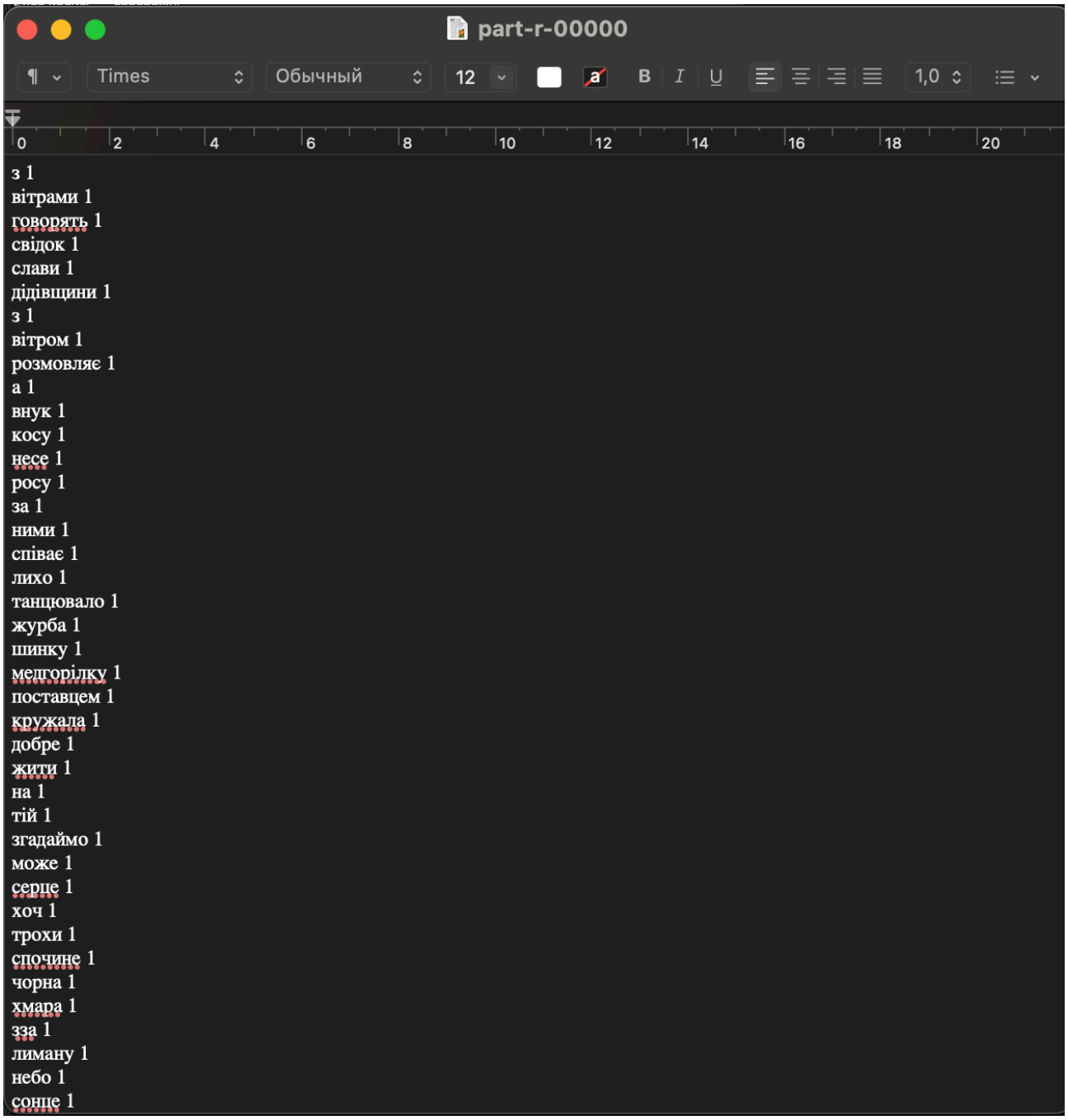


Рисунок 3.5 – Аналіз віршу «Іван Підкова» частина 2.

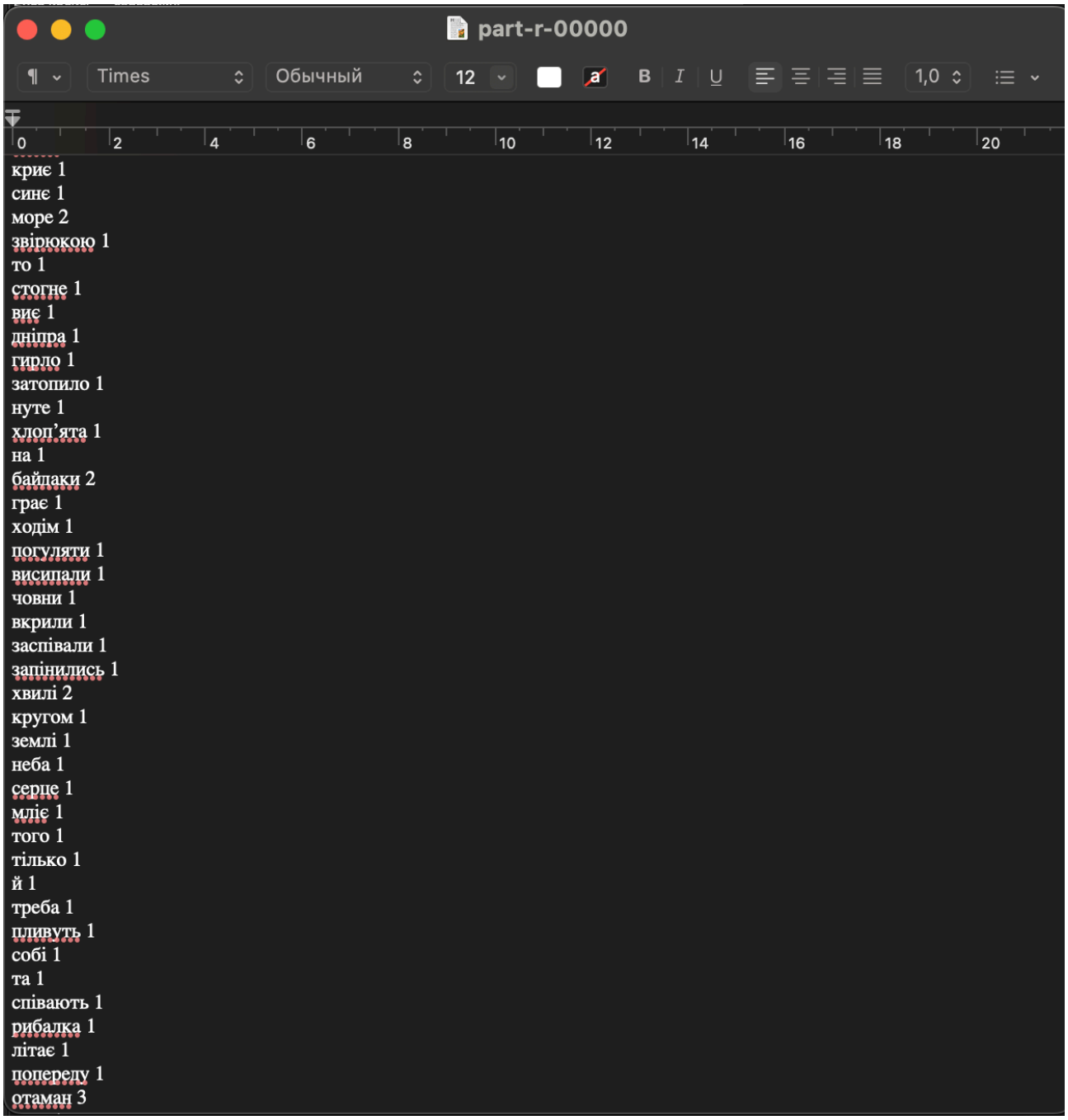


Рисунок 3.6 – Аналіз віршу «Іван Підкова» частина 3.

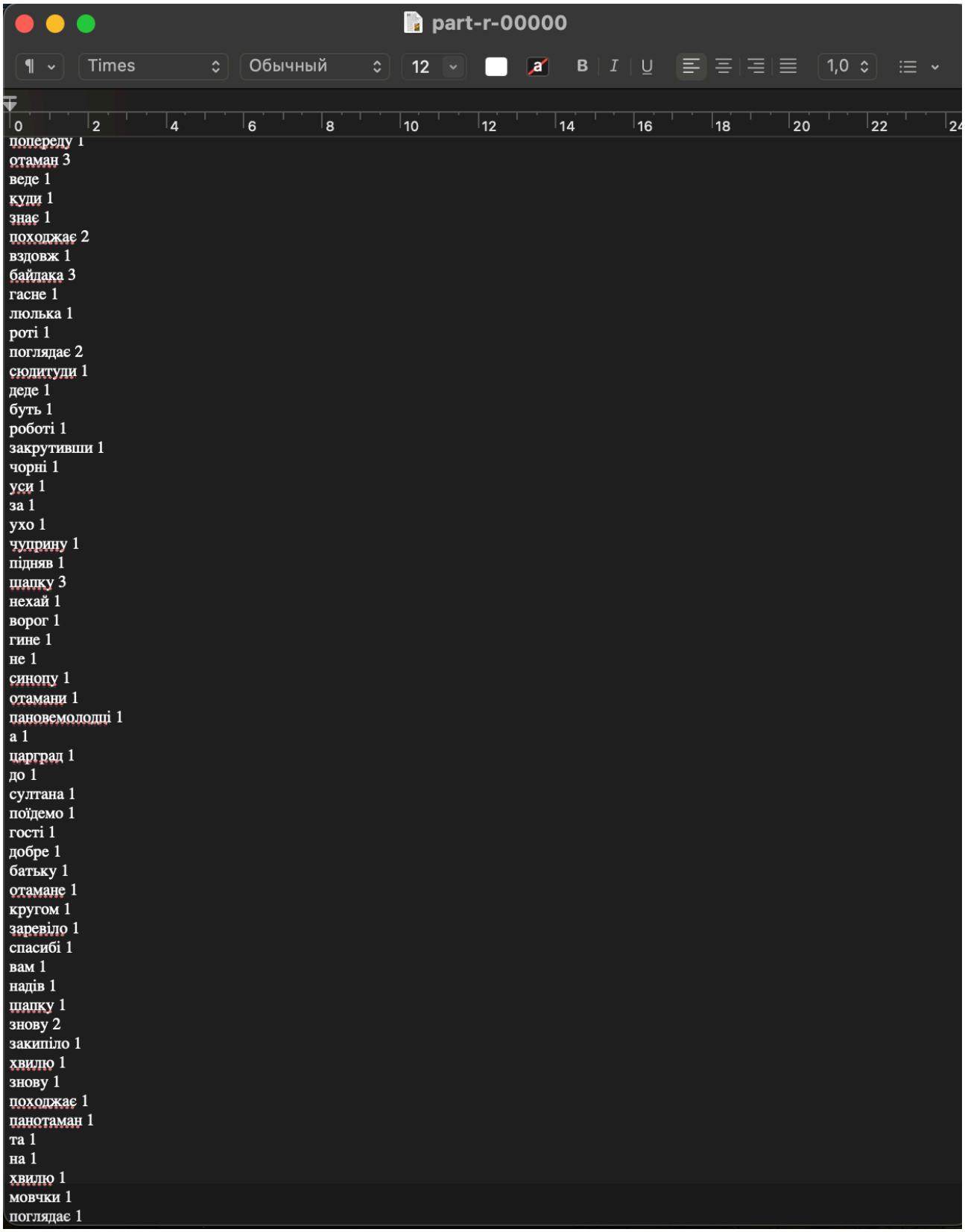


Рисунок 3.7 – Аналіз віршу «Іван Підкова» частина 4.

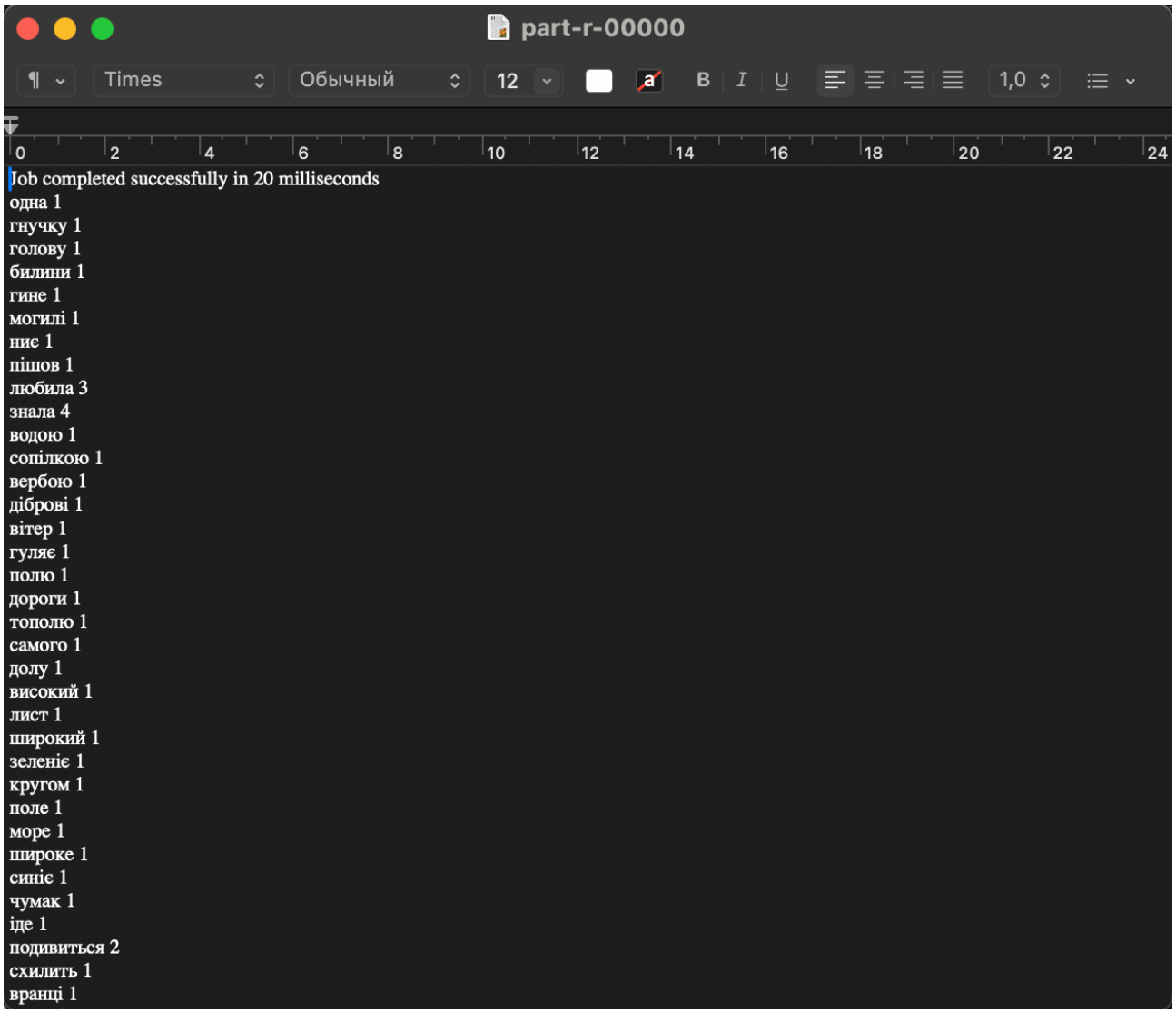


Рисунок 3.8 – Аналіз віршу «Тополя» частина 1.

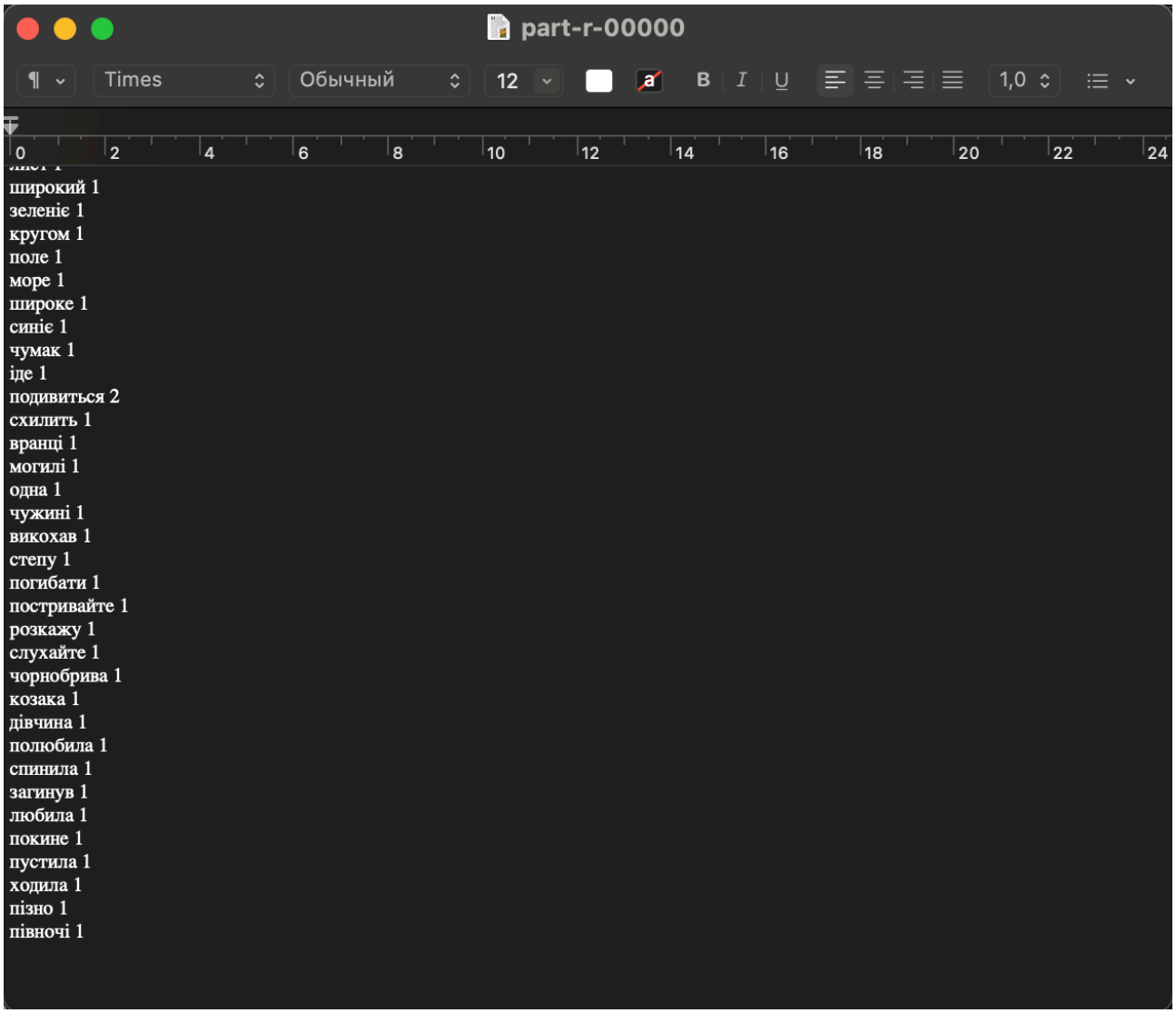


Рисунок 3.9 – Аналіз віршу «Тополя» частина 2.

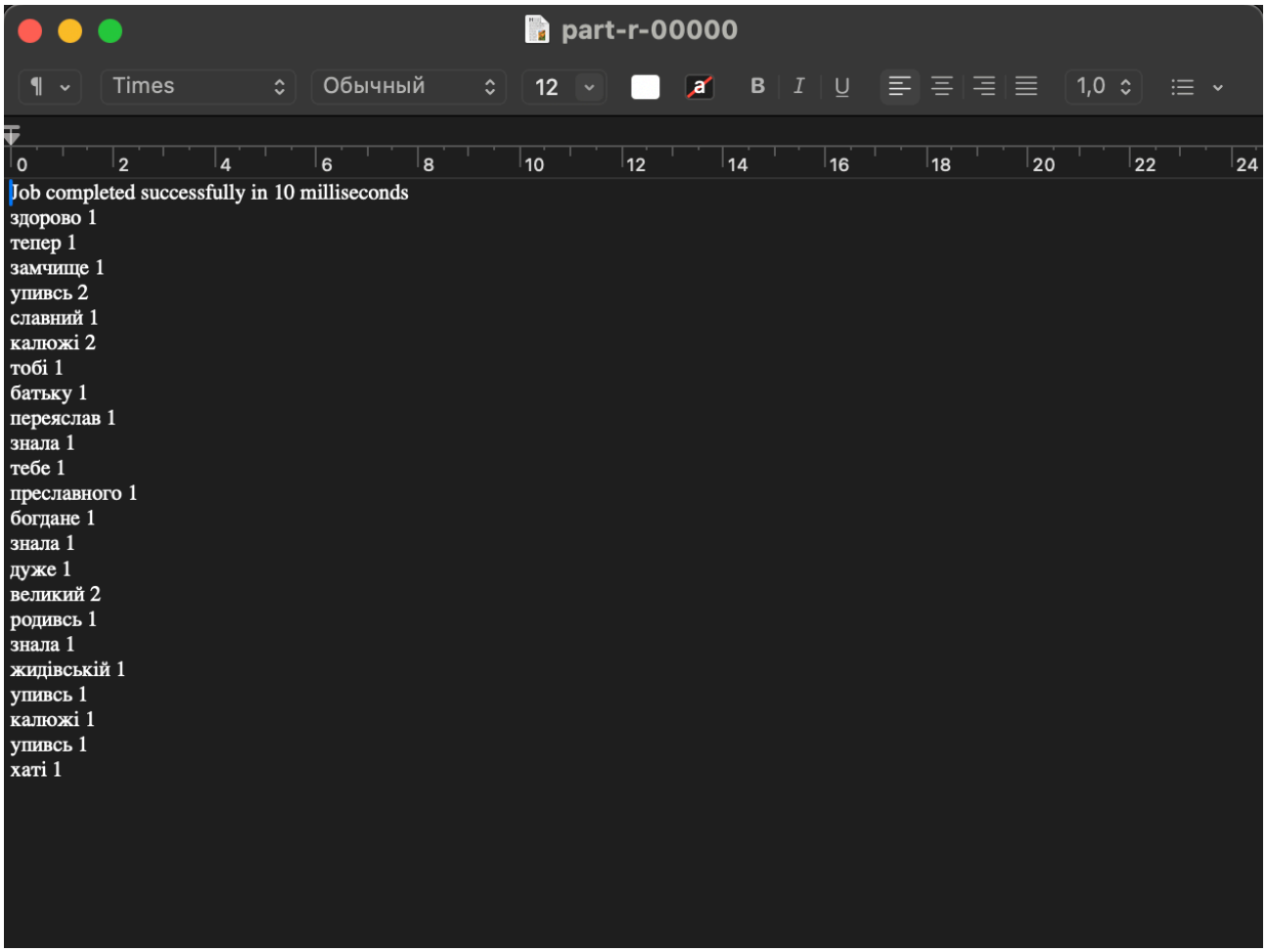


Рисунок 3.10 – Аналіз віршу «Якби-то ти, Богдане п’яний».

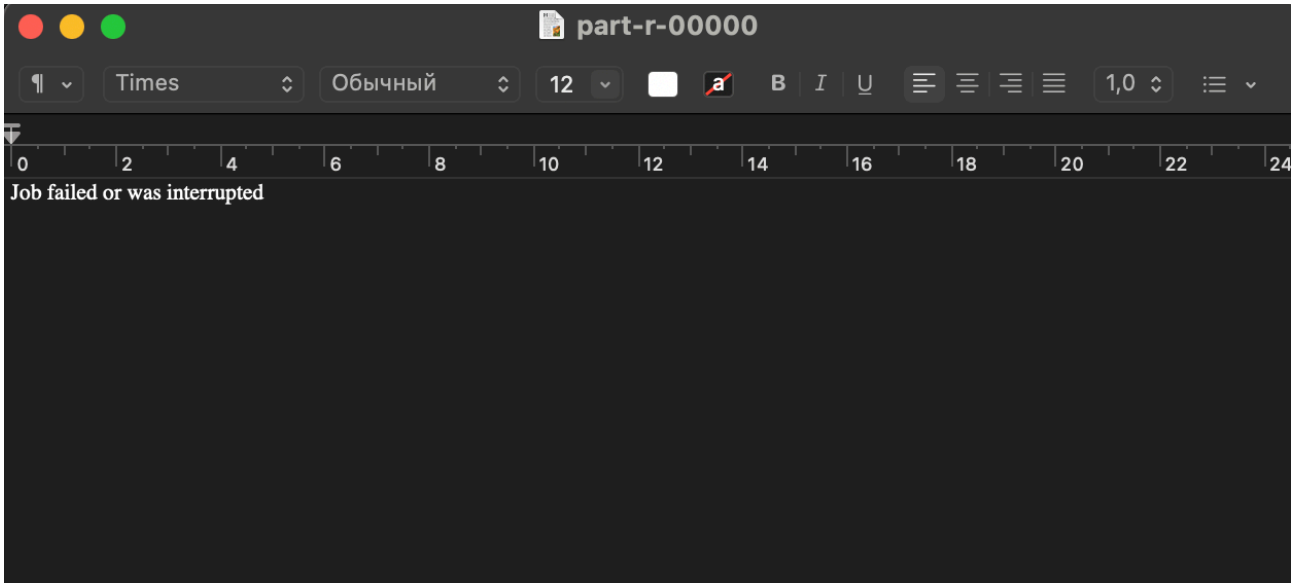


Рисунок 3.11 – Результат невдалого аналізу.

Для того щоб було зручно порівняти отримані результати, зробимо відповідну таблицю (табл. 3.1).

Таблиця 3.1 – Отримані результати.

Категорія	«Іван Підкова»	«Тополя»	«Якби-то ти, Богдане п’яний»
Загальна кількість слів	218	114	65
Загальна кількість символів	1397	674	392
Результат без повторів	342	112	46
Час, мс	30	20	10

Висновки до розділу 3

Метод MapReduce використовується для розподілу обчислювальної роботи на кілька незалежних частин, щоб зменшити час обробки великих обсягів даних. Основний висновок за наданими результатами полягає в тому, що обробка текстів трьох віршів за допомогою методу MapReduce значно зменшила час обчислення. Наприклад, час обробки "Якби-то ти, Богдане п’яний" у 10 мілісекунд значно менше, ніж для інших двох віршів (30 мс та 20 мс). Також можна відзначити, що взагалі обробка великого обсягу текстів (загальна кількість слів та символів) відбулася досить швидко, що є ще одним позитивним висновком відносно ефективності методу MapReduce.

Також зазначимо, що для виконання даного дослідження з використанням методу MapReduce не знадобилося ніякого додаткового обладнання чи вкладень. Це свідчить про доступність та ефективність використання цього методу як для

особистих потреб, так і для бізнесу. Реалізація програми не потребує специфічних додатків та може бути реалізована на всіх операційних системах за допомогою команд у терміналі для завантаження необхідних бібліотек, мови програмування та самого Hadoop.

Враховуючи велику швидкість обробки даних та відсутність потреби у складних обчислювальних ресурсах, метод MapReduce може бути цінним інструментом для різноманітних аналітичних та оброблювальних завдань з використанням великих обсягів даних.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на дослідженні демографічного стану.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація. Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

- F_1 – вибір мови програмування;
- F_2 – вибір способу реалізацій алгоритмів;
- F_3 – вибір середовища програмування.

Кожна з цих функцій має декілька варіантів реалізації:

- функція F_1: Java; C++;

- функція F_2: використання готових бібліотек обробки та аналізу Big Data у терміналі; створення власних алгоритмів обробки та аналізу Big Data;
- функція F_3:Jupyter Notebook; Visual Studio Code 2.

Варіанти реалізації основних функцій наведені у морфологічній карті системи на рис. 4.1.

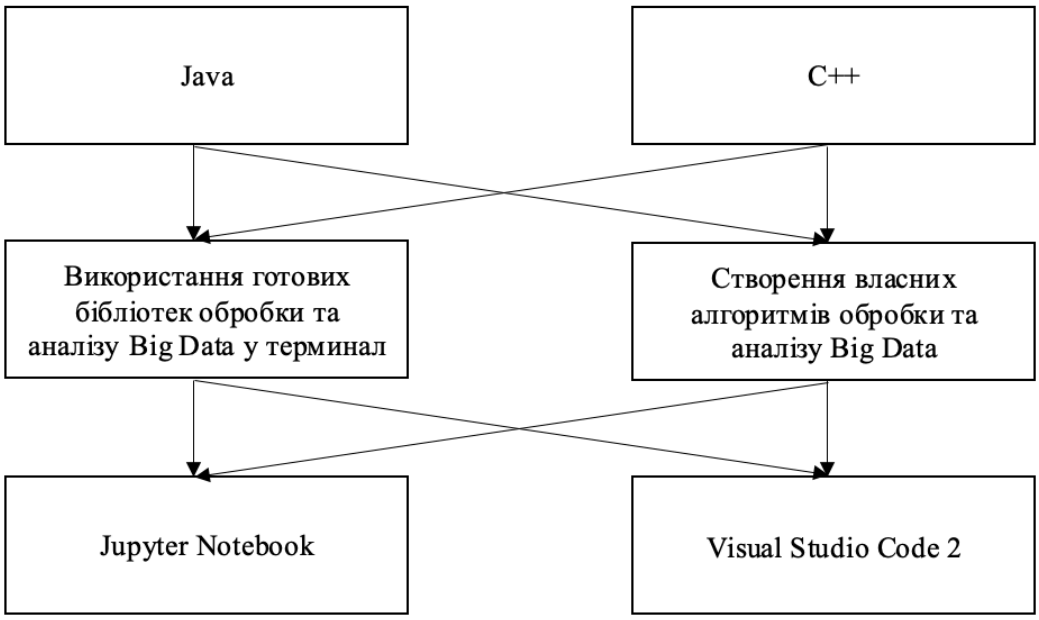


Рисунок 4.1 – Морфологічна карта системи

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	*Варіанти реалізації*	*Переваги*	*Недоліки*
F_1	А	Швидка розробка, чіткий і зрозумілий синтаксис, велика стандартна бібліотека, крос-платформеність, широке застосування.	Повільна швидкість виконання, велике використання пам’яті, має обмеження у реалізації мультипоточних програм.
	Б	Висока продуктивність, контроль над пам’яттю, гнучкість, мультиплатформенність	Проблеми з безпекою пам’яті, складність використання та написання програм
F_2	А	Доступність та легкість у використанні, перевірені та протестовані алгоритми	Менша гнучкість, виконують раніше поставлену задачу
	Б	Реалізація кастомного функціоналу, який потрібен програмі	Великий час реалізації, потреба у додатковому етапі тестування
F_3	А	Інтерактивність середовища, візуалізація помилок, підтримка багатьох мов програмування, легка інтеграція з хмарними сервісами	Обмежені можливості налагодження коду, великі блокноти стають важкими в управлінні
	Б	Автодоповнення коду, підсвічування синтаксису, можливість інтеграції з Git	Ресурсомісткість, вартість, складність освоєння

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1*:* перевагу даємо загальнодоступності та швидкості реалізації; для спрощення роботи по написанню коду варіант Б має бути відкинутий. Функція F_2*:* перевагу даємо доступності в реалізації та меншим часовим витратам на розробку програмного продукту; для спрощення роботи по написанню коду варіант Б має бути відкинутий. Функція F_3*:* програма допускає обрання обох варіантів; можливо використати варіанти А чи Б.

Таким чином, будемо розглядати такий варіанти реалізації програмного продукту з формул (4.1) та (4.2).

$$F_1 a - F_2 a - F_3 a, \tag{4.1}$$

$$F_1 a - F_2 a - F_3 \text{б}. \tag{4.2}$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X_1 – швидкодія мови програмування;
- X_2 – об’єм пам’яті для обчислень та збереження даних;
- X_3 – час навчання даних;
- X_4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметру	*Умовне позначення*	*Одиниці виміру*	*Значення параметра*		
			гірші	*середні*	*кращі*
Швидкодія мови програмування	X_1	оп/мс	50	80	120
Об’єм пам’яті	X_2	Мб	70	40	30
Час попередньої обробки даних	X_3	мс	100	30	5
Потенційний об’єм програмного коду	X_4	кількість рядків коду	400	200	100

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

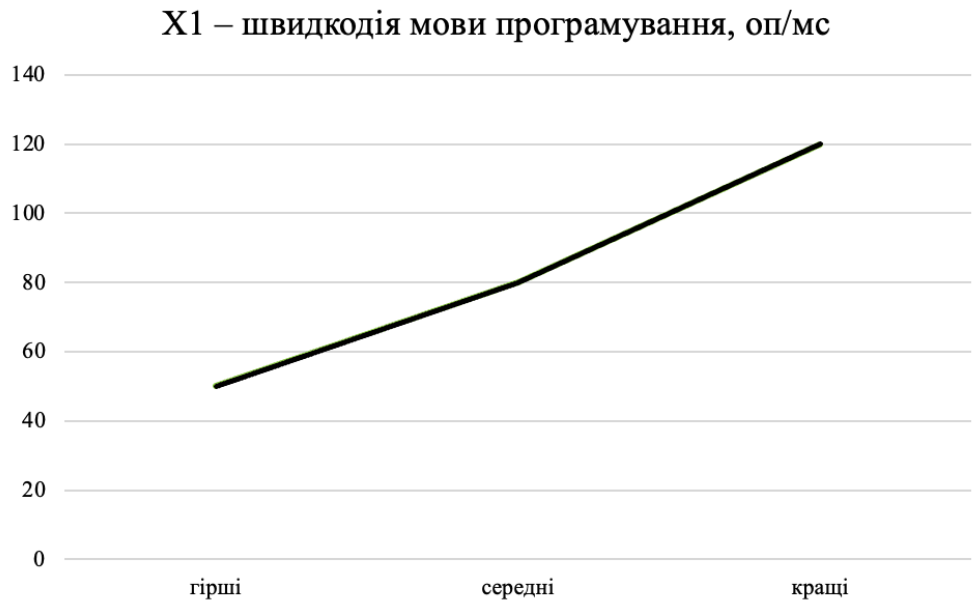


Рисунок 4.2 – Графік значень параметру X1 (швидкодія мови), оп/мс

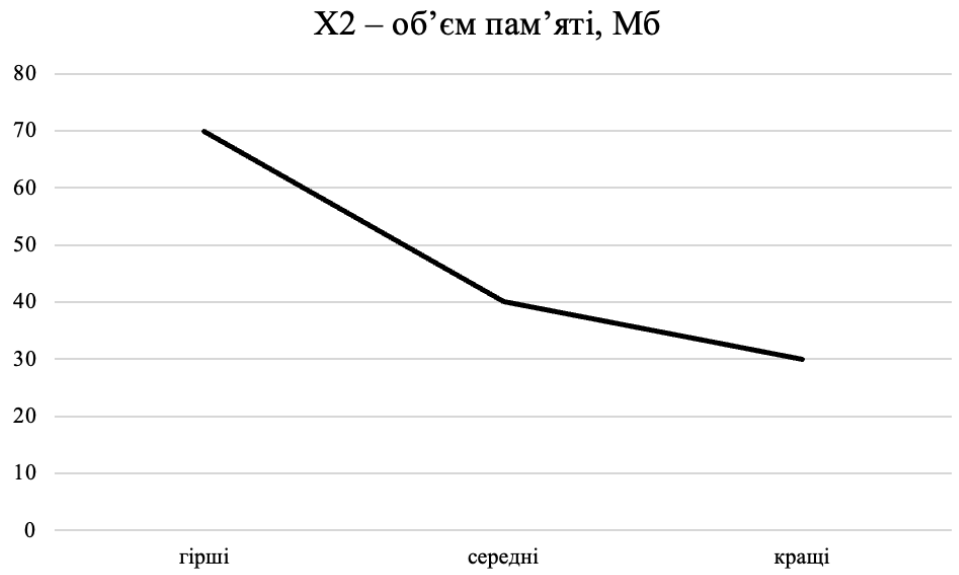


Рисунок 4.3 – Графік значень параметру X2 (об’єм пам’яті), Мб

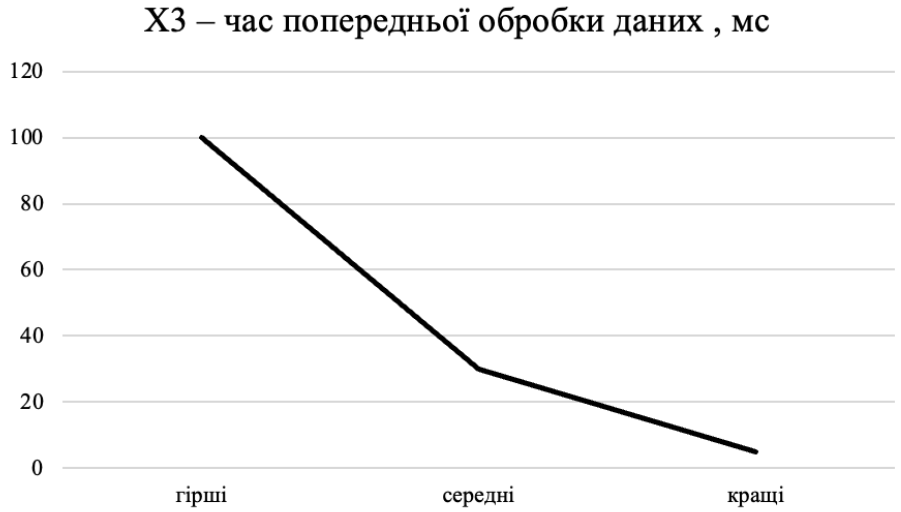


Рисунок 4.4 – Графік значень параметру X3 (час попередньої обробки даних), мс

Рисунок 4.5 – Графік значень параметру X4 (кількість рядків коду)

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Основні параметри програмного продукту

Назва параметру та одиниці виміру	Умовне позначення	Ранг параметра за оцінкою експерта							Сума рангів R_i	Δ_i	Δ_i^2
		1	2	3	4	5	6	7			
Швидкодія мови програмування, оп/мс	X_1	1	1	2	2	1	1	2	10	-7,5	56,25
Об'єм пам'яті, Мб	X_2	3	4	3	3	4	3	4	24	6,5	42,25
Час попередньої обробки даних, мс	X_3	2	2	1	1	2	3	1	12	-5,5	30,25
Потенційний об'єм програмного коду, кількість рядків коду	X_4	4	3	4	4	3	3	3	24	6,5	42,25
Разом		10	10	10	10	10	10	10	70	0	171

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

- сума рангів кожного з параметрів і загальна сума рангів визначається за формулою (4.3):

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.3)$$

де N – число експертів,

n – кількість параметрів

R_{ij} – ранг і-го параметру проставлений j-им експертом

- середня сума рангів визначається за формулою (4.4):

$$T = \frac{1}{n} R_{ij} = 17{,}5. \quad (4.4)$$

- відхилення суми рангів кожного параметра від середньої суми рангів за формулою (4.5):

$$\Delta_i = R_i - T. \quad (4.5)$$

Сума відхилень по всім параметрам повинна дорівнювати 0.

- загальна сума квадратів відхилення за формулою (4.6):

$$S = \sum_{i=1}^{N} \Delta_i^2 = 171. \quad (4.6)$$

Порахуємо коефіцієнт узгодженості за формулою (4.7):

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 171}{7^2(4^3-4)} = 0{,}69796 > W_k = 0{,}67. \quad (4.7)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів

Умовне позначення	*Експерти*							*Кінцева оцінка*	*Числове значення*
	1	*2*	*3*	*4*	*5*	*6*	*7*		
X_1 і X_2	<	>	>	<	>	<	>	>	1,5
X_1 і X_3	>	<	<	<	>	<	>	<	0,5
X_1 і X_4	>	<	>	<	>	>	<	>	1,5
X_2 і X_3	<	>	>	>	>	<	>	>	1,5
X_2 і X_4	>	<	<	<	>	<	<	<	0,5
X_3 і X_4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається за формулою (4.8):

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_{j.} \end{cases} \quad (4.8)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$. Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за формулами (4.9) та (4.10):

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \quad (4.9)$$

$$b_i = \sum_{i=1}^{N} a_{ij}. \quad (4.10)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за формулами (4.11) та (4.12):

$$K_{\text{вi}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (4.11)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j. \quad (4.12)$$

На таблиці 4.5 зображено розрахунок вагомості параметрів на трьох ітераціях. На цьому етапі різниця значень коефіцієнтів вагомості параметрів не перевищує 2%, тому більшої кількості ітерацій розрахунків немає потреби проводити.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація		Третя ітерація	
	X_1	X_2	X_3	X_4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X_1	1	1,5	0,5	1,5	4,5	0,28	20,25	0,31	91,125	0,34
X_2	1,5	1	1,5	0,5	4,5	0,28	20,25	0,31	91,125	0,34
X_3	0,5	1,5	1	0,5	3,5	0,22	12,25	0,19	42,875	0,16
X_4	1,5	0,5	0,5	1	3,5	0,22	12,25	0,19	42,875	0,16
Всього					16	1	65	1	268	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (об'єм пам'яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується за формулою (4.13).

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}, \quad (4.13)$$

де n – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості і-го параметра;

B_i – оцінка і-го параметра в балах.

Розрахунки коефіцієнтів вагомості параметрів та коефіцієнтів рівня якості для основних функцій наведено в таблиці 4.6.

Таблиця 4.6 – Розрахунок вагомості параметрів

Функція	Варіант реалізації функції	Параметр	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	А	X_1	80	10	0,34	3,4
F_2	А	X_2	30	5	0,34	1,7
F_3	А	X_4	5	9	0,16	1,44
	Б	X_4	100	3	0,16	0,48

За даними таблиці 4.6 можемо використати формулу (4.14) для розрахунку рівня якості кожного з варіантів.

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}]. \quad (4.14)$$

Числові розрахунки двох варіантів наведено в формулах (4.15) та (4.16).

$$K_{K1} = 3{,}1 + 1{,}7 + 1{,}44 = 6{,}24, \quad (4.15)$$

$$K_{K2} = 3{,}1 + 1{,}7 + 0{,}48 = 5{,}28. \quad (4.16)$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня K_{K1} має більше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

- розробка проекту програмного продукту;
- розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється за формулою (4.17).

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.17)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P = 30 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.8. Поправочний

коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ} = 0.9$. Тоді загальна трудомісткість програмування першого завдання розраховується як $Т_1 = 30 \cdot 1{,}8 \cdot 0{,}9 = 48{,}6$ людино-днів.

Для другого завдання (використовується алгоритм третьої групи складності, ступінь новизни Б) – $Т_Р = 37$ людино-днів, $К_П = 0.9$, $К_{СК} = 1$, $К_{СТ} = 0.8$. Таким чином, загальна трудомісткість $Т_2 = 37 \cdot 0{,}9 \cdot 0{,}8 = 26{,}64$ людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість, розрахунки наведено в формулах (4.18) та (4.19).

$$Т_I = (48{,}6 + 26{,}64 + 4{,}8 + 26{,}64) \cdot 8 = 853{,}44 \text{ людино-годин} \quad (4.18)$$

$$Т_{II} = (48{,}6 + 26{,}64 + 7{,}9 + 26{,}64) \cdot 8 = 878{,}24 \text{ людино-годин.} \quad (4.19)$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 25000 грн., один аналітик в області даних з окладом 30000. Визначимо середню зарплату за годину за формулою (4.20).

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.,} \quad (4.20)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

Розрахунки наведено формулою (4.21).

$$С_{ч} = \frac{25000+25000+30000}{3 \cdot 21 \cdot 8} = 158{,}73 \text{ грн.} \quad (4.21)$$

Загальна заробітна плата розраховується за формулою (4.22).

$$С_{ЗП} = С_{ч} \cdot Т_{i} \cdot К_{Д} \text{ грн.}, \quad (4.22)$$

де $С_{ч}$ – величина погодинної оплати праці програміста;

$Т_{i}$ – трудомісткість відповідного завдання;

$К_{Д}$ – норматив, який враховує додаткову заробітну плату.

Зарплату розробників за варіантами наведено в формулах (4.23) та (4.24).

$$С_{ЗП\,I} = 158{,}73 \cdot 853{,}44 \cdot 1{,}2 = 162559{,}8 \text{ грн.} \quad (4.23)$$

$$С_{ЗП\,II} = 158{,}73 \cdot 878{,}24 \cdot 1{,}2 = 167283{,}64 \text{ грн.} \quad (4.24)$$

Відрахування на єдиний соціальний внесок становить 22%, розрахунки наведено в формулах (4.25) та (4.26).

$$С_{ВІД\,I} = С_{ЗП\,I} \cdot 0{,}22 = 162559{,}8 \cdot 0{,}22 = 35763{,}156 \text{ грн.} \quad (4.25)$$

$$С_{ВІД\,II} = С_{ЗП\,II} \cdot 0{,}22 = 167283{,}64 \cdot 0{,}22 = 36802{,}4 \text{ грн.} \quad (4.26)$$

Тепер визначимо витрати на оплату однієї машино-години ($С_{М}$). Так як одна ЕОМ обслуговує одного програміста з окладом 25000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо результат наведений формулою (4.27).

$$С_{Г} = 12 \cdot M \cdot К_{З} = 12 \cdot 25000 \cdot 0{,}2 = 60000 \text{ грн.} \quad (4.27)$$

З урахуванням додаткової заробітної плати отримаємо формулу (4.28).

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 60000 \cdot (1 + 0{,}2) = 72000 \text{ грн.} \quad (4.28)$$

Відрахування на соціальний внесок розраховано за формулою (4.29).

$$С_{ВІД} = С_{ЗП} \cdot 0{,}22 = 72000 \cdot 0{,}22 = 15840 \text{ грн.} \quad (4.29)$$

Амортизаційні відрахування розраховуємо при амортизації 25% та середньої вартості електронно-обчислювальної машини – 20000 грн. Розрахунки наведено формулою (4.30).

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1{,}4 \cdot 0{,}25 \cdot 20000 = 7000 \text{ грн.}, \quad (4.30)$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо формулою (4.31).

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1{,}4 \cdot 20000 \cdot 0{,}08 = 2240 \text{грн.}, \quad (4.31)$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою (4.32).

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_З \cdot К_В = 233 \cdot 8 \cdot 0{,}4 = 745{,}6 \text{ год.}, \quad (4.32)$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою (4.33).

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 745{,}6 \cdot 0{,}2 \cdot 0{,}3 \cdot 5{,}23 = 233{,}97 \text{ грн.}, \quad (4.33)$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнт зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії для юридичних осіб.

Накладні витрати розраховуємо за формулою (4.34).

$$С_Н = Ц_{ПР} \cdot 0{,}67 = 20000 \cdot 0{,}67 = 13400 \text{ грн.} \quad (4.34)$$

Тоді, річні експлуатаційні розраховуємо за формулами (4.35) та (4.36).

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н \text{ грн.} \quad (4.35)$$

$$С_{ЕКС} = 72000 + 12672 + 15840 + 7000 + 233{,}97 + 13400 = 121145{,}97 \text{ грн.} \quad (4.36)$$

Собівартість однієї машино-години ЕОМ розраховуємо за формулою (4.37).

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 121145{,}97 / 745{,}6 = 162{,}48 \text{ грн.} \quad (4.37)$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації:

$$C_{М\,I} = C_{М\text{-}Г} \cdot T_I = 162{,}48 \cdot 853{,}44 = 138666{,}93 \text{ грн.} \quad (4.38)$$

$$C_{М\,II} = C_{М\text{-}Г} \cdot T_{II} = 162{,}48 \cdot 878{,}24 = 142696{,}44 \text{ грн.} \quad (4.39)$$

Накладні витрати складають 67% від заробітної плати. Таким чином, можемо розрахувати так для двох варіантів реалізації ПП:

$$C_{Н\,I} = C_{ЗП\,I} \cdot 0{,}67 = 162559{,}8 \cdot 0{,}67 = 108915 \text{ грн.} \quad (4.40)$$

$$C_{Н\,II} = C_{ЗП\,I} \cdot 0{,}67 = 167283{,}64 \cdot 0{,}67 = 112080 \text{ грн.} \quad (4.41)$$

Отже, вартість розробки ПП за варіантами розраховуємо:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_{М} + C_{Н,} \quad (4.42)$$

$$C_{ПП\,I} = 162559{,}8 + 35763{,}156 + 138666{,}93 + 108915 = 445904{,}886 \text{ грн.} \quad (4.43)$$

$$C_{ПП\,II} = 167283{,}64 + 36802{,}4 + 142696{,}44 + 112080 = 458862{,}48 \text{ грн.} \quad (4.44)$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня:

$$К_{ТЕРj} = К_{Кj} / С_{Фj,} \quad (4.45)$$

$$К_{ТЕР1} = 4{,}89 / 445904{,}886 = 1{,}097 \cdot 10^{-5}, \quad (4.46)$$

$$К_{ТЕР2} = 4{,}53 / 458862{,}48 = 0{,}987 \cdot 10^{-5}. \quad (4.47)$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}=1{,}097 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 1{,}097 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- вибір мови програмування Java;
- використання готових бібліотек обробки та аналізу Big Data у терміналі;
- використання Jupyter Notebook як інтерфейсу розробки.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

Висновки до розділу 4

В даному розділі було проведено повний функціонально-вартісний аналіз системи обробки Big Data хмарних обчислень. Також було оцінено основні функції програмного продукту.

В першій частині функціонально-вартісного аналізу було проаналізовано можливі варіанти розробки системи. Було проведено оцінку різних варіантів реалізації. Внаслідок чого було здійснено розрахунок коефіцієнту технічого рівня та обрано найкращу альтернативу.

Друга частина функціонально-вартісного аналізу була присвячена оцінці вартісної складової реалізації програмного продукту. В рамках неї враховано заробітну плату робітників, витрати на електроенергію та ЕОМ тощо. Результатом було обрано найефективніший варіант, який потребує менших витрат.

ВИСНОВКИ

Хмарні обчислення сьогодні відіграють ключову роль у розвитку сучасних інформаційних технологій. Вони забезпечують всім організаціям, незалежно від їх розмірів, можливість ефективного використання хмарних ресурсів шляхом оренди комп'ютерних потужностей через Інтернет замість налаштування власної інфраструктури. Це дозволяє зменшити витрати на технічну підтримку і забезпечити масштабованість, гнучкість і надійність систем для обробки великих даних (Big Data).

У цій роботі було детально розглянуто технологію MapReduce, що є важливою складовою сучасних хмарних обчислень, яка дозволяє обробляти великі обсяги даних на численних вузлах. MapReduce забезпечує високу масштабованість і паралелізм, що є важливим для обробки великих обсягів даних.

У результаті виконання дипломної роботи було створено програму для точного підрахунку частоти повторюваності слів у заданому текстовому файлі з подальшим збереженням даних у вихідному файлі.

Основний висновок за наданими результатами полягає в тому, що обробка текстів трьох віршів за допомогою методу MapReduce значно зменшила час обчислення.

Враховуючи високу швидкість обробки даних та відсутність потреби у складних обчислювальних ресурсах, метод MapReduce може бути цінним інструментом для різноманітних аналітичних та оброблювальних завдань з використанням великих обсягів даних.

У четвертому розділі було проведено повний функціонально-вартісний аналіз системи обробки великих даних хмарних обчислень. Також було оцінено основні функції програмного продукту.

Такі технології дозволяють організаціям швидко адаптуватися до змін, масштабувати свої ресурси відповідно до потреб і знижувати витрати на

підтримку інфраструктури. Такі переваги роблять Big Data хмарні обчислення важливим інструментом у сучасному бізнесі та науці, забезпечуючи основу для подальшого розвитку інформаційних технологій.

ПЕРЕЛІК ПОСИЛАНЬ

1. MANNING, Patrick; MANNING, Patrick. Challenges of Big Data in History. Palgrave Macmillan UK, 2013.
2. MOHANTY, Hrushikesha. Big data: An introduction. Big Data: A Primer, 2015, 1-28.
3. NAEEM, Muhammad, et al. Trends and future perspective challenges in big data. In: Advances in Intelligent Data Analysis and Applications: Proceeding of the Sixth Euro-China Conference on Intelligent Data Analysis and Applications, 15–18 October 2019, Arad, Romania. Springer Singapore, 2022. p. 309-325.
4. BRADY, Henry E. The challenge of big data and data science. Annual Review of Political Science, 2019, 22: 297-323.
5. ULARU, Elena Geanina, et al. Perspectives on big data and big data analytics. Database Systems Journal, 2012, 3.4: 3-14.
6. DEMCHENKO, Yuri; DE LAAT, Cees; MEMBREY, Peter. Defining architecture components of the Big Data Ecosystem. In: 2014 International conference on collaboration technologies and systems (CTS). IEEE, 2014. p. 104-112.
7. AGRAWAL, Divyakant; DAS, Sudipto; EL ABBADI, Amr. Big data and cloud computing: current state and future opportunities. In: Proceedings of the 14th international conference on extending database technology. 2011. p. 530-533.
8. JI, Changqing, et al. Big data processing in cloud computing environments. In: 2012 12th international symposium on pervasive systems, algorithms and networks. IEEE, 2012. p. 17-23.
9. PURCELL, Bernice M. Big data using cloud computing. Journal of Technology Research, 2014, 5: 1.

10. CHEN, Xuebin, et al. Big data storage architecture design in cloud computing. In: Big Data Technology and Applications: First National Conference, BDTA 2015, Harbin, China, December 25-26, 2015. Proceedings 1. Springer Singapore, 2016. p. 7-14.
11. BUYYA, Rajkumar, et al. Big data analytics-enhanced cloud computing: Challenges, architectural elements, and future directions. In: 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2015. p. 75-84.
12. DEMCHENKO, Yuri, et al. Cloud based big data infrastructure: Architectural components and automated provisioning. In: 2016 International Conference on High Performance Computing & Simulation (HPCS). IEEE, 2016. p. 628-636.
13. POTTER, Jerry L. The massively parallel processor. MIT press, 1985.
14. HASHEM, Ibrahim Abaker Targio, et al. MapReduce: Review and open challenges. Scientometrics, 2016, 109: 389-422.
15. DEWITT, David; STONEBRAKER, Michael. MapReduce: A major step backwards. The Database Column, 2008, 1: 23.
16. TAYLOR, Ronald C. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. BMC bioinformatics, 2010, 11: 1-6.
17. TAO, Yufei; LIN, Wenqing; XIAO, Xiaokui. Minimal mapreduce algorithms. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. 2013. p. 529-540.
18. YANG, Gaizhen. The application of mapreduce in the cloud computing. In: 2011 2nd International Symposium on Intelligence Information Processing and Trusted Computing. IEEE, 2011. p. 154-156.
19. DAHIPHALE, Devendra, et al. An advanced mapreduce: cloud mapreduce, enhancements and applications. IEEE Transactions on Network and Service Management, 2014, 11.1: 101-115.

20. SHIM, Kyuseok. MapReduce algorithms for big data analysis. In: International Workshop on Databases in Networked Information Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 44-48.
21. JI, Changqing, et al. Big data processing in cloud computing environments. In: 2012 12th international symposium on pervasive systems, algorithms and networks. IEEE, 2012. p. 17-23.
22. NABAVINEJAD, Seyed Morteza; GOUDARZI, Maziar. Faster mapreduce computation on clouds through better performance estimation. IEEE Transactions on Cloud Computing, 2017, 7.3: 770-783.
23. SOKIYNA, Munsif Yousef; J AQEL, Musbah; NAQSHBANDI, Omar A. Cloud computing technology algorithms capabilities in managing and processing big data in business organizations: Mapreduce, hadoop, parallel programming. Journal of Information Technology Management, 2020, 12.3: 100-113.
24. BENDAHMANE, Ahmed, et al. Result verification mechanism for MapReduce computation integrity in cloud computing. In: 2012 IEEE International Conference on Complex Systems (ICCS). IEEE, 2012. p. 1-6.
25. XU, Xiaoyong; TANG, Maolin. A new approach to the cloud-based heterogeneous MapReduce placement problem. IEEE Transactions on Services Computing, 2015, 9.6: 862-871.
26. PLIMPTON, Steven J.; DEVINE, Karen D. Mapreduce in MPI for large-scale graph algorithms. Parallel Computing, 2011, 37.9: 610-632.
27. KHANAM, Zeba; AGARWAL, Shafali. Map-reduce implementations: survey and performance comparison. AIRCC's International Journal of Computer Science and Information Technology, 2015, 119-126.
28. BABU, Shivnath, et al. Massively parallel databases and mapreduce systems. Foundations and Trends® in Databases, 2013, 5.1: 1-104.
29. DEDE, Elif, et al. Marissa: Mapreduce implementation for streaming science applications. In: 2012 IEEE 8th International Conference on E-Science. IEEE, 2012. p. 1-8.

30. SMITH, Calvin; ALBARGHOUTHI, Aws. MapReduce program synthesis. Acm Sigplan Notices, 2016, 51.6: 326-340.
31. ABDUL, Jhummarwala; ALKATHIRI, Mazin; POTDAR, M. B. Geospatial Hadoop (GS-Hadoop) an efficient mapreduce based engine for distributed processing of shapefiles. In: 2016 2nd International Conference on Advances in Computing, Communication, & Automation (ICACCA)(Fall). IEEE, 2016. p. 1-7.
32. CASTRO, Eduardo PS, et al. Review and comparison of Apriori algorithm implementations on Hadoop-MapReduce and Spark. The Knowledge Engineering Review, 2018, 33: e9.
33. GIL, Joon-Min; LIM, JongBum; SEO, Dong-Mahn. Design and implementation of mapreduce-based book recommendation system by analysis of large-scale book-rental data. In: Advanced Multimedia and Ubiquitous Engineering: FutureTech & MUE. Springer Singapore, 2016. p. 713-719.
34. MORÁN, Jesús; DE LA RIVA, Claudio; TUYA, Javier. Testing MapReduce programs: A systematic mapping study. Journal of Software: Evolution and Process, 2019, 31.3: e2120.
35. MUPPIDI, Satish; MURTY, M. Ramakrishna. Document clustering with map reduce using Hadoop framework. International Journal on Recent and Innovation Trends in Computing and Communication, 2015, 3.1: 409-413.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import java.io.IOException;
import java.util.*;

public class Word_counter
{
   public static class Token_Mapper extends Mapper <LongWritable, Text, Text, IntWritable>
      {
         private Text wordToken = new Text();
         private static final IntWritable one = new IntWritable(1);

         public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
            {
               // Splitting the string into tokens
               String[] tokens = value.toString()
```

```
                    .replaceAll("\\d+", "")          // Remove numbers
                    .replaceAll("[^a-zA-Z ]", " ")    // Remove punctuation
                    .toLowerCase()                   // Convert to lowercase
                    .trim()                          // Trim spaces
                    .replaceAll("\\s+", " ")
                    .split(" ");

            // Write each word with one occurrence
            for(String word : tokens)
            {
               wordToken.set(word);
               context.write(wordToken, one);
            }
         }
      }

      public static class IntSumReducer extends Reducer <Text, IntWritable, Text,
IntWritable>
      {
         private IntWritable result = new IntWritable();

         public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException
         {
            int sum = 0;

            // Sum up the counts for each word
            for (IntWritable val : values)
              sum += val.get();
```

```
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception
    {
        long startTime = System.currentTimeMillis(); // Record start time

        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();

        if (otherArgs.length < 2)
        {
            System.err.println("Usage: wordcount <input-path> [...] <output-
path>");
            System.exit(2);
        }

        Job job = Job.getInstance(conf, "Word Frequency Counter");
        job.setJarByClass(Word_counter.class);
        job.setMapperClass(Token_Mapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        for (int i = 0; i < otherArgs.length - 1; ++i)
        {
            FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
```

```
            }
            FileOutputFormat.setOutputPath(job,                    new
Path(otherArgs[otherArgs.length - 1]));

            boolean jobCompleted = job.waitForCompletion(true);

            long endTime = System.currentTimeMillis(); // Record end time
            long executionTime = endTime - startTime; // Calculate execution time

            if (jobCompleted) {
               System.out.println("Job completed successfully in " + executionTime +
" milliseconds");
               System.exit(0);
            } else {
               System.out.println("Job failed or was interrupted");
               System.exit(1);
            }
         }
      }
```