НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____2016 р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код та назва спеціальності)

на тему: Системи обробки великих об'ємів даних із застосуванням платформи Apache Hadoop

Виконав (-ла): студент (-ка) __4__ курсу, групи ___ДА-21___
(шифр групи)

Загороднюк Андрій Олександрович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник асистент, к.т.н., Свірін П.В _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант економічний професор, д.е.н. Семенченко Н.В. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент старший викладач каф. ММСА, к.т.н., Мурга М.О. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль ст. викладач Британов О.А. _____
(підпис)

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

КИЇВ 2016

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2016 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Загороднюку Андрію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)) Системи обробки великих об'ємів даних із застосуванням платформи Apache Hadoop

керівник проекту (роботи)) Свірін Павло Володимирович, к.т.н., асистент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 12 травня 2016 р. № 50-ст

2. Строк подання студентом проекту (роботи) 08.06.2016

3. Вихідні дані до проекту (роботи)

Розгорнути 3 віртуальні машини з Apache Hadoop за допомогою Docker
Реалізувати 5 тестових прикладів які використовують MapReduce та HDFS Apache Hadoop
Проаналізувати результати, та вказати на сильні та слабкі сторони Apache Hadoop

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Розглянути задачі розподілення обчислень великих даних.

2. Проаналізувати методики аналізу великих даних.
3. Дослідити принцип та вимоги розподілення обчислень.
4. Проаналізувати існуючи дистрибутиви Apache Hadoop.
5. Розробити актуальні тестові приклади для Apache Hadoop.
6. Знайти сильні та слабкі сторони Apache Hadoop.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Блок-схема алгоритму роботи MapReduce – плакат.
2. Схема структури HDFS – плакат.
3. Графіки тестування прикладів – плакат.
4. Таблиці тестування прикладів – плакат.

6. Консультанти розділів проекту (роботи)·

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Семенченко Н.В., професор, д.е.н.		

7. Дата видачі завдання 01.02.2016

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Збір інформації	15.02.2016	
3	Вивчення розділу розподілення обчислень великих даних	28.02.2016	
4	Вивчення основ Apache Hadoop	10.03.2016	
5	Дослідження MapReduce та HDFS	15.03.2016	
6	Розробка архітектури	25.03.2016	
7	Налаштування Apache Hadoop	25.04.2016	
8	Тестування прикладів та їх аналіз	30.04.2016	
9	Оформлення дипломної роботи	31.05.2016	
10	Отримання допуску до захисту та подача роботи в ДЕК	08.06.2016	

Студент _____ А.О.Загороднюк
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ П.В.Свірін
(підпис) (ініціали, прізвище)

· Консультантом не може бути зазначено керівника дипломного проекту (роботи).

АНОТАЦІЯ

бакалаврської дипломної роботи Загороднюка Андрія Олександровича на тему: «Системи обробки великих об’ємів даних із застосуванням платформи Apache Hadoop»

Дипломна робота присвячена використанню програмного продукту HADOOP для прискорення обчислення об'ємних завдань. У роботі наведено аналіз існуючих прикладів використання платформи. Досліджено час виконання системи HADOOP на прикладі реальних задач, які використовуються у великих компаніях.

Актуальність роботи полягає у тому, що розподіл обчислень зменшує вартість та час, необхідні для виконання завдання. Шляхи подальшого розвитку предмета дослідження - використання більшої кількості більш потужних систем для складних обчислень.

Загальний обсяг роботи: 72 сторінки, 15 рисунків, 12 таблиць, 1 додаток на 6 стр., 15 посилань.

Ключові слова: РОЗПОДІЛ ОБЧИСЛЕНЬ, КЛАСТЕР, APACHE, HADOOP, BIG DATA.

АННОТАЦИЯ

бакалаврской дипломной работи Загороднюка Андрея Александровича на тему: «Системы обработки больших объемов данных с использованием Apache Hadoop»

Дипломная работа посвящена использованию программного продукта HADOOP для ускорения вычисления объемных задач. В работе приведен анализ существующих использования платформы. Исследавано время выполнения задач в HADOOP на примере реальних задач, которые используются в больших компаниях.

Актуальность работи лежит в том, что распределение вычислений уменьшает стоимость и время, необходимые для выполнения задания. Пути дальнейшего развития предмета исследования - использование большего количества более мощных систем для сложных вычислений.

Общий объем работи: 72 страницы, 15 рисунков, 12 таблиц, 1 приложение на 6 стр., 15 библиографических наименований.

Ключевые слова: РАСПРЕДЕЛЕНИЕ ВЫЧИСЛЕНИЙ, КЛАСТЕР, APACHE, HADOOP, BIG DATA.

ABSTRACT

Bachelor thesis Zahorodniuk Andriy O.

On «Systems for processing large amounts of data using APACHE HADOOP»

Thesis is devoted to the use of HADOOP software to speed up the complex tasks. The paper provides an analysis of existing algorithms to test real task using framework. The time of execution of the program with the help of HADOOP via an example of a big companies real tasks has been researched.

Relevance of the work is that the distribution of computation on several computers will reduce the time and required to complete the task. Through further development of the research subject - using more powerful systems for more complex tasks.

The total amount of work: 72 pages, 15 figures, 12 tables, 1 appendix for 6p., 15 references.

Keywords: DISTRIBUTED COMPUTING, CLUSTER, APACHE, HADOOP, BIG DATA.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ **9**

ВСТУП **10**

1. РОЗГЛЯД ЗАДАЧІ РОЗПОДІЛЕННЯ ОБЧИСЛЕНЬ ВЕЛИКИХ ДАНИХ **12**

1.1 Задачі дипломної роботи **12**

1.2 Ціль дипломної роботи **12**

1.3 Актуальність задачі **13**

1.4 Методики аналізу великих даних **14**

1.5 Дослідження принципу та вимог розподілення обчислень **15**

1.5.1 Історія виникнення терміну 15

1.5.2 Цілі побудови розподілених обчислювальних систем 17

1.6 Вимоги до розподілених обчислювальних систем **19**

1.6.1 Прозорість 19

1.6.2 Відкритість 22

1.6.3 Масштабованість 24

1.7 Висновки **30**

2. АРХІТЕКТУРА ПЛАТФОРМИ APACHE HADOOP **31**

2.1 Існуючі дистрибутиви Apache Hadoop **31**

2.2 Приклад використання Apache Hadoop у CERN **33**

2.3 MapReduce **34**

2.4 Потік даних **35**

2.5 Будова HDFS **38**

2.6 Основні концепції HDFS **40**

2.6.1 Блоки 40

2.6.2 Висока доступність HDFS 42

2.6.3 Подолання збоїв і ізоляція 43

2.7 Конфігурація обладнання **44**

2.8 Висновки **45**

3. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ **46**

3.1 Опис задачі **46**

3.2 Розробка актуальних тестових прикладів **46**

3.2.1 Генерація текстових файлів 46

3.2.2 Підрахунок кількості слів (Wordcount) 46
3.2.3 Підрахунок максимальної температури за рік 47
3.2.4 Terasoft тест 47
3.2.5 Розв'язок судоку 47
3.3 Оцінка сильних та слабких сторін Apache Hadoop 47
3.4 Порівняльна таблиця 48
3.5 Графіки порівнянь 49
3.6 Висновки 51
4. ЕКОНОМІЧНО-ОРГАНІЗАЦІЙНА ЧАСТИНА 52
4.1 Вступ 52
4.2 Постановка задачі техніко-економічного аналізу 53
4.3 Обґрунтування функцій програмного продукту 53
4.4 Варіанти реалізації основних функцій 54
4.5 Обґрунтування системи параметрів ПП 56
4.5.1 Опис параметрів 56
4.5.2 Кількісна оцінка параметрів 57
4.5.3 Аналіз експертного оцінювання параметрів 59
4.6 Аналіз рівня якості варіантів реалізації функцій 63
4.7 Економічний аналіз варіантів розробки ПП 64
4.8 Вибір кращого варіанта ПП техніко-економічного рівня 67
4.9 Висновки 68
ВИСНОВКИ 69
ПЕРЕЛІК ПОСИЛАНЬ 71
ДОДАТОК А 73

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

ІТ– інформаційні технології

ОС – операційна система

ПП – програмний продукт

GPU–graphics processing unit

HDFS –hadoop distributed file system

VDS– virtual disk service

SMP – shared-memory multiprocessing

ВСТУП

Ідея «великих даних» не нова, вона виникла у 1970-х, у часи мейнфреймів і пов'язаних з ними наукових комп'ютерних обчислень. Як усім відомо, наукомісткі обчислення, які відрізняються складністю і зазвичай нерозривно пов'язані з необхідністю обробки великих обсягів інформації.

Безпосередньо термін «великі дані» з'явився у вжитку лише в кінці 2000-х. Він відноситься до числа небагатьох назв, що мають цілком достовірну дату свого народження - 3 вересня 2008 року, коли вийшов спеціальний номер найстарішого британського наукового журналу Nature, присвячений пошуку відповіді на питання «Як можуть вплинути на майбутнє науки технології, що відкривають можливості роботи з великими обсягами даних?».

Можна виявити кілька причин, що викликали нову хвилю інтересу до великих даними. Обсяги інформації росли по експоненціальному закону і її левова частка належить до неструктурованих даних. Іншими словами, питання коректної інтерпретації інформаційних потоків ставали все більш актуальними і одночасно складними. Великі гравці придбали найбільш успішні вузькоспеціалізовані компанії і почали розвивати інструменти для роботи з великими даними, кількість відповідних стартапів і зовсім перевершувало всі мислимі очікування.

Поряд зі зростанням обчислювальної потужності і розвитком технологій зберігання можливості аналізу великих даних поступово стають доступними малому і середньому бізнесу і перестають бути виключно прерогативою великих компаній і науково-дослідних центрів. Неабиякою мірою цьому сприяє розвиток хмарної моделі обчислень.

У цей час очікується, що з подальшим проникненням IT в бізнес-середовище та повсякденне життя повинні стати предметом обробки інформаційні потоки продовжать невпинно зростати.

Робота складається з п’яти розділів:

У першому розділі наведено аналіз існуючих систем для вирішення заданої проблеми, описано про особливості та проблематики задачі розподілення обчислень, сформульована постановка і актуальність даної задачі.

У другому розділі було обґрунтовано вибір платформи та мови реалізації програмного продукту, було зроблено аналіз архітектури системи. Було сконфігуровано систему та налаштування платформи.

У третьому були розроблені експерименти за допомогою яких можна було проводити аналіз результатів, отриманих в роботі. Було проведено аналіз отриманих результатів, а саме порівняння швидкодії програмного забезпечення Apache Hadoop у різних сферах, та на різній кількості даних.

У четвертому розділі було проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту.

1. РОЗГЛЯД ЗАДАЧІ РОЗПОДІЛЕННЯ ОБЧИСЛЕНЬ ВЕЛИКИХ ДАНИХ

1.1 Задачі дипломної роботи

Задачами дипломної роботи є:

1. Дослідити та вивчити основні принципи роботи програм для розподілення обчислень.

2. Розробити прикладну задачу, на прикладі якої можна ефективно дослідити завдання розподілених обчислень.

3. Зробити критичний огляд Apache Hadoop для побудови паралельних систем.

4. Проаналізувати існуючі види Apache Hadoop та мови для реалізації даного програмного продукту та вибрати ті, що найкраще підходять для реалізації.

5. Розробити вхідний набір даних для оцінки системи.

6. Зробити висновки щодо отриманих результатів.

1.2 Ціль дипломної роботи

Основною ціллю дипломної роботи є вирішення задачі обробки великих даних використовуючи Apache Hadoop для розподілу обчислень.

Великі дані (англ. Big Data) в інформаційних технологіях — це серія підходів, інструментів і методів обробки структурованих і неструктурованих різноманітних даних великих розмірів для отримання результатів, які легко сприймаються людиною, є ефективними в умовах неперервного приросту, розподілення по численних вузлах обчислювальної мережі. Великі дані — це набори даних такого об'єму, що традиційні інструменти не здатні здійснювати їх охоплення, управління та обробку у задовільний час. Важливо також відзначити те, що під терміном **Big Data** у різних контекстах можуть мати на увазі дані

великого об'єму, технології їх обробки, проекти, компанії, які активно використовують дану технологію. [1]

Розподілені обчислення (розподілена обробка даних) — спосіб розв'язання трудомістких обчислювальних завдань з використанням двох і більше комп'ютерів, об'єднаних в мережу.[2]

Apache Hadoop — вільна програмна платформа і каркас для організації розподіленої обробки великих обсягів даних (що міряється у петабайтах) з використанням парадигми MapReduce, при якій завдання ділиться на багато дрібніших відособлених фрагментів, кожен з яких може бути запущений на окремому вузлі кластера.

1.3 Актуальність задачі

Великі дані призначені для прогнозування. Зазвичай їх описують як частина комп'ютерної науки під назвою «штучний інтелект» (точніше, її розділ «машинне навчання»). Розглядається застосування математичних прийомів до великої кількості даних для прогнозу ймовірностей, наприклад таких, що електронний лист є спамом; що замість слова «коіпу» передбачалося набрати «Копія»; що траєкторія і швидкість руху людини, що переходить дорогу в недозволеному місці, кажуть про те, що він встигне перейти вулицю вчасно і автомобілю потрібно лише трохи знизити швидкість. Але головне - ці системи працюють ефективно завдяки надходженню великої кількості даних, на основі яких вони можуть будувати свої прогнози. Більш того, системи спроектовані таким чином, щоб з часом поліпшуватися за рахунок відстеження найкорисніших сигналів і моделей у міру надходження нових даних.

В якості визначальних характеристик для великих даних відзначають «три V»: обсяг (англ. Volume, в сенсі величини фізичного обсягу), швидкість (англ. Velocity в сенсах як швидкості приросту, так і необхідності високошвидкісної обробки і отримання результатів), різноманіття (англ. Variety, в сенсі можливості одночасної обробки різних типів структурованих і напівструктурованих даних)

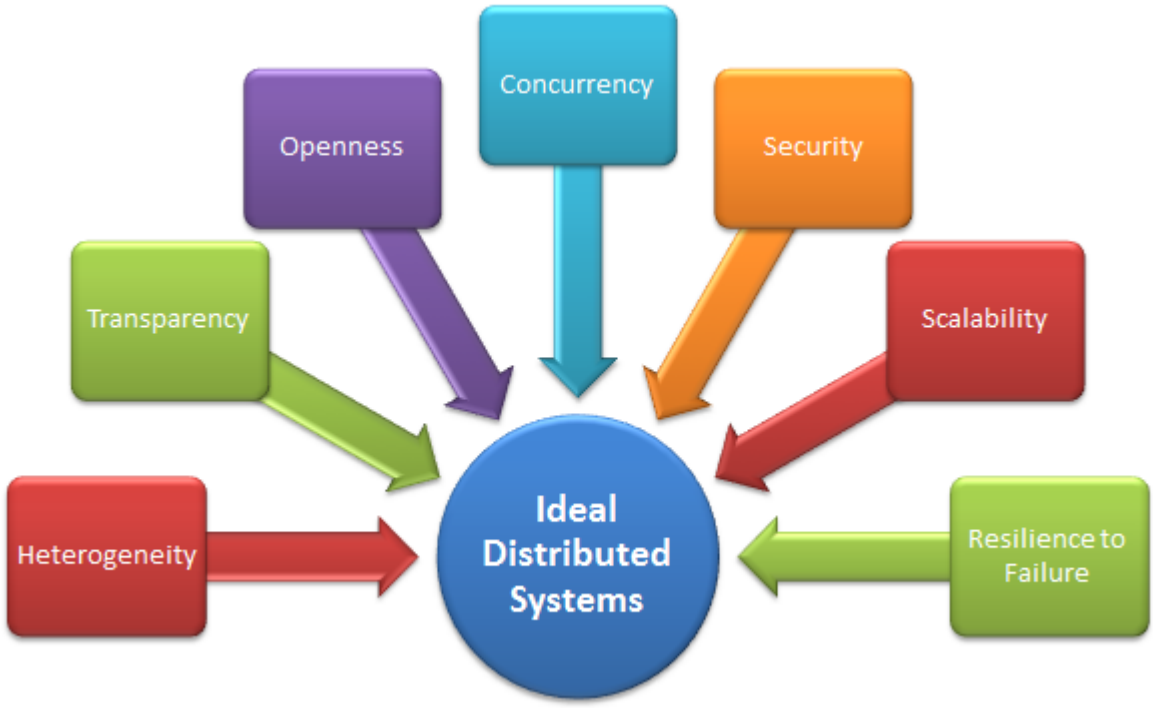

Рисунок 1.1 – Ідеальна розподілена система[3]

1.4 Методики аналізу великих даних

Існує безліч різноманітних методик аналізу масивів даних, в основі яких лежить інструментарій, запозичений з статистики та інформатики (наприклад, машинне навчання). Ось деякі з них:

- методи класу Data Mining: навчання асоціативним правилами (англ. Association rule learning), класифікація (методи категоризації нових даних на основі принципів, раніше застосованих до вже наявним даними), кластерний аналіз, регресійний аналіз;
- краудсорсінг - категоризація та збагачення даних силами широкого, невизначеного кола осіб, залучених на підставі публічної оферти, без вступу в трудові відносини;
- змішання і інтеграція даних (англ. data fusion and integration) - набір технік, що дозволяють інтегрувати різнорідні дані з різноманітних джерел для можливості глибинного аналізу, в якості прикладів таких технік, складових цей клас методів наводяться цифрова обробка сигналів і обробка природного мови (включаючи тональний аналіз);

- машинне навчання, включаючи навчання з учителем і без учителя, а також Ensemble learning (англ.) - використання моделей, побудованих на базі статистичного аналізу або машинного навчання для отримання комплексних прогнозів на основі базових моделей;
- штучні нейронні мережі, мережевий аналіз, оптимізація, в тому числі генетичні алгоритми;
- прогнозна аналітика;
- імітаційне моделювання;
- просторовий аналіз (англ. Spatial analysis) - клас методів, використовують топологічну, геометричну і географічну інформацію в даних;
- статистичний аналіз, як приклади методів наводяться A / B- тестування і аналіз часових рядів;
- візуалізація аналітичних даних - подання інформації у вигляді малюнків, діаграм, з використанням інтерактивних можливостей та анімації як для отримання результатів, так і для використання в якості вихідних даних для подальшого аналізу.

1.5 Дослідження принципу та вимог розподілення обчислень

1.5.1 Історія виникнення терміну

За визначенням, розподілені обчислення (розподілена обробка даних) — спосіб розв'язання трудомістких обчислювальних завдань з використанням двох і більше комп'ютерів, об'єднаних в мережу.

Термін “розподілені обчислення” іноді використовують як використання розподілених систем для вирішення трудомістких задач. У такому контексті розподілені обчислення являють собою особливий випадок паралельних обчислень.

Розподілені обчислення є окремим випадком паралельних обчислень, тобто одночасного розв'язання різних частин одного обчислювального завдання декількома процесорами одного або кількох комп'ютерів. Тому необхідно, щоб

завдання, що розв'язується, було сегментоване — розділене на підзадачі, які можуть обчислюватися паралельно. При цьому для розподілених обчислень доводиться також враховувати можливу відмінність в обчислювальних ресурсах, які будуть доступні для розрахунку різних підзадач. Проте, не кожне завдання можна «розпаралелити» і прискорити його розв'язання за допомогою розподілених обчислень.[4]

Слово «розподілений» в таких термінах, як «розподілена система», «розподілене програмування" та "розподілений алгоритм" спочатку відносилося до комп'ютерних мереж, де окремі комп'ютери були фізично розподілені в деякому географічному регіоні. Ці терміни в даний час використовуються в набагато ширшому сенсі, навіть коли стосуються автономних процесів, які працюють на одному фізичному комп'ютері і взаємодіють один з одним, посилаючи повідомлення. У той час як немає єдиного визначення розподіленої системи, використовуються такі визначальні властивості:

1. Є кілька автономних обчислювальних сутностей, кожна з яких має свою власну локальну пам'ять.

2. Об'єкти взаємодіють один з одним за допомогою передачі повідомлень.

Розподілена система може мати спільну мету, наприклад, вирішення великої обчислювальної задачі. З іншої сторони, кожен комп'ютер може мати свого власного користувача зі своїми індивідуальними потребами, і метою розподіленої системи є координація використання загальних ресурсів або надання послуг зв'язку для користувачів.

Інші типові властивості розподілених систем включають в себе наступні:

1. Система повинна бути толерантною до помилок або несправностей в окремих комп'ютерах.

2. Структура системи (топології мережі, затримки в мережі, кількість комп'ютерів) невідома заздалегідь, може складатися з різних видів комп'ютерів і мережевих зв’язків, а також може змінюватися в ході виконання розподіленої програми.

3. Кожен комп'ютер має тільки обмежене неповне уявлення про систему. Кожен комп'ютер може знати тільки одну частину вхідного сигналу.

1.5.2 Цілі побудови розподілених обчислювальних систем

За останні кілька років розподілені системи ставали все більш популярними і їх роль тільки зростала. Серед основних причин зростання їх значимості можна виділити наступні.

Географічно розподілена обчислювальна середа. Сьогодні в більшості випадків сама обчислювальна середовище за своєю природою являє собою територіально розподілену систему. У якості прикладу можна привести банківську мережу. Кожен банк обслуговує рахунки своїх клієнтів і обробляє операції з ними. У разі ж переказу грошей з одного банку в інший потрібно здійснення міжбанківської транзакції і взаємодія систем банків один з одним. Іншим прикладом географічно розподіленої обчислювальної середовища є всім добре знайома мережа Інтернет.

Вимога збільшення продуктивності обчислень. Швидкодія традиційних однопроцесорних систем стрімко наближається до своєї межі. Різні архітектури (такі як суперскалярна архітектура, матричні і векторні процесори, однокристальні багатопроцесорні системи) покликані збільшувати продуктивність обчислювальних систем за рахунок різних механізмів паралельного виконання команд. Однак всі ці прийоми здатні підвищити продуктивність максимум в десятки разів у порівнянні з класичними послідовними рішеннями. Крім того, масштабованість подібних підходів залишає бажати кращого. Щоб підвищити продуктивність в сотні або тисячі разів і при цьому забезпечувати хорошу масштабованість рішення, необхідно звести воєдино численні процесори і забезпечити їх ефективну взаємодію. Цей принцип реалізується у вигляді великих багатопроцесорних систем і багатомашинних комплексів.

Спільне використання ресурсів. Важливою метою створення і використання розподілених систем є надання користувачам (і додаткам) доступу

до віддалених ресурсів і забезпечення їх спільного використання. Тут ми вважаємо, що термін ресурс відноситься як до компонентів апаратного забезпечення обчислювальної системи, так і до програмних абстракцій, з якими працює розподілена система. Наприклад, користувач комп'ютера 1 може використовувати дисковий простір комп'ютера 2 для зберігання своїх файлів. Або додаток А може використовувати вільну обчислювальну потужність декількох комп'ютерів для прискорення власних розрахунків. Розподілені бази даних та розподілені системи об'єктів можуть бути відмінним прикладом спільного використання програмних компонентів, коли відповідні програмні абстракції розподілені по декількох комп'ютерах і узгоджено обслуговуються декількома процесами, що утворюють розподілену систему.[5]

Відмовостійкість. У традиційних "нерозподілених" обчислювальних системах, побудованих на базі одиничного комп'ютера (можливо високопродуктивного), вихід з ладу одного з його компонентів зазвичай призводить до непрацездатності всієї системи. Такий збій в одному або декількох компонентах системи називають частковим відмовою, якщо він не зачіпає інші компоненти. Характерною рисою розподілених систем, яка відрізняє їх від одиничних комп'ютерів, є стійкість до часткових відмов, тобто система продовжує функціонувати після часткових відмов, правда, трохи знижуючи при цьому загальну продуктивність. Подібна можливість досягається за рахунок надмірності, коли в систему додається додаткове обладнання (апаратна надмірність) або процеси (програмна надмірність), які уможливлюють правильне функціонування системи при непрацездатності або некоректної роботи деяких з її компонентів. В цьому випадку розподілена система намагається приховувати факти відмов або помилок в одних процесах від інших процесів. Наприклад, в системах з потрійним модульним резервуванням (англ. Triple Modular Redundancy, TMR) використовуються три однакових обчислювальних модуля, що здійснюють ідентичні обчислення, а коректний результат визначається простим голосуванням.

1.6 Вимоги до розподілених обчислювальних систем

1.6.1 Прозорість

Ефективна розподілена система повинна володіти такими властивостями: прозорість, відкритість, безпека, масштабованість. Однак варто зазначити, що, незважаючи на уявну простоту і очевидність перерахованих властивостей, їх реалізація на практиці часто є непростим завданням.

Під прозорістю розподіленої системи розуміють її здатність приховувати свою розподілену природу, а саме, розподіл процесів і ресурсів по безлічі комп'ютерів, і представлятися для користувачів і розробників додатків у вигляді єдиної централізованої комп'ютерної системи. Стандарти еталонної моделі для розподіленої обробки у відкритих системах Reference Model for Open Distributed Processing визначають декілька типів прозорості. Найбільш важливі з них перераховані нижче.

Прозорість доступу. Незалежно від способів доступу до ресурсів і їх внутрішнього уявлення, звернення до локальних і віддалених ресурсів здійснюється однаковим чином. На базовому рівні ховається різниця архітектур обчислювальних платформ, але, що більш важливо, досягається угода про те, як ресурси різнорідних машин будуть представлятися користувачам розподіленої системи єдиним чином. Як приклад можна привести прикладний програмний інтерфейс для роботи з файлами, що зберігаються на безлічі комп'ютерів різних архітектур, який надає однакові виклики операцій як з локальними, так і з віддаленими файлами.

Прозорість розташування. Дозволяє звертатися до ресурсів без знання їх фізичного розташування. У цьому випадку ім'я запитуваного ресурсу не повинно давати жодного уявлення про те, де ресурс розташований. Тому важливу роль для забезпечення прозорості розташування грає іменування ресурсів. Наприклад, щоб відправити повідомлення електронної пошти на адресу user@company.com не потрібно знати фізичного місця розташування одержувача, його поштової скриньки або поштового сервера. У свою чергу звернення до файлу

\\computerName\user1\file1 має на увазі знання імені сервера, на якому він розташований, а значить, не є повністю прозорим з точки зору розташування.

Прозорість переміщення. Переміщення ресурсу або процесу в інше фізичне місце розташування залишається непомітним для користувача розподіленої системи. Тут варто зазначити, що виконання вимоги прозорості місцеположення не гарантує прозорості переміщення. Іншими словами, якщо розподілена система приховує місце розташування ресурсу, це не означає, що його можна змінити непомітно для користувача. Наприклад, розподілені файлові системи дозволяють монтувати файлові системи віддалених комп'ютерів в локальний простір імен клієнта, надаючи єдине дерево каталогів і тим самим забезпечуючи прозорість розташування. Однак, якщо файли на віддалених комп'ютерах будуть переміщені в інше місце, в більшій частині розподілених файлових систем вони стануть недоступні для користувача.

Прозорість зміни місця розташування. Більш суворе по відношенню до попереднього - вимога приховати факт переміщення ресурсу під час його використання. Прикладом можуть служити мобільні користувачі, що використовують мобільні телефони. У цьому випадку, якщо розглядати абонента в якості користувача розподіленої системи, а виклик - як її ресурсу, то система буде прозорою з точки зору зміни місця розташування. Дійсно, переміщення "ресурсу" в процесі розмови залишається непомітним для того, хто дзвонить.

Прозорість реплікації. Якщо для підвищення доступності або збільшення продуктивності використовується кілька копій ресурсу (реплік), цей факт залишається прихованим від користувача, і він вважає, що в системі присутній тільки один екземпляр ресурсу. Для забезпечення прозорості реплікації необхідно, щоб всі репліки мали одне і те ж ім'я, очевидно, що не залежить від місця розташування копії ресурсу. Таким чином, системи, які забезпечують прозорість реплікації, також повинні підтримувати і прозорість розташування.

Прозорість одночасного доступу. Дозволяє декільком користувачам одночасно виконувати операції над загальним, спільно використовуваних ресурсом без взаємного впливу один на одного. Інакше кажучи, ховається факт

використання ресурсу іншими користувачами. Варто відзначити, що сам ресурс повинен залишатися в несуперечливому стані, що може досягатися, наприклад, за допомогою механізму блокування, коли користувачі по черзі отримують виключні права на запитуваний ресурс.

Прозорість відмов. Мається на увазі, що система повинна намагатися приховувати часткові відмови, дозволяючи користувачам і додаткам виконати свою роботу незалежно від збоїв в апаратних або програмних компонентах розподіленої системи, а також приховувати факт їх подальшого відновлення. У зв'язку з тим, що будь-який процес, комп'ютер або мережеве з'єднання можуть відмовляти незалежно від інших в довільні моменти часу, кожен компонент розподіленої системи повинен бути готовий до збоїв в інших компонентах і обробляти подібні ситуації відповідним чином.

Ступінь прозорості. Важливо відзначити, що ступінь, до якого кожен із перелічених вище властивостей має бути виконано, може сильно варіюватися в залежності від завдань побудови розподіленої системи.[6] Дійсно, повністю приховати розподіл процесів і ресурсів навряд чи вдасться. Через обмеження в швидкості передачі сигналу, затримка на звернення до ресурсів, територіально віддалених від клієнта, завжди буде більше, ніж до ресурсів, розташованим поблизу. Тому не кожна система в стані або навіть повинна намагатися приховувати всі свої особливості від користувача. Зазвичай, це твердження виражається в пошуку компромісу між прозорістю розподіленої системи і її продуктивністю.

Наприклад, якщо для підвищення відмовостійкості в системі присутні географічно розподілені копії ресурсу, то підтримка їх ідентичного стану для забезпечення прозорості реплікації зажадає набагато більшого часу виконання кожної операції оновлення. Іншими словами, кожна операція оновлення повинна буде поширитися на всі репліки до того, як буде дозволена наступна операція з даними ресурсом. Або, наприклад, багато додатків роблять кілька послідовних спроб зв'язатися з сервером, намагаючись приховати його тимчасову

недоступність, тим самим сповільнюючи роботу системи. Однак, в деяких випадках, наприклад, якщо насправді сервер вийшов з ладу, було б розумніше відразу повідомити користувача про недоступність ресурсу.

1.6.2 Відкритість

Згідно з визначенням, прийнятим комітетом IEEE POSIX 1003.0, відкрита система - це система, яка реалізує відкриті специфікації (стандарти) на інтерфейси[7], служби і підтримувані формати даних, достатні для того, щоб забезпечити:

1. можливість перенесення розробленого прикладного програмного забезпечення на широкий діапазон систем з мінімальними змінами (мобільність додатків, переносимість);
2. спільну роботу (взаємодія) з іншими прикладними програмами на локальних і віддалених платформах (інтероперабельність, здатність до взаємодії);
3. взаємодія з користувачами в стилі, що полегшує останнім перехід від системи до системи (мобільність користувача).

Ключовий момент у цьому визначенні - використання поняття відкрита специфікація, яке, в свою чергу, визначається як загальнодоступна специфікація, яка підтримується відкритим, гласним погоджувальним процесом, спрямованим на постійну адаптацію до нових технологій, і відповідає стандартам. Згідно з цим визначенням відкрита специфікація не залежить від конкретної технології, тобто не залежить від конкретних технічних і програмних засобів або продуктів окремих виробників. Відкрита специфікація однаково доступна будь-якій зацікавленій стороні. Більш того, відкрита специфікація знаходиться під контролем громадської думки, тому зацікавлені сторони можуть брати участь в її розвитку.

У контексті розподілених систем наведене вище визначення означає, що властивість відкритості не може бути досягнуто, якщо специфікація і опис

ключових інтерфейсів програмних компонентів системи не доступні для розробників. Таким чином, ключові інтерфейси повинні бути описані і опубліковані. Важливо відзначити, що тут в першу чергу маються на увазі інтерфейси внутрішніх компонентів системи, а не тільки інтерфейси верхнього рівня, з якими працюють користувачі і додатки. При цьому синтаксис інтерфейсів, тобто імена доступних функцій, типи переданих параметрів, значень, що повертаються, зазвичай описується за допомогою мови визначення інтерфейсів. У свою чергу семантика інтерфейсів, тобто те, що насправді роблять служби, що надають ці інтерфейси, зазвичай задається неформально, за допомогою природної мови.

Подібний опис дозволяє деякому процесу, що потребує певної служби, звернутися до іншого процесу, який реалізує цю службу, через відповідний інтерфейс. Крім того, такий підхід дозволяє створювати кілька різних реалізацій однієї і тієї ж служби, які з точки зору зовнішніх процесів працюватимуть абсолютно однаково. Як наслідок, кілька реалізацій програмних компонентів (можливо, від різних виробників) можуть взаємодіяти і працювати спільно, утворюючи єдину розподілену систему. Таким чином, досягається властивість здатності до взаємодії. Більш того, в цьому випадку прикладний додаток, розроблений для розподіленої системи А, може без змін виконуватися в розподіленій системі В, що реалізує ті ж інтерфейси, що і А. Тобто досягається властивість переносимості.

Ще одна важлива перевага полягає в тому, що відкрита розподілена система потенційно може бути утворена з різнорідного апаратного і програмного забезпечення (знову-таки, можливо, від різних виробників). При цьому додавання нових компонентів або заміна існуючих може здійснюватися відносно легко, не зачіпаючи інших компонентів. На апаратному рівні це виражається в здатності простого підключення до системи додаткових комп'ютерів або заміни існуючих на більш потужні. На програмному - в можливості простого впровадження нових служб або нових реалізацій вже існуючих. Іншими словами, важливою властивістю відкритої розподіленої системи є розширюваність.

1.6.3 Масштабованість

У загальному випадку масштабованість визначають, як здатність обчислювальної системи ефективно справлятися зі збільшенням числа користувачів або підтримуваних ресурсів без втрати продуктивності і без збільшення адміністративного навантаження на її управління.[8] При цьому систему називають масштабуємою, якщо вона здатна збільшувати свою продуктивність при додаванні нових апаратних засобів. Іншими словами, під масштабністю розуміють здатність системи рости разом з ростом навантаження на неї.

Масштабованість є важливою властивістю обчислювальних систем, якщо їм знадобитися працювати під великим навантаженням, оскільки це означає, що вам не доведеться починати з нуля і створювати абсолютно нову інформаційну систему. Якщо у вас є масштабована система, то, швидше за все, вам вдасться зберегти те ж саме програмне забезпечення, просто наростивши апаратну частину.

Для розподілених систем зазвичай виділяють кілька параметрів, які характеризують їх масштаб: кількість користувачів і кількість компонентів, що складають систему, ступінь територіальної віддаленості мережевих комп'ютерів системи один від одного і кількість адміністративних організацій, які обслуговують частини розподіленої системи. Тому масштабованість розподілених систем також визначають за відповідними напрямами:

Навантажена масштабованість. Здатність системи збільшувати свою продуктивність при збільшенні навантаження шляхом заміни існуючих апаратних компонентів на більш потужні або шляхом додавання нових апаратних засобів. При цьому перший випадок збільшення продуктивності кожного компонента системи з метою підвищення загальної продуктивності називають вертикальним масштабуванням, а другий, що виражається в збільшенні кількості мережевих комп'ютерів (серверів) розподіленої системи - горизонтальним масштабуванням.

Географічна масштабованість. Здатність системи зберігати свої основні характеристики, такі як продуктивність, простота і зручність використання, при територіальному рознесенні її компонентів від більш локального взаємного розташування до більш розподіленого.

Адміністративна масштабованість. Характеризує простоту управління системою при збільшенні кількості адміністративно незалежних організацій, що обслуговують частини однієї розподіленої системи.

Складнощі масштабування. Побудова масштабованих систем має на увазі рішення широкого кола завдань і часто стикається з обмеженнями реалізованих в обчислювальних системах централізованих служб, даних і алгоритмів. А саме, багато служб централізовані в тому сенсі, що вони реалізовані у вигляді єдиного процесу і можуть виконуватися тільки на одному комп'ютері (сервері). Проблема такого підходу полягає в тому, що при збільшенні числа користувачів або додатків, що використовують цю службу, сервер, на якому вона виконується, стане вузьким місцем і буде обмежувати загальну продуктивність. Якщо навіть припустити можливість необмеженого збільшення потужності такого сервера (вертикальне масштабування), то тоді обмежуючим фактором стане пропускна спроможність ліній зв'язку, що з'єднують його з іншими компонентами розподіленої системи. Аналогічно, централізація даних вимагає централізованої обробки, приводячи до тих же самим обмеженням. Як приклад переваг децентралізованого підходу можна навести службу доменних імен (англ. Domain Name Service, DNS), яка на сьогоднішній день є однією з найбільших розподілених систем іменування.

Служба DNS використовується в першу чергу для пошуку IP-адрес по доменному імені і обробляє мільйони запитів з комп'ютерів по всьому світу. При цьому розподілена база даних DNS підтримується за допомогою ієрархії DNS-серверів, що взаємодіють за певним протоколом.[10] Якби всі дані DNS централізовано зберігалися б на єдиному сервері, і кожен запит на інтерпретацію

доменного імені передавався б на цей сервер, скористатися такою системою в масштабах всього світу було б неможливо.

Окремо варто відзначити обмеження, створювані застосуванням централізованих алгоритмів. Справа в тому, що централізовані алгоритми для своєї роботи вимагають отримання всіх вхідних даних і тільки після цього роблять відповідні операції над ними, а вже потім поширюють результати всім зацікавленим сторонам. З цієї точки зору проблеми використання централізованих алгоритмів еквівалентні розглянутим вище проблем централізації служб і даних. Тому для досягнення гарної масштабованості слід застосовувати розподілені алгоритми, що передбачають паралельне виконання частин одного і того ж алгоритму незалежними процесами.

На відміну від централізованих алгоритмів, розподілені алгоритми мають такі властивості, які насправді значно ускладнюють їх проектування і реалізацію:

- Відсутність знання глобального стану. Як вже було сказано, централізовані алгоритми мають повну інформацію про стан всієї системи і визначають наступні дії, виходячи з її поточного стану. У свою чергу, кожен процес, який реалізує частину розподіленого алгоритму, має безпосередній доступ тільки до свого стану, але не до глобального стану всієї системи. Відповідно, процеси приймають рішення тільки на основі своєї локальної інформації. Слід зазначити, що інформацію про стан інших процесів в розподіленої системі кожен процес може отримати тільки з повідомлень, що прийшли, і ця інформація може виявитися застарілою на момент отримання. Аналогічна ситуація має місце в астрономії: знання про досліджуваному об'єкті (зірці / галактиці) формуються на підставі світового та іншого електромагнітного випромінювання, і це випромінювання дає уявлення про стан об'єкта в минулому. Наприклад, знання про об'єкт, що знаходиться на відстані п'яти тисяч світлових років, є застарілими на п'ять тисяч років.
- Відсутність загального єдиного часу. Події, що становлять хід виконання централізованого алгоритму повністю впорядковані: для будь-якої пари

подій можна з упевненістю стверджувати, що одне з них відбулося раніше іншого. При виконанні розподіленого алгоритму внаслідок відсутності єдиного для всіх процесів часу, події не можна вважати повністю упорядкованими: для деяких пар подій ми можемо стверджувати, яке з них відбулося раніше іншого, для інших - ні.

- Відсутність детермінізму. Централізований алгоритм найчастіше визначається як строго детермінована послідовність дій, що описує процес перетворення об'єкта з початкового стану в кінцеве. Таким чином, якщо ми будемо запускати централізований алгоритм на виконання з одним і тим же набором вхідних даних, ми будемо отримувати один і той же результат і однакову послідовність переходів зі стану в стан. У свою чергу виконання розподіленого алгоритму носить недетермінований характер через незалежне виконання процесів з різною і невідомою швидкістю, а також через випадкові затримки передачі повідомлень між ними. Тому, незважаючи на те, що для розподілених систем може бути визначено поняття глобального стану, виконання розподіленого алгоритму може лише обмежено розглядатися як перехід з одного глобального стану в інше, тому що для цього ж алгоритму виконання може бути описано іншою послідовністю глобальних станів. Такі альтернативні послідовності зазвичай складаються з інших глобальних станів, і тому немає особливого сенсу говорити про те, що той чи інший стан досягається під час виконання розподіленого алгоритму.
- Стійкість до відмов. Збій в будь-якому з процесів або каналів зв'язку не повинен викликати порушення роботи розподіленого алгоритму.

Для забезпечення географічної масштабованості потрібні свої підходи. Одна з основних причин поганої географічної масштабованості багатьох розподілених систем, розроблених для локальних мереж, полягає в тому, що в їх основі лежить принцип синхронної зв'язку. У цьому виді зв'язку клієнт, що викликає будь-яку службу сервера, блокується до отримання відповіді. Це

непогано працює, коли взаємодія між процесами відбувається швидко і непомітно для користувача. Однак при збільшенні затримки на звернення до віддаленої службі в глобальній системі подібний підхід стає все менш привабливим і, дуже часто, абсолютно неприйнятним.

Інша складність забезпечення географічної масштабованості полягає в тому, що зв'язок в глобальних мережах за своєю природою ненадійна і взаємодія процесів практично завжди є Point-to-point. У свою чергу, зв'язок в локальних мережах є високонадійним і має на увазі використання широкомовних повідомлень, що значно спрощує розробку розподілених додатків. Наприклад, якщо процесу потрібно виявити адресу іншого процесу, який надає певну службу, в локальних мережах йому досить розіслати широкомовне повідомлення з проханням для шуканого процесу відгукнутися до нього. Всі процеси отримують і обробляють це повідомлення. Але тільки процес, що надає необхідну службу, відповідає на отримане прохання, вказуючи свою адресу у відповідному повідомленні. Очевидно, подібна взаємодія перевантажує мережу, і використовувати його в глобальних мережах нереально.

Технології масштабування. У більшості випадків складності масштабування проявляються в проблемах з ефективністю функціонування розподілених систем, викликаних обмеженою продуктивністю її окремих компонентів: серверів і мережевих з'єднань. Існують кілька основних технологій, що дозволяють зменшити навантаження на кожен компонент розподіленої системи. До таких технологій зазвичай відносять поширення (англ. Distribution), реплікацію (англ. Replication) і кешування (англ. Caching).

Поширення передбачає розбиття множини підтримуваних ресурсів на частини з подальшим рознесенням цих частин по компонентах системи. Простим прикладом поширення може служити розподілена файлова система за умови, що кожен файловий сервер обслуговує свій набір файлів із загального адресного простору. Іншим прикладом може бути вже згадувана служба доменних імен

DNS, в якій весь простір DNS-імен розбивається на зони, і імена кожної зони обслуговуються окремим DNS-сервером.

Важливу роль для забезпечення масштабованості грають реплікація і кешування. Реплікація не тільки підвищує доступність ресурсів в разі виникнення часткової відмови, але і допомагає балансувати навантаженню між компонентами системи, тим самим збільшуючи продуктивність. Кешування є особливою формою реплікації, коли копія ресурсу створюється в безпосередній близькості від користувача, що використовує цей ресурс. Різниця полягає лише в тому, що реплікація ініціюється власником ресурсу, а кешування - користувачем при зверненні до цього ресурсу. Однак варто зазначити, що наявність декількох копій ресурсу призводить до інших складнощів, а саме до необхідності забезпечувати їх несуперечливість (англ. Consistency), що, в свою чергу, може негативно позначитися на масштабованості системи.

Таким чином, поширення і реплікація дозволяють розподілити надходження в систему запитів до кількох її компонентів, в той час як кешування зменшує кількість повторних звернень до одного й того ж ресурсу.

Кешування покликане не тільки знижувати навантаження на компоненти розподіленої системи, але і дозволяє приховувати від користувача затримки комунікації при зверненні до віддалених ресурсів. Подібні технології, що приховують затримки комунікації, важливі для досягнення географічної масштабованості системи. До них, зокрема, ще можна віднести механізми асинхронного зв'язку, в яких клієнт не блокується при зверненні до віддаленої служби, а отримує можливість продовжити свою роботу відразу після звернення. Пізніше, коли буде отримано відповідь, клієнтський процес зможе зійти нанівець і викликати спеціальний обробник для завершення операції.

Однак асинхронний зв'язок може бути застосовано не завжди. Наприклад, в інтерактивних додатках користувач змушений чекати реакції системи. У таких випадках можна скористатися технологіями перенесення коду, коли частина коду програми завантажується на сторону клієнта і виповнюється локально для забезпечення швидкого відгуку на дії користувача. Перевага подібних рішень

полягає в тому, що вони дозволяють скоротити кількість мережевих взаємодій і знизити залежність роботи програми від випадкових затримок обміну повідомленнями через мережу. В даний час перенесення коду широко використовується в Інтернеті в формі аплетів Java і Javascript.

1.7 Висновки

У даному розділі розглянуто та досліджено задачу побудови системи розподілених обчислень. У ході дослідження було описано особливості та основну проблематику при реалізації даної задачі, розглянуто існуючі системи для вирішення проблеми.

Враховуючи предмет дослідження та специфіку даного завдання, було сформульовано постановку задачі, де описано основні проблеми, які треба вирішити для створення програмного продукту за допомогою якого можна дослідити ефективність застосування системи розподілених обчислень HADOOP.

2. АРХІТЕКТУРА ПЛАТФОРМИ APACHE HADOOP

2.1 Існуючі дистрибутиви Apache Hadoop

У своєму первісному варіанті, Hadoop була розроблена як проста інфраструктура зберігання однократного запису. Але через роки вона перетворилася, розширивши рамки простої потужності веб-індексації. Ґрунтуючись на моделі MapReduce Google, Hadoop призначений для зберігання і обробки великих обсягів і різноманітності даних, які можуть знаходитись в декількох серверах.

У той час як розподілена файлова система Hadoop (в HDFS) допомагає розділити всі вхідні дані і зберігати їх на кількох вузлах, компонент MapReduce полегшує одночасну обробку даних по декільком вузлам.

Гнучка модульна архітектура Hadoop дозволяє додавати нові функціональні можливості для виконання різноманітних завдань з обробки великих даних. Кілька постачальників скористалися Hadoop та оптимізували свої коди, щоб змінити або розширити функціональні можливості. У процесі вони змогли виправити деякі з недоліків, властивих Apache Hadoop. Три компанії, які дійсно виділяються в роботі: Cloudera, MapR і Hortonworks.
Cloudera був протягом самого довгого часу з моменту створення Hadoop. Hortonworks утворився пізніше. У той час як Cloudera і Hortonworks 100 відсотків з відкритим вихідним кодом, більшість версій MapR оснащені фірмовими модулями. Кожен постачальник має свою унікальну сильні та слабкі сторони, у кожного є певні пересічні функції, а також.

Для того, щоб побудувати дійсно інформаційно орієнтовані продукти, де рішення на базуються на основі даних, а не емпіричних робіт, компанії потрібно рішення для управління даними, яке не тільки пропонує надійне управління даними, але також легко кероване і легко інтегрується з існуючою інфраструктурою підприємства. Тому на практиці використовують готові дистрибутиви, кожен з яких має свої переваги та недоліки.

Порівняємо дистрибутиви Hadoop(MapR не будемо розглядати тому, що нас цікавить файлова система HDFS).

Таблиця 2.1 – Порівняння дистрибутивів

Дистрибутив	Час розгортання	Інтерфейс користувача	Наявність додаткового ПО
Apache	Декілька годин. Користувач має бути впевниним користувачем Linux.	Залежить від дистрибутива Linux, який вибере користувач.	Можна встановлювати самому, але це займе додатковий час.
Cloudera	Після завантаження готова до використання	Графічний на базі CentOS.	Підтримка веб-сервісів.
HortonWorks	Після завантаження готова до використання	Термінал. Веб-інтерфейс.	Нема
Docker	Після завантаження готова та налаштування, яке займає 5 хвилин готова до використання.	Графічний інтерфейс на базі ОС, яку обере користувач.	Нема

2.2 Приклад використання Apache Hadoop у CERN

CERN, Європе́йська організація з я́дерних дослі́джень (ЦЕРН) (фр. l'Organisation européenne pour la recherche nucléaire) — міжнародний дослідницький центр європейської спільноти, найбільша у світі лабораторія фізики високих енергій, відома своїм прискорювачем елементарних частинок, більш відомий як Великий адронний колайдер. Прискорювач пролягає у тунелі довжиною 27км та на глибині близько 150м.

Кожен рік ЦЕРН записує більше 50 петабайт експериментальних даних у свої сховища. Дані записуються на спеціальні стрічки тому, що вони набагато дешевші у використанні, ніж жорсткі диски, які споживають багато електрики. Не має потреби одночасного доступу до усіх даних. Тому у ЦЕРНі Hadoop використовують для зберігання та обробки метаданих.

Сьогодні у ЦЕРНі працює три Hadoop кластери:

- CASTOR Cluster ~ 10 серверів (приблизно 108 терабайт логів)
- ATLAS Cluster ~20 серверів (працює з індексами каталогів експериментальних даних)
- Monitoring Cluster with ~10 серверів (логує події з CERN Computer Center)

Метадані записують дані о фізичній події, яка виникла у ВАК. Кожного року виникає близько 2 млрд. справжній подій, та близько 4 млрд. відбуваються у штучних умовах. Щоб швидко опрацювати таку кількість даних у ЦЕРНі використовують зв’язку Apache Hadoop + Oracle DB + “R”(programming language).

Розглянемо приклад, який надав ЦЕРН у відкритий доступ. Як ми бачимо з рисунку 2.1, у цьому прикладі шукають необхідні записи з логів стрічок, на яких зберігається інформація.

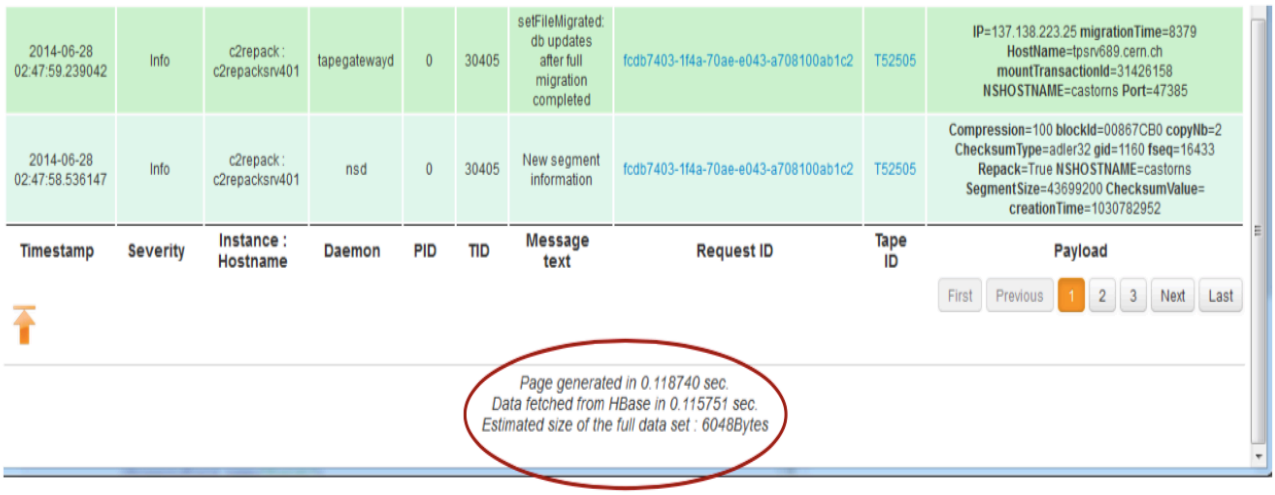

Рисунок 2.1– Приклад пошуку запису з логів

Отже можна зробити висновок, що Apache Hadoop якісний продукт, який використовують великі компанії, дослідницькі центри, які володіють дійно великим об’ємом інформації.

2.3 MapReduce

MapReduce - модель програмування, орієнтована на обробку даних. Ця модель проста, але не настільки, щоб в її контексті не можна було реалізувати корисні програми. Hadoop дозволяє запускати програми MapReduce, написанні на різних мовах. Наприклад: Java, Ruby, Python і C ++. Але найважливіше полягає в тому, що програми MapReduce паралельні за своєю природою, а отже, великомасштабний аналіз даних стає доступним для всіх, у кого в розпорядженні є дастатньо комп'ютерів. Переваги MapReduce в повній мірі проявляються в роботі з великими наборами даних.[11]

Робота MapReduce заснована на розбитті обробки даних на три фази: фазу відображення (map), фазу тасування(shuffle) та фазу згортки (reduce). Кожна фаза використовує в якості вхідних і вихідних даних пари «ключ - значення», типи яких вибираються програмістом. Програміст також задає дві функції: функцію відображення і функцію згортки.

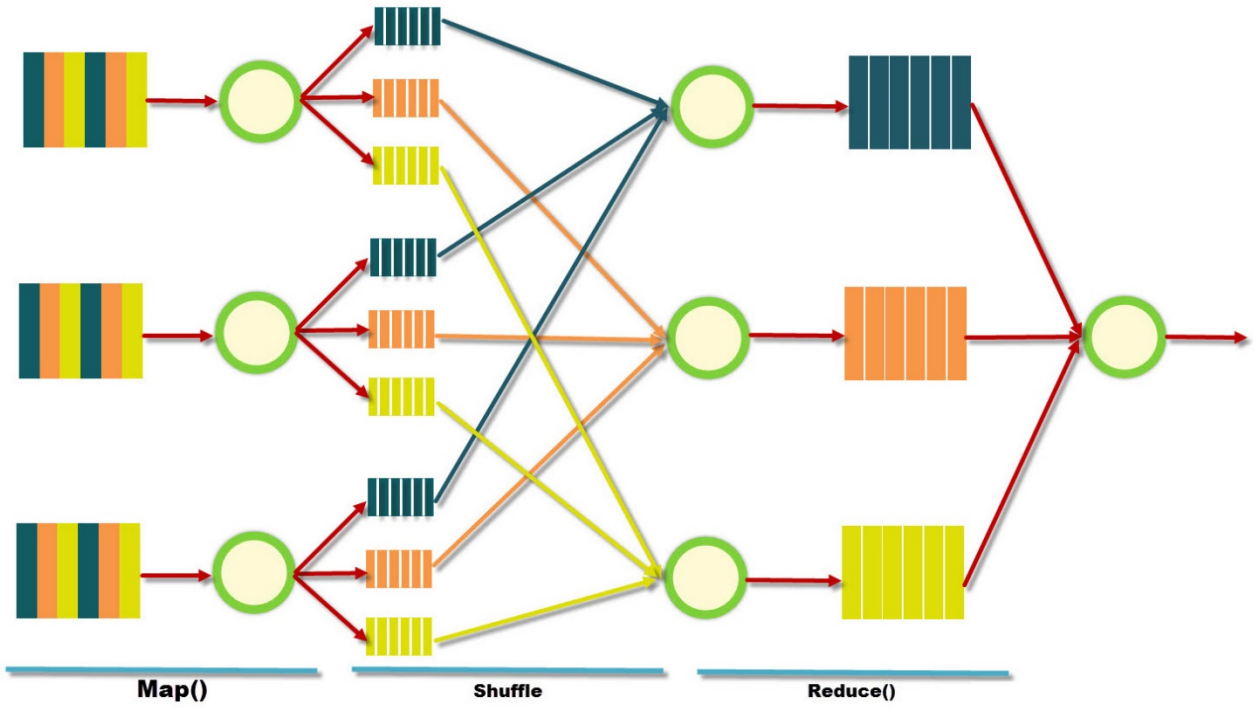

Рисунок 2.2 – схема роботи MapReduce[12]

2.4 Потік даних

Для початку трохи термінології. Завдання MapReduce є одиницю роботи, яку хоче виконати клієнт: воно складається з вхідних даних, програми MapReduce і конфігураційної інформації. Щоб виконати завдання, Hadoop розбиває його на завдання: які діляться на два типи: за дачі відображення і завдання згортки.

Вузли, що керують процесом виконання завдання, діляться на два типи: трекер завдань і кілька трекерів завдань. Трекер завдань координує всі завдання, що виконуються системою; для цього він планує виконання завдань на трекерах завдань. Трекери завдань виконують завдання і відправляють звіти про хід роботи трекера завдань, який відстежує загальний прогрес кожного завдання. Якщо спроба виконання завдання завершується невдачею, трекер може заново спланувати її на іншому трекері.

Hadoop ділить вхідні дані завдань MapReduce на фрагменти фіксованого розміру, звані сплітами. Hadoop створює для кожного спліта одну задачу

відображення, яка виконує певну користувачем функцію відображення для кожного запису в сплите.

Наявність багатьох сплітів означає, що час, витрачений на обробку кожного спліта, малий у порівнянні з часом обробки всіх вхідних даних. Таким чином, якщо спліти будуть оброблятися паралельно, навантаження буде краще збалансоване при малому розмірі сплітів, оскільки більш швидка машина зможе обробити більше сплітів, ніж повільна. Навіть якщо машини ідентичні, збійні процеси або інші одночасно виконувані завдання змушують звернути увагу на розподіл навантаження, а якість розподілу навантаження зростає зі зменшенням сплітів.

З іншого боку, при дуже малому розмірі спліта витрати на управління сплітами і створення завдань відображення займають дуже велику частку загального часу виконання. Для більшості завдань бажаний розмір спліта зазвичай відповідає розміру блоку HDFS - 64 Мбайт за замовчуванням, хоча цю величину можна змінити для кластера (для всіх новостворюваних файлів) або задати при створенні файлу.

Hadoop намагається по можливості виконувати завдання відображення в вузлі, в кото ром вхідні дані зберігаються в HDFS. Цей принцип, званий оптимізацією локальності даних, прагне знизити навантаження на цінну пропускну спроможність кластера. Проте в деяких випадках усі три вузли, на яких розміщені репліки блоку HDFS з вхідним сплітом завдання відображення, зайняті виконанням інших завдань відображення. Тоді планувальник завдань шукає вільний слот відображення на вузлі в тому ж сегменті, що і один з блоків. Вкрай рідко навіть таке рішення виявляється неможливим, і тоді використовується внесегментний вузол, що призводить до межсегментной передачі даних по мережі.

Зрозуміло, чому оптимальний розмір спліта збігається з розміром блоку: це максимальний розмір даних, які можуть гарантовано зберігатися на одному вузлі. Якщо спліт займає два блоки, навряд чи знайдеться вузол HDFS, на якому зберігаються обидва блоки, і частина спліта доведеться передавати по мережі

вузла, на якому виконується завдання відображення. Звичайно, в цьому випадку виконання стане менш ефективним, ніж при виконанні всієї завдання відображення з використанням локальних даних.

Завдання відображення записують свої вихідні дані на локальний диск, а не в HDFS. Оскільки висновок відображень є проміжним: він оброблюється завданнями згортки для отримання підсумкового висновку, і після завершення завдання, висновок завдань відображення можна видалити. Таким чином, зберігання його в HDFS з реплікацією неефективно. Якщо на вузлі, на якому працює завдання відображення, відбудеться збій до того, як висновок відображення буде отримано завданням згортки, Hadoop автоматично заново виконає завдання відображення на іншому вузлі, щоб відтворити висновок завдання відображення.

Завдання згортки не користуються перевагами локальності даних, вхідні дані одного завдання згортки зазвичай утворюються з вихідних даних всіх завдань відображення. У нашому прикладі одна задача згортки отримує дані від всіх завдань відображення. Таким чином, відсортований результат повинен бути переданий по мережі вузла, на якому працює завдання згортки. На цьому вузлі дані об'єднуються і передаються функції згортки. Вивід згортки зазвичай зберігається в HDFS з міркувань надійності[13] с 25-26. Для кожного блоку HDFS, що містить вихідні дані згортки, перша репліка зберігається в локальному вузлі, а інші репліки зберігаються у внесегментних вузлах. Відповідно, запис виведення згортки призводить до витрачання пропускної здатності мережі, але не більше, ніж для звичайного конвеєра запису HDFS.

Увесь потік даних для одного завдання згортки зображений на рис. 2.1. Пунктирні прямокутники позначають вузли, пунктирні стрілки - передачу даних вузлів, а жирні стрілки - передачу даних між вузлами.

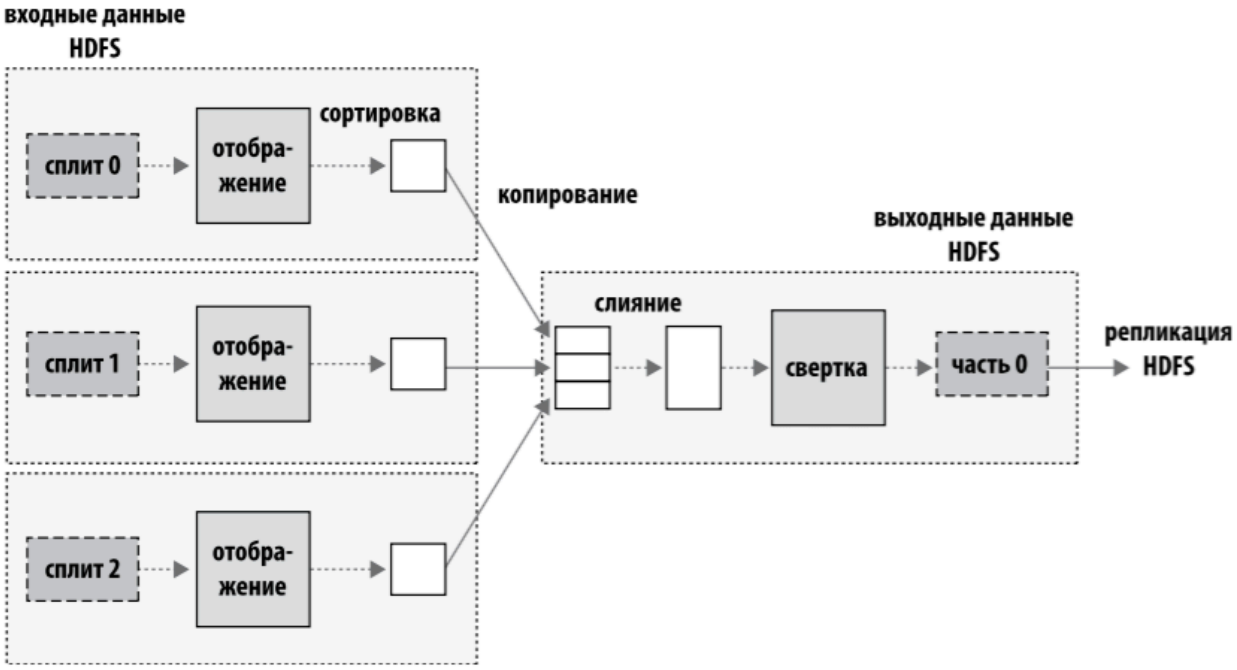

Рисунок 2.3 – Потік даних MapReduce для однієї задачі згортки[14]

2.5 Будова HDFS

Коли набір даних переростає ємність однієї фізичної машини, його доводиться розподіляти по декількох різних машинах. Файлові системи, які управляють зберіганням даних в мережі, називаються розподіленими файловими системами. Оскільки вони працюють в мережевому середовищі, проектувальнику доводиться враховувати всі складності мережевого програмування, тому розподілені файлові системи складніше звичайних дискових файлових систем[13], с 75-76. Наприклад, одна з найсерйозніших проблем - зробити так, щоб файлова система переживала збої окремих вузлів без втрати даних.

Hadoop поставляється з розподіленої файлової системою, яка називається HDFS (Hadoop Distributed Filesystem). Іноді - в старій документації або конфігураціях або в неформальному спілкуванні - також зустрічається скорочення «DFS»; воно означає те ж саме. HDFS - основна файлова система Hadoop, але в Hadoop також реалізована абстракція файлової системи(наприклад Amazon S3).

Файлова система HDFS спроектована для зберігання дуже великих файлів з потокової схемою доступу до даних в кластерах звичайних машин.

Дуже великі файли. Під «дуже великими» в цьому контексті маються на увазі файли, розмір яких складає сотні мегабайт, гігабайт і терабайт. Зараз існують кластери Hadoop, в яких зберігаються петабайти даних.

Потоковий доступ до даних. В основу HDFS закладена концепція однократного запису / багаторазового читання як найефективніша схема обробки даних. Набір даних зазвичай генерується або копіюється з джерела, після чого з ним виконуються різні аналітичні операції. У кожній операції задіються велика частина набору даних (або весь набір), тому час читання всього набору даних важливіше затримки читання першого запису.

Звичайне обладнання. Hadoop не вимагає дорогого устаткування високої надійності. Система спроектована для роботи на стандартному обладнанні (загально доступне обладнання, яке може бути придбано у багатьох фірм). Технологія HDFS спроектована таким чином, щоб у разі відмови система продовжувала роботу без помітного переривання.

Також слід виділити області застосування, для яких в даний час HDFS підходить не найкращим чином (при тому, що в майбутньому ситуація може змінитися):

Швидкий доступ до даних. Додатки, що вимагають доступу до даних з мінімальною затримкою (в діапазоні десятків мілісекунд), погано поєднуються з HDFS. Нагадаємо, що система HDFS оптимізована для забезпечення високої пропускної спроможності передачі даних, за яку доводиться розплачуватися забарінням доступу.

Численні дрібні файли. Так як вузол імен зберігає метадані файлової системи в пам'яті, межа кількості файлів в файлову систему визначається обсягом пам'яті вузла імен. Як показує досвід, кожен файл, каталог і блок займають близько 150 байт. Таким чином, наприклад, якщо у вас є мільйон файлів, кожен з яких займає один блок, для зберігання інформації потрібно не

менше 300 Мбайт пам'яті. Зберігання мільйонів файлів ще прийнятно, але мільярди файлів вже виходять за межі можливостей сучасного устаткування.

Множинні джерела записи, довільні модифікації файлів. Запис в файли HDFS може виконуватися тільки одним джерелом. Запис завжди здійснюється в кінець файлу. Підтримка множинних джерел запису або модифікації з довільним зміщенням у файлі відсутній.

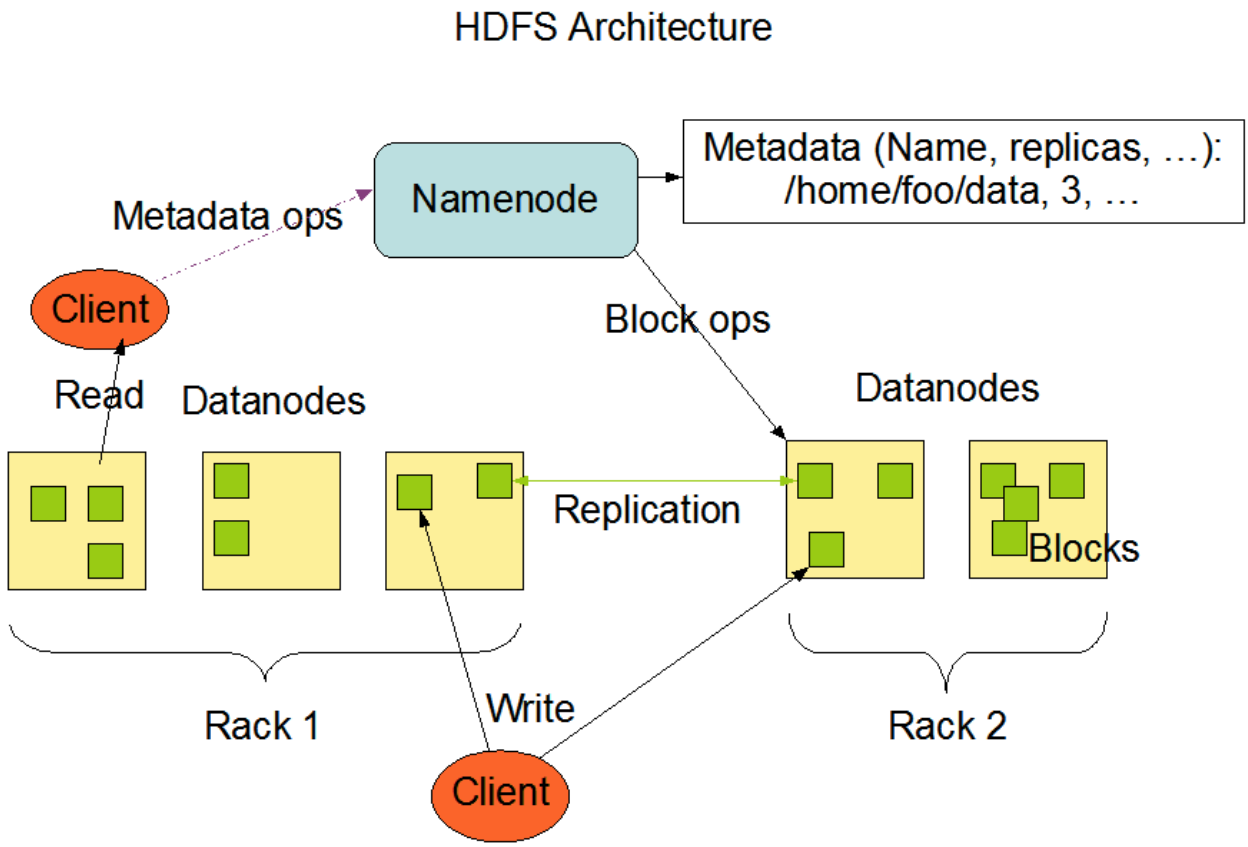

Рисунок 2.4 – Будова HDFS[15]

2.6 Основні концепції HDFS

2.6.1 Блоки

З дисковим пристроєм зв'язується розмір блоку - мінімальний обсяг даних, які використовуються в операції читання або запису. Однодискові файлові системи використовують цю обставину, повертаючи дані в блоках, розмір яких кратний розміру дискового блоку. Розмір блоку файлової системи зазвичай становить кілька кілобайт, тоді як розмір дискового блоку зазвичай дорівнює 512 байтам. Як правило, всі ці нюанси повністю прозорі для користувачів файлових

систем, які просто читають або записують в файли дані довільної довжини. Однак програми супроводу файлових систем (такі, як df і fsck) працюють на рівні блоків файлової системи.

У HDFS теж існує концепція блоку, але він має істотно більший розмір - за замовчуванням 64 Мбайт. Як і в однодискових файлових системах, файли HDFS розбиваються на блокові фрагменти, збережені незалежно один від одного. На відміну від однодискових файлових систем, файл HDFS, менший одного блоку, не займає весь простір, виділений під блок.

Абстракція блоків в розподіленій файловій системі має кілька перваг. Перша перевага найбільш очевидна: файл може бути більше будь-якого окремого диска в мережі. Блоки файлу зовсім необов'язково зберігати на одному диску, вони можуть використовувати будь-які диски в кластері. Більш того, при зберіганні файлу у кластері HDFS його блоки можуть бути розподілені по всіх дисках кластера (хоча ця ситуація дещо нетипова).

Друга перевага: використання в якості абстрактної одиниці блоку замість файлу спрощує підсистему зберігання. Простота - особливо важлива у розподілених системах з їх різноманітними режимами відмов. Використання блоків в підсистемі зберігання даних спрощує зберігання (так як блоки мають фіксований розмір, система може легко обчислити кількість блоків, які можуть зберігатися на заданому диску). Блок - всього лише фрагмент даних, призначений для збереження, з ним не потрібно зберігати метадані файлу - скажімо, інформацію про дозволи доступу, які можуть окремо оброблюватися іншою системою.

Крім того, блоки добре вписуються в механізм реплікації - вони покращують відмовостійкість і доступність системи. Для захисту від пошкоджених блоків і збоїв дисків/машин, кожен блок реплікується на невеликій кількості фізично розділених машин (як правило, трьох). Якщо блок стає недоступним, його копія зчитується з іншого місця, причому це відбувається прозоро для клієнта. Блок, який став недоступним через пошкодження даних або збою обладнання, копіюється з альтернативних сховищ на інші працездатні

машини, щоб довести фактор реплікації до нормального рівня. Крім того, деякі програми можуть встановити високий фактор реплікації для блоків часто запитуваного файлу, щоб поліпшити розподіл навантаження зчитування в межах кластера.

2.6.2 Висока доступність HDFS

Поєднання реплікації метаданих вузлів імен в декількох файлових системах і використання вторинного вузла імен для створення контрольних точок захищає від втрати даних, але не забезпечує високої доступності системи. Вузол імен як і раніше залишається єдиною точкою збою. Якщо в цій точці відбувається збій, всі клієнти - включаючи завдання MapReduce - втрачають можливість читання, записи або отримання списку файлів, тому що вузол імен є єдиним зховищем метаданих та інформації про відповідність між файлами і блоками[13], с126-127. У цьому випадку вся система Hadoop фактично втрачає працездатність, поки не буде запущений новий вузол.

Щоб відновитися після збою вузла імен в такій ситуації, адміністратор запускає новий основний вузол імен з однією з реплік метаданих файлової системи і налаштовує вузли даних і клієнтів на використання нового вузла. Новий вузол імен не може обслуговувати запити до того, як 1) в пам'ять буде завантажений образ простору імен, 2) буде відтворено журнал змін і 3) від вузлів даних буде отримано достатньо звітів для виходу з безпечного режиму. У великих кластерах з безліччю файлів і блоків час запуску вузла імен може становити 30 і більше хвилин.

Довге відновлення також створює проблеми для повсякденного супроводу. Більш того, непередбачені збої вузлів імен настільки рідкісні, що на практиці заплановані простої грають більш важливу роль.

У серії випусків Hadoop 2.x ця ситуація була виправлена додаванням підтримки високої доступності (HA, High Availability) HDFS. У цій реалізації використовуються два вузли імен в конфігурації «активний/резервний». У разі

збою активного вузла імен резервний вузол бере на себе обов'язки по обслуговуванню клієнтських запитів без скільки-небудь помітної перерви.

2.6.3 Подолання збоїв і ізоляція

Переходом від активного вузла імен до резервного управляє новий компонент системи - контролер подолання збоїв. Контролери подолання збоїв замінні. Перша реалізація використовує ZooKeeper для перевірки того, що активний тільки один вузол імен. На кожному вузлі імен працює полегшений процес контролера подолання збоїв, завдання якого полягає у відстеженні збоїв на вузлі (з використанням простого механізму «перевірки пульсу») та ініціюванні подолання збою в разі відмови.

Подолання збоїв також може ініціюватися вручну адміністратором наприклад, при регулярному технічному обслуговуванні. Це називається коректним подоланням збоїв, так як контролер подолання збоїв забезпечує перемикання ролей вузлів імен. Однак в разі аварійного подолання збоїв неможливо бути повністю впевненим у тому, що зіпсований вузол імен перестав працювати. Наприклад, повільна робота мережі або мережевого розділу може призвести до переходу в стан подолання збоїв, хоча раніше активний вузол імен продовжує працювати і вважає, що він все ще є активним. Реалізація високої доступності прикладає великі зусилля, щоб запобігти можливим пошкодженням даних зі сторони раніше активного вузла імен - цей метод називається ізоляцією. Система використовує різні механізми ізоляції, у числі яких є знищення процесу вузла імен, відгук його прав доступу до каталогу загального зберігання даних (зазвичай з використанням команди NFS конкретного розробника) і відключення його мережевого порту командою дистанційного керування. Як крайній захід раніше активний вузол імен може бути ізольований за допомогою примусового відключення живлення на керуючому комп'ютері зі спеціального розподілителя електроживлення.

Подолання збоїв на стороні клієнта виконується прозоро клієнтською бібліотекою. Найпростіша реалізація використовує для управління подоланням

збоїв дані конфігурації на стороні кліента. HDFS URI використовує логічне ім'я хоста, що відображається на пару адрес вузлів імен (в файлі конфігурації), а клієнтська бібліотека по черзі випробує кожну адресу, поки операція не завершиться успіхом.

2.7 Конфігурація обладнання

Перейдемо до налаштування нашого обладнання.

Ми маємо у своєму розпорядженні ПК на якому встановлено Windows 8.1. Процесор Intel i5 2.8Hz – 4 ядра.

HDD – 3 Tb, 7200 об/хв.

RAM – DDR3, 8 Гб, 1600.

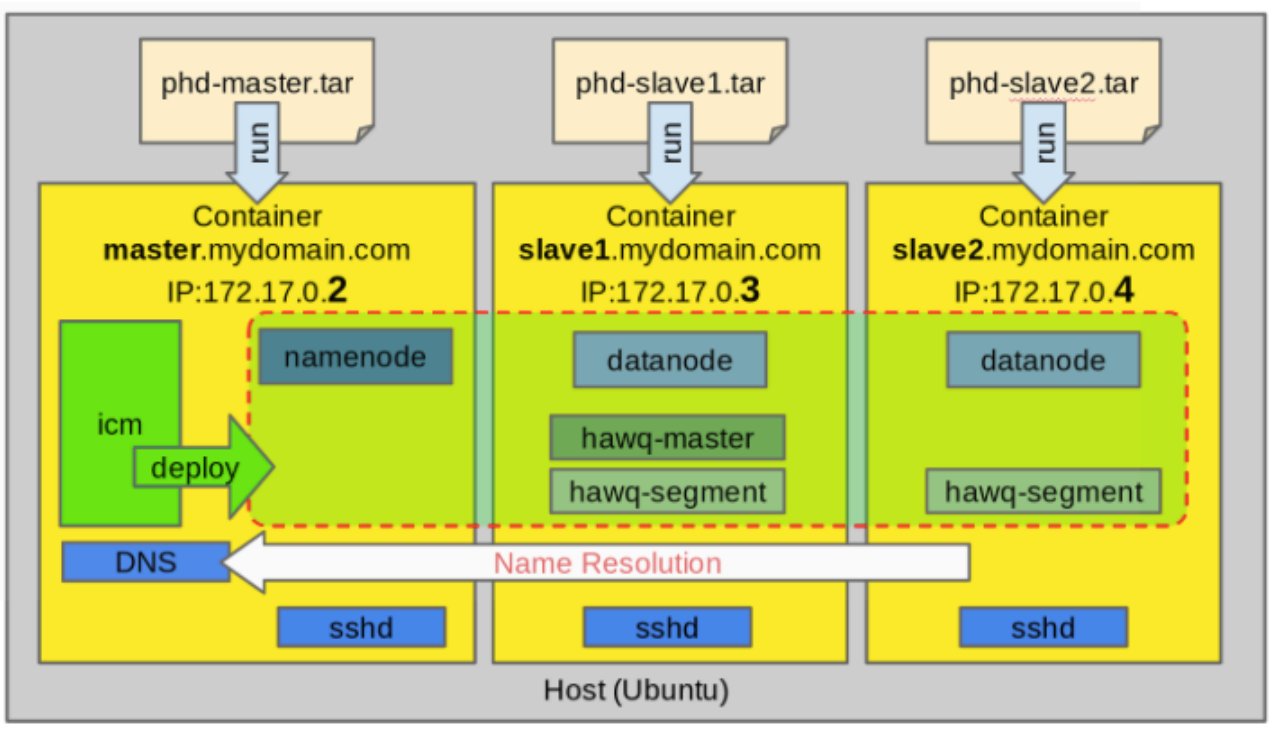

Рисунок 2.5 – Архітектура проекту[16]

Проаналізувавши параметри нашого ПК, було прийнято рішення встановити VirtualBox від Oracle, на який у свою чергу встановимо Ubuntu 14.04. За допомогою Dockera ми швидко розгорнули 3 віртуальні машини Linux Red Hat edition на яких було встановлено Hadoop. Отже, ми отримали кластер із 3

вузлів з одним Master та двума Slave. На кожну віртуальну машину ми виділили 2 Гб RAM, ще 2 Гб ми залишили для основної ОС.

До віртуальної машини мастера ми можемо підключатись за допомогою ssh. У налаштуваннях віртуальної машині мастера ми прописали у hosts адреси віртуальних машин слейвів, щоб мастер мав доступ до своїх дочірних вузлів і міг спілкуватися з ними. Загрузити набори даних з відкритих джерел можна за допомогою команди wget.

2.8 Висновки

У даному розділі було проаналізовано особливості архітектур програмного забезпечення та пакету розробки додатків до системи Hadoop. Було розглянуто основні компоненти Apache Hadoop, їх роботу та взаємодію. Було з’ясовано, що у ході виконання дипломної роботи буде розгорнуто 3 віртуальні машини, які будуть складати тестовий кластер. У ході дослідження було описано реалізацію даної архітектури і описано про реалізацію кожного рівня.

3. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Опис задачі

Метою даної роботи є дослідження використання фреймворку Hadoop, що не є можливим без опису тестових прикладів. У цьому розділі будуть описані типи задач які ми будемо розв'язувати, безпосередньо алгоритми розв'язання, виведені результати роботи фреймворку.

3.2 Розробка актуальних тестових прикладів

Отже, для показовості оберемо три класичні задачі для Hadoop: генерація великих текстових файлів на кожному із вузлів, підрахунок кількості слів у згенерованих текстових файлах, обробка метеорологічних даних за 10 років та вивід максимальної температури за кожен рік.

3.2.1 Генерація текстових файлів

Перша наша задача – сформувати великі текстові файли на кожному з вузлів. Ця задача буде нам дуже корисна. По-перше, ми згенеруємо необхідну нам кількість даних для інших задач. По-друге, ми протестуємо вузьке місце у будь-якій програмі – читання/запис у файл.

3.2.2 Підрахунок кількості слів (Wordcount)

Після того, як мі згенерували набори даних, зробимо вирішимо просту задачу. Підрахунок кількості заданих слів у файлі. Ця задача несе практичний зміст. Ми симулюємо аналіз логів сервера рекламної компанії, щоб покращити таргетинг. Або щоб порахувати середню ціну реклами по містам. Великі онлайн магазини увесь час використовують wordcount, щоб рекомендувати необхідні товари користувачам.

3.2.3 Підрахунок максимальної температури за рік

Метеорологічні станції записують величезну кількість інформації кожен день. Щоб науковці могли передбачити погоду, відслідкувати зміни клімату, попередити людей про надзвичайну ситуацію, стихію.

Кожна станція збирає свої дані за рік в один файл та відправляє на один з вузлів кластера. У кінці ми отримуємо тисячі файлів на вузлах кластеру, які необхідно проаналізуати та зробити один висновок. У нашій задачі ми будемо аналізувати дані за 2 роки від тисяч метеорологічних станцій. Дані будуть записані у стандартизованому виді. Ми їх будемо обробляти і у кінці виведемо максимальну температуру за кожен рік.

3.2.4 Terasoft тест

Найвідоміший тест, який використовують, щоб порівняти потужність кластерів. Суть завдання: зробити сортування 1 Tb даних за найменший час. Тест цікавий тим, що у ньому використовуються як HDFS, так і MapReduce, які ми аналізували у архітектурній частині.

3.2.5 Розв’язок судоку

Багато людей люблять судоку, існує багато алгоритмів його рішення, чому б не спробовати його розвязати за допомогою Hadoop. Будемо використовувати алгоритм “Dancing links”

3.3 Оцінка сильних та слабких сторін Apache Hadoop

Для того щоб мати змогу об’єктивно оцінити результати, зробимо задачі обчислювально складними, з великою кількістю даних. Основним критерієм за яким буде оцінена робота системи – це час. Буде розроблене програмне забезпечення на мові програмування Java.

Для того щоб зовнішні фактори не впливали на чистоту тесту, усі задачі будуть виконуватися в однакових умовах, а вимір часу буде проводитися за допомогою вбудованого інструментарію фреймворку та системної команди Linux time.

Результатом порівняння буде 5 таблиць для кожного з тестових прикладів де будуть записані часи розрахунків для різної кількості даних.
Для більшої точності кожний тест виконувався 5 разів, а результат рахувався як середнє арифметичне серед них. У результаті треба дивится на слабкі сторони Hadoop, вплив яких треба буде мінімізувати у майбутньому.

3.4 Порівняльна таблиця

Таблиця 3.1 – Генерація текстових файлів

Розмір файлу	5 Мб	500Мб	10 Гб
Час виконання	1хв. 32с.	8хв. 54с.	123хв. 8с.

Таблиця 3.2 – Підрахунок кількості заданого слова(різні слова)

Розмір файлу	5 Мб	500Мб	10 Гб
Час виконання	27с	31с	51с

Таблиця 3.3 – Знаходження максимальної температури

Розмір файлу	5 Мб	500Мб	10 Гб
Час виконання	38с	1хв 22с	3хв 41с

Таблиця 3.4 – Terasoft тест

Розмір файлу	10 Tb
Час виконання	~6 год

Таблиця 3.5 – Розв’язок судоко

Розмір файлу	9x9	25x25	42x42
Час виконання	0.224с	0.291с	0.341с

3.5 Графіки порівнянь

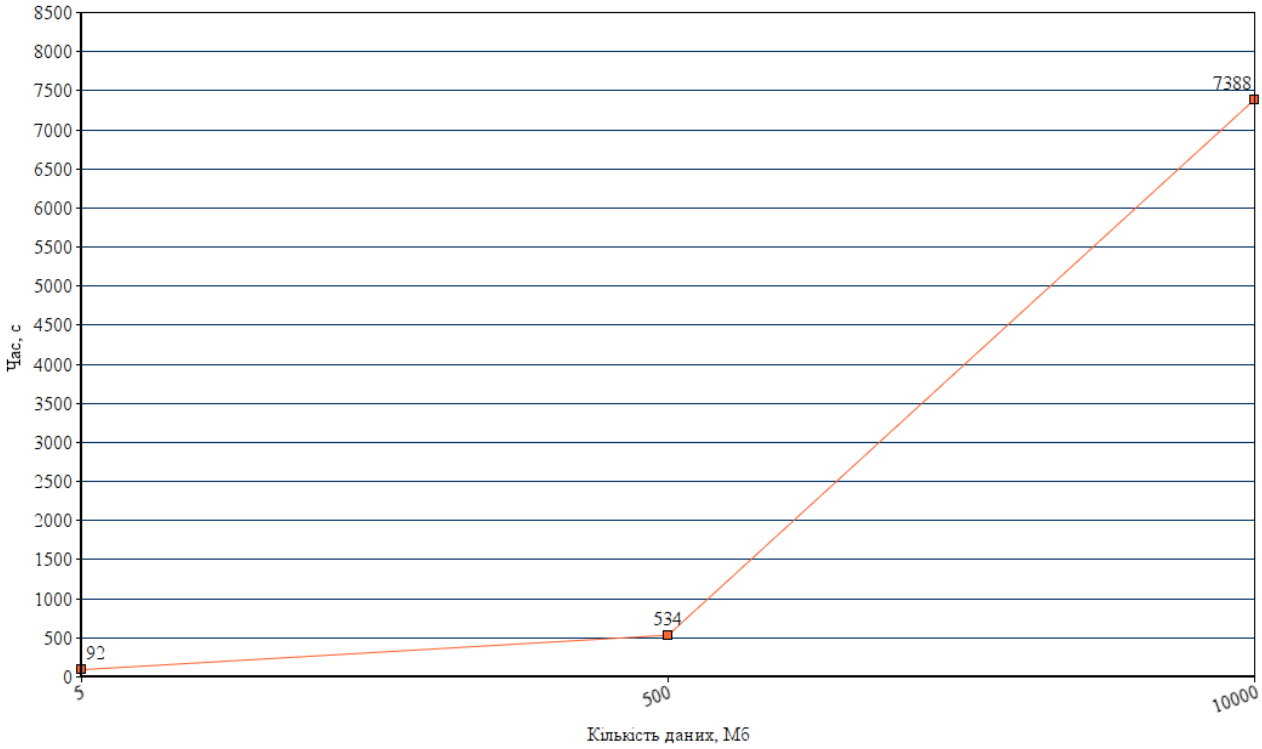

Рисунок 3.1 – Генерація текстових файлів

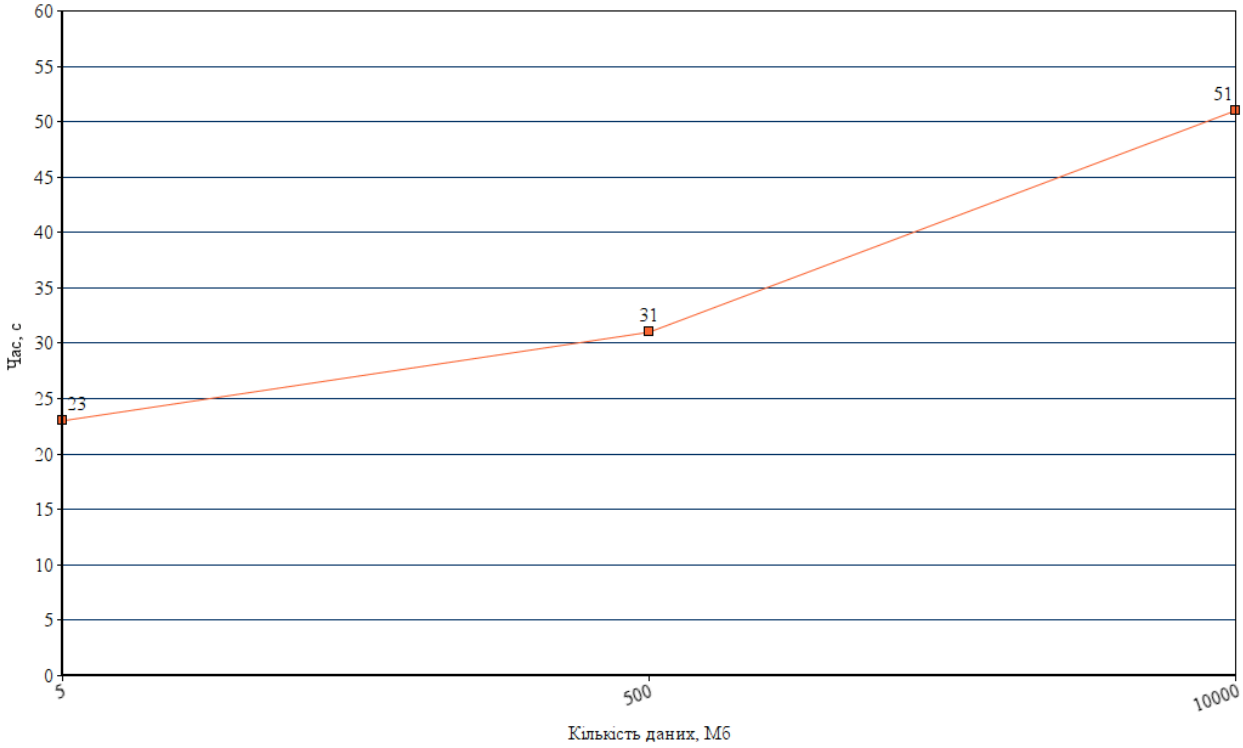

Рисунок 3.2 – Підрахунок кількості заданого слова(різні слова)

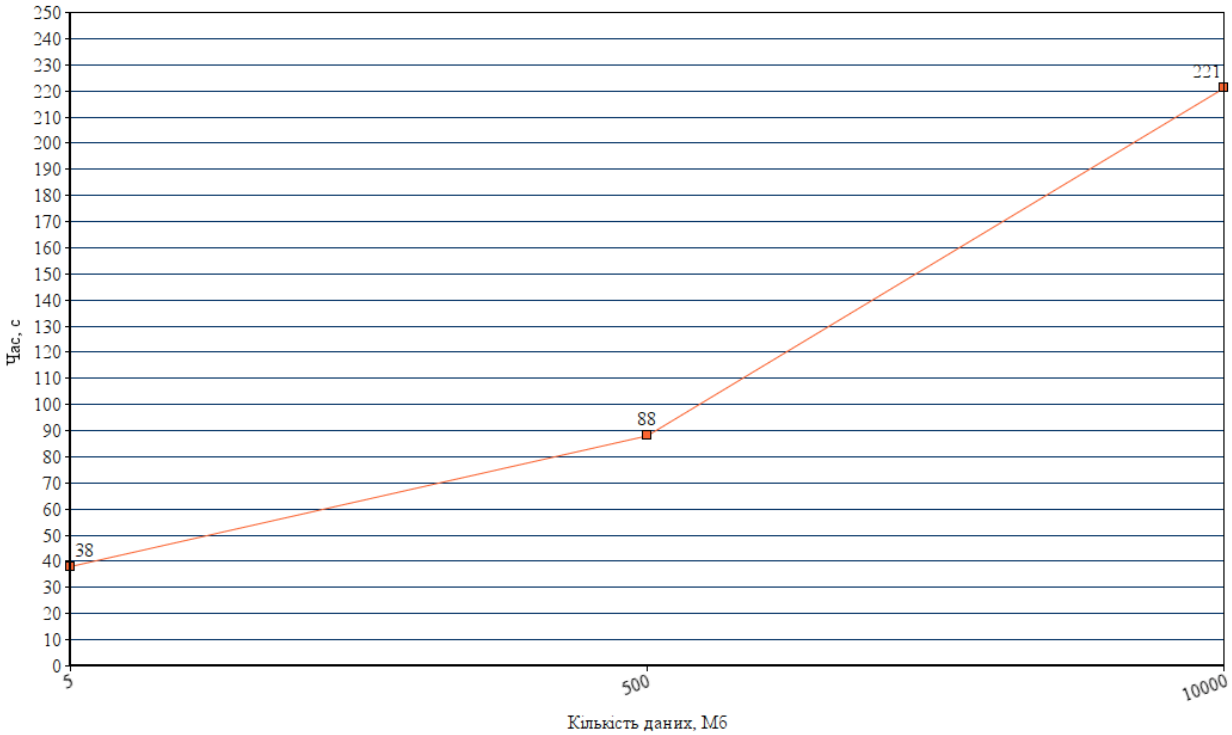

Рисунок 3.3 – Знаходження максимальної температури

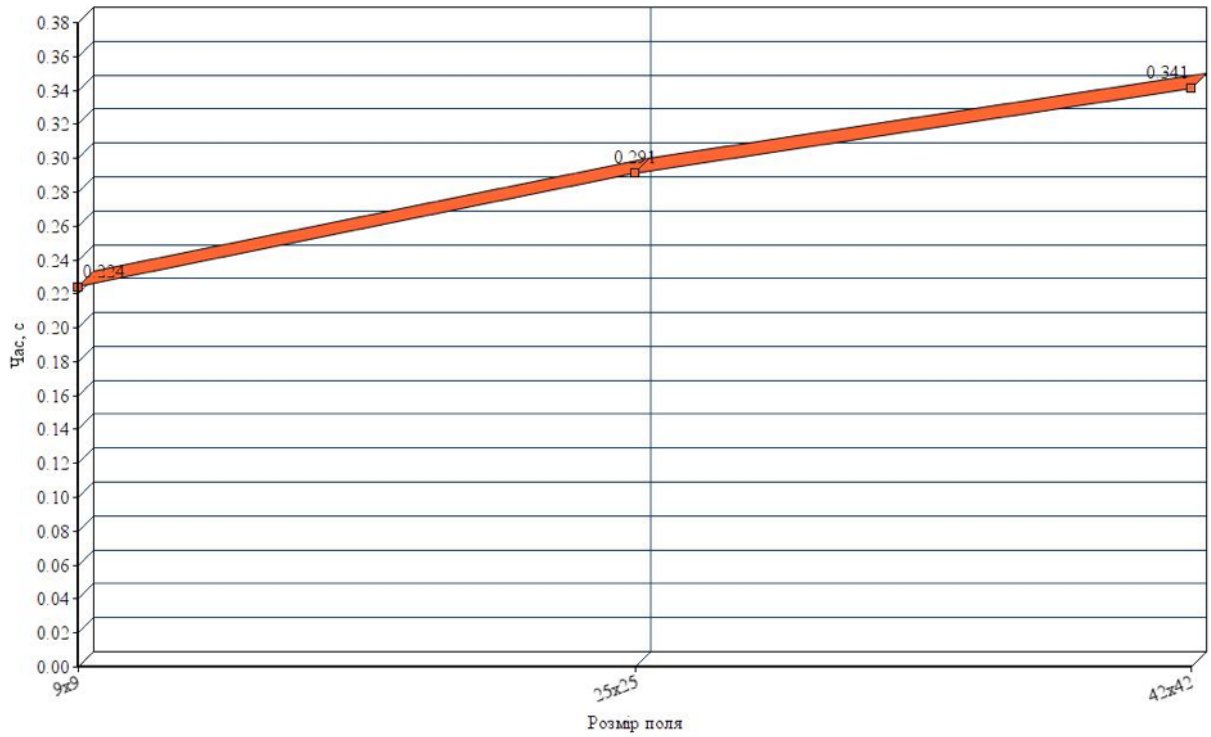

Рисунок 3.4 – Розв'язок судоку

3.6 Висновки

У даному розділі проводиться аналіз отриманих результатів, а саме поведінка фреймворку HADOOP при вирішені різних задач.

Виявилося, що досліджувана платформа дуже добре поводить себе в середовищі великих обчислень та задач із об’ємними вхідними або вихідними даними. Вузьким місцем можна вважати – запис у файл.

4. ЕКОНОМІЧНО-ОРГАНІЗАЦІЙНА ЧАСТИНА

4.1 Вступ

В даному розділі проводиться аналіз варіантів реалізації модулю з метою вибору оптимальної, з економічної точки зору. А саме проводиться функціонально-вартісний аналіз (ФВА).

Функціонально-вартісний аналіз — це метод комплексного техніко-економічного дослідження об’єкта з метою розвитку його корисних функцій при оптимальному співвідношенні між їхньою значимістю для споживача і витратами на їхнє здійснення. Є одним з основних методів оцінки вартості науково-дослідної роботи, оскільки ФВА враховує як технічну оцінку продукту, що розробляється, так і економічну частину розробки. Крім того, даний метод дозволяє вибрати оптимальний варіант розв’язання задачі, як з погляду розробника, так і з точки зору покупця. Також він дозволяє оптимізувати витрати й час виконання робіт.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

– визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

– для кожної функції визначаються повні річні витрати й кількість робочих часів.

– для кожної функції наоснові оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

– після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

У даній роботі проводиться оцінка основних характеристик програмного продукту призначеного для розпізнавання звукових образів та голосу людини .

4.2 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

– програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;

– забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;

– забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;

– передбачати мінімальні витрати на впровадження програмного продукту.

4.3 Обґрунтування функцій програмного продукту

Головна функція F_0– розробка програмного продукту, який аналізує процес за вхідними даними та будує його модель для подальшого прогнозування. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – вибір оптимальної СКБД;

F_3 – інтерфейс користувача.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування C#, .NET 4.5.

б) мова програмування Python;

в) мова програмування Java.

Функція F_2:

а) MS SQL;

б) Oracle;

в) MySQL.

Функція F_3:

а) інтерфейс користувача, створений за технологією WPF;

б) інтерфейс користувача, створений за технологією PyQt;

в) інтерфейс користувача, створений за технологією JavaFX.

4.4 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

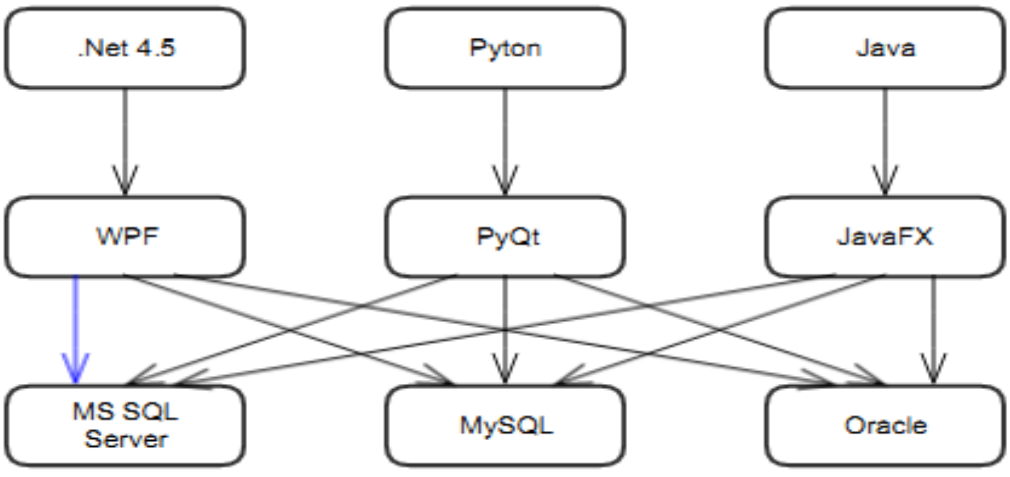

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Займає менше часу при написанні коду	Не кросплатформений
	Б	Безкоштовний	Погана обробка великих об’ємів даних
	В	Кросплатформений	Низька швидкодія
F2	А	Більш дешева вартість корпоративної ліцензії	Необхідність додаткової інсталяції, низький рівень користувацької підтримки
	Б	Подовжений термін користувацької підтримки	Більш висока вартість корпоративної ліцензії. Необхідність додаткової інсталяції
	В	Простота, багатий функціонал, високий рівень безпеки	Обмежений фунціонал
F3	*А*	Гнучкість интерфейсу, відокремлення бізнес-логіки від користувацького інтерфейсу	Відсутність кросплатформеності
	Б	Кросплатформеність	Не включається із встановленням Python
	В	Простота створення	Відсутність кросплатформеності

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 *F1:*

Оскільки задача вимагає написання та підтримання великої кількості коду необхідно вибрати максимально зручну IDE, для мінімізації часу розробки ПО. Також важливим фактором є швидкість виконання програми та зручний користувацький інтерфейс. Враховуючи ці умови найбільш прийнятним варіантом для побудови інтерфейсної частини є Java FX. А для написання алгоритмів обробки використовуватимемо мову програмування Java. Така комбінація дозволить створити добре розширюваний проект з незалежною серверною частиною та користувацьким інтерфейсом.

Функція *F2:*

Вибір СКБД не відіграє великої ролі для даного програмного продукту, тому вважаємо варіанти а), б) та в) рівноможливими.

Функція *F3:*

Оскільки, програмний продукт реалізується з використанням JAVA використовуємо варіант В (переваги цього варіанту описанні в описі F1).

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1в – F2а – F3в
2. F1в – F2б – F3в
3. F1в – F2в – F3в

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

4.5 Обґрунтування системи параметрів ПП

4.5.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

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

– *X1* – швидкодія мови програмування;

– *X2* – об’єм пам’яті для збереження даних;

– *X3* – час обробки даних;

– *X4* – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

4.5.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.1.

Таблиця 4.2 – Основні параметри ПП

Назва Параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**Кращі**
Швидкодія мови програмування	X1	Оп/мс	18000	10000	3000
Об’єм пам’яті для збереження даних	X2	Мб	8128	4096	2048
Час обробки запитів користувача	X3	с	60	20	5
Потенційний об’єм програмного коду	X4	кількість строк коду	15000	10000	7500

За даними таблиці 4.1 будуються графічні характеристики параметрів – рис. 4.1 – рис. 4.4

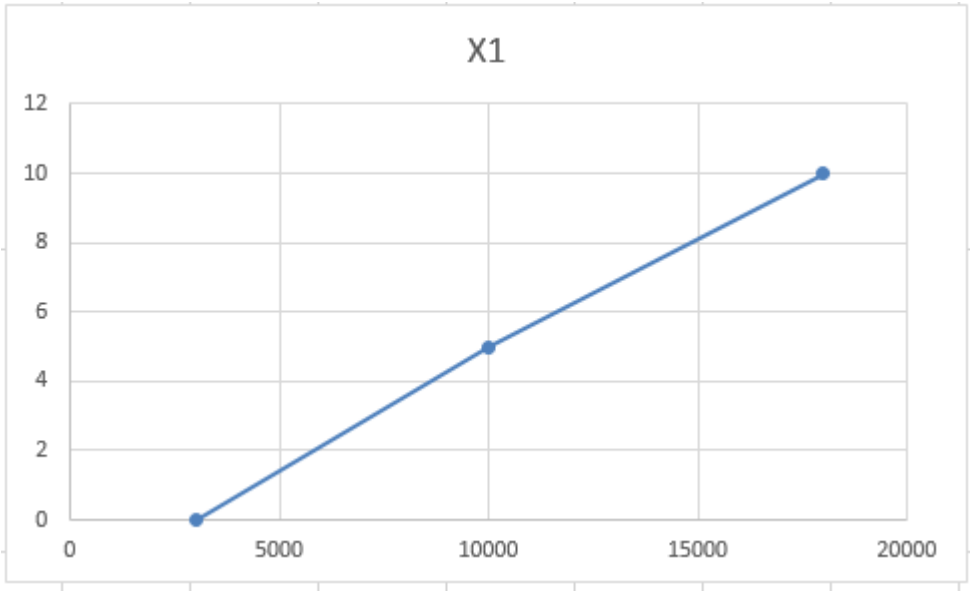

Рисунок 4.1 – швидкодія мови програмування

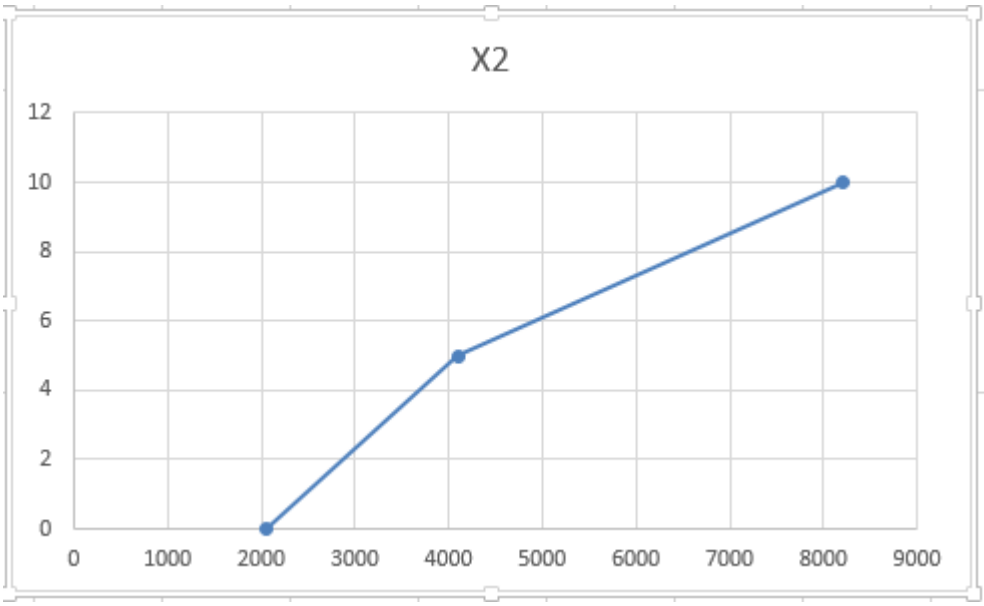

Рисунок 4.2 – об’єм пам’яті для збереження даних

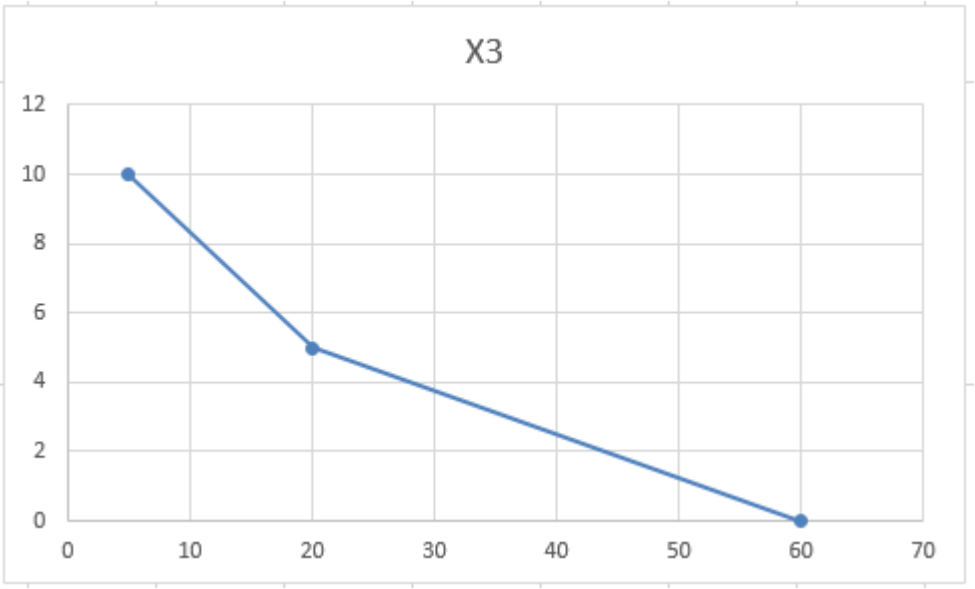

Рисунок 4.3 – час виконання запитів користувача

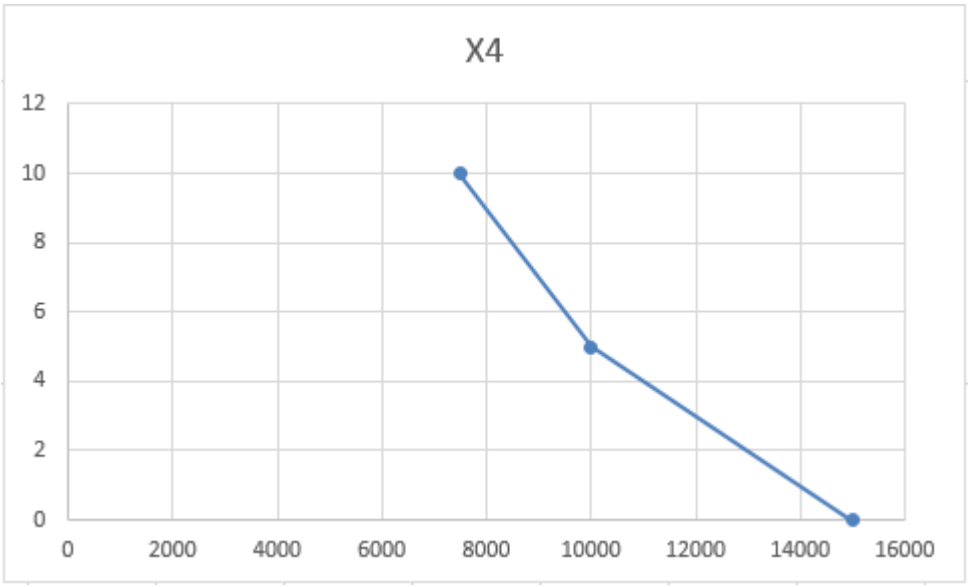

Рисунок 4.4 – потенційний об’єм програмного коду

4.5.3 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 5 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 4.2

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта					Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5			
X1	Швидкодія мови програмування	Оп/мс	4	3	4	4	4	19	1,50	2,25
X2	Об’єм пам’яті для збереження даних	Мб	4	4	4	3	4	19	1,50	2,25
X3	Час обробки запитів користувача	Мс	2	2	1	2	1	8	- 9,50	90,25
X4	Потенційний об’єм програмного коду	кількість строк коду	4	5	5	5	5	24	6,50	42,25
	Разом		14	14	14	14	14	70	0	420,75

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

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

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 137.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 137}{5^2(4^3 - 4)} = 1,096 > W_k = 0,67$$

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 4.3

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти					Кінцева оцінка	Числове значення
	1	2	3	4	5		
X1 і X2	=	>	=	<	=	<	0,5
X1 і X3	<	<	<	<	<	<	0,5
X1 і X4	>	>	>	>	>	>	1,5
X2 і X3	<	<	<	<	<	<	0,5
X2 і X4	>	>	>	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}. \quad (4.1)$$

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 наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j. \quad (4.2)$$

Як видно з таблиці 4.4, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1.0	0.5	0.5	1.5	3.5	0.219	22.25	0.216	100	0.215
X2	1.5	1.0	0.5	1.5	4.5	0.281	27.25	0.282	124.25	0.283
X3	1.5	1.5	1.0	1.5	5.5	0.344	34.25	0.347	156	0.348
X4	0.5	0.5	0.5	1.0	2.5	0.156	14.25	0.155	64.75	0.154
Всього:					16	1	98	1	445	1

4.6 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2*(об'єм пам'яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 60 с або варіанту в) 40 с.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.3):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\ , \qquad (4.3)$$

де n – кількість параметрів;$K_{вi}$– коефіцієнт вагомості i–го параметра;B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	10000	3,6	0,215	0,781
F2(X2)	А	4096	3,4	0,283	0,962
F3(X3,X4)	А	60	2,4	0,348	0,835
	Б	40	1	0,154	0,154

За даними з таблиці 4.3 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \qquad (4.4)$$

визначаємо рівень якості кожного з варіантів:

$K_{K1} = 0{,}781 + 0{,}962 + 0{,}835 = 2{,}64$

$K_{K2} = 0{,}781 + 0{,}962 + 0{,}154 = 1{,}96$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.7 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. 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.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.5)$$

де T_P – трудомісткість розробки ПП;$K_П$ – поправочний коефіцієнт;$K_{СК}$ – коефіцієнт на складність вхідної інформації; K_M – 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 =95 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$

= 0.8. Тоді, за формулою 4.1, загальна трудомісткість програмування першого завдання дорівнює:

T_1 = 95*1.7*0.8 = 129.2 людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =30 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

T_2 = 30 * 0.9 ·*0.8 = 21,6 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

T_I = (129.2 + 21.6 + 4.8 + 21.6) · 8 = 1417.6 людино-годин;

T_{II} = (129.2 + 21.6 + 6.91 + 21.6) · 8 = 1434.48 людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 17000 грн. Визначимо зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.}, \quad (4.6)$$

де М – місячний оклад працівників;T_m – кількість робочих днів тиждень;t – кількість робочих годин в день.

$$С_Ч = \frac{17000 + 17000}{3 \cdot 23 \cdot 8} = 67{,}46 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$$С_{ЗП} = С_Ч \cdot T_i \cdot К_Д,$$

де $С_Ч$– величина погодинної оплати праці програміста;T_i – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП}$ = 67,46· 1417,6 · 1,2 = 92540,93 грн.

II. $С_{ЗП}$ = 67,46· 1434,48 · 1,2 = 93642,85 грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0{,}3677$ = 92540,93 *0,22 = 34027,30 грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0,3677 = 93642,85 * 0,22 = 34432,48$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Оскільки одна ЕОМ обслуговує одного програміста з окладом 17000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 17000 \cdot 0,2 = 40800$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 40800 \cdot (1 + 0,2) = 48960$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0,22 = 48960 * 0,22 = 10771.2$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 30000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 30000 = 8625$ грн.,

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 як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 30000 \cdot 0.04 = 1380$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 12) \cdot 8 \cdot 0.9 = 1735.2$ годин,

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:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1735,2 \cdot 0,162 \cdot 1,385 \cdot 2.0218 = 787,14$ грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н$ = $Ц_{ПР}$·0.67 = 30000· 0,67 =20100 грн.

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н \quad (4.7)$$

$С_{ЕКС}$ = 48960 + 9792 + 8625+1380+787,14+20100 =90623,34 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ}$ = 90623,34 /1735,2 = 52,22 грн/год.

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, складає:

$$С_М = С_{М-Г} \cdot Т \quad (4.8)$$

I. $С_М$ = 51,66 * 1417,6 = 74027,07 грн.;

II. $С_М$ = 51,66 * 1434,48 = 74908,55 грн.;

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67$$

I. $С_Н$ = 92540,93 * 0,67 = 62004,42 грн.;

II. $С_Н$ = 93642,85 * 0,67 = 62740,70 грн.;

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н \quad (4.9)$$

I. $С_{ПП}$ = 92540,93 + 34027,30 + 73233,22 + 62004,42 = 262599,72 грн.;

II. $С_{ПП}$ = 93642,85 + 34432,48 + 74105,24 + 62740,70 = 265724,48 грн.;

4.8 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj}, \quad (4.10)$$

$К_{ТЕР1} = 2,64 / 262599,72 = 0,952 \cdot 10^{-5}$;

$К_{ТЕР2} = 1,96 / 245699,23 = 0,713 \cdot 10^{-5}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 0,952 \cdot 10^{-5}$.

4.9 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

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 $K_{ТЕР} = 0{,}978 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – Java;

– СКБД MS SQL Server;

– інтерфейс користувача, створений за технологією Java FX.

Даний варіант виконання програмного комплексу дає зручний користувацький інтерфейс, найкращі показники надійності програмного продукту, функціонал, що задовольняє заданим вимогам та непогану швидкодію.

ВИСНОВКИ

Дипломна робота присвячена актуальній задачі обробки великих об’ємів даних із застосуванням платформи Apache Hadoop.

У результаті виконання даної дипломної роботи було розроблено програмний продукт, за допомогою якого можна дослідити роботу системи розподілених обчислень HADOOP на різних тестових прикладах.

Виявилося, що фреймворк у цілому справляється з поставленою задачею, але є такі випадки або такі задачі де використання більш простої системи буде доцільнішим. Також, є алгоритми де Apache Hadoop у зв’язку із специфічним алгоритмом роботи платформи на рівень ефективніше виконує програму.

У першому розділі була досліджена задача розподілення обчислень великих даних, яка використовувалася для вирішення заданої проблеми, описано про особливості та проблематики задачі розподілення обчислень. Було сформульовано постановку і актуальність даної задачі.

Другий розділ було присвячено дослідженню існуючих методів для розв’язання проблеми, побудовано архітектуру, описана конфігурація системи. Проаналізовано основні компоненти Apache Hadoop, які використовувалися у роботі.

У третьому розділі проводився аналіз отриманих результатів, а саме порівняння продуктивності Apache Hadoop при вирішені різних задач. Найкраще платформа справляється з сортуванням, та пошуком, завдяки внутрішним компонентам: MapReduce та HDFS. Досліди проводилися на різній кількості обсягу даних. Платформа Apache Hadoop покращувала свої результати з ростом об’ємів даних. Також, було помічено, що вузьким місцем є запис даних до файлу. Отже, можна зробити висновки, що платформа призначена дійсно для великої кількості інформації.

У четвертому розділі було проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Даний варіант виконання програмного комплексу дає зручний користувацький

інтерфейс, найкращі показники надійності програмного продукту, функціонал, що задовольняє заданим вимогам та непогану швидкодію.

Використання Hadoop і MapReduce для побудови системи обробки великих даних дозволило забезпечити автоматичне розпаралелювання і зберігання даних на внутрішніх дисках вузлів кластера. Запропонована архітектура приховує від прикладного програміста деталі внутрішнього устрою Hadoop і надає простий програмний інтерфейс, який дозволяє сконцентруватися на алгоритмах обробки даних, а не на організації паралельної обробки. Розподілена файлова система дозволяє зберігати дані великого обсягу на дисках серверів стандартної архітектури без необхідності установки дорогої системи зберігання.

Актуальність роботи полягає у тому, що розподіл обчислень зменшує вартість та час, необхідні для виконання завдання. Шляхи подальшого розвитку предмета дослідження - використання більшої кількості більш потужних систем для складних обчислень. Поглибитись у взаємодію Apache Hadoop з СУБД, знайти

ПЕРЕЛІК ПОСИЛАНЬ

1. Сайт української команди розподілених обчислень. — Режим доступа : http://distributed.org.ua/index.php?newlang=ua. — Дата доступа : 25.04.15.
2. Семчишин Ю. Б.Методи та засоби розподілення обчислень в задачах теплового проектування електронних пристроїв. / Семчишин Ю. Б. - Львів : Львівська політехніка, 2010. - 24 с.
3. Розподілені обчислення в Україні. — Режим доступа : http://kirdey.com/rozpodileni-obchislennya-v-ukrayini — Дата доступа : 25.04.15.
4. Holger J. PletschDeepest All-Sky Surveys for Continuous Gravitational Waves. / Holger J. Pletsch. –Hannover. : Leibniz Universität Hannover, July 25, 2010.
5. Офіційний сайт PrimeGrid. — Режим доступа : http://www.primegrid.com. — Дата доступа : 26.04.15.
6. Prime Number. — Режим доступа : http://mathworld.wolfram.com/PrimeNumber.html. — Дата доступа : 26.04.15.
7. Séroul R. "The Sieve of Eratosthenes." /Séroul, R. // Programming for Mathematicians. - Berlin: Springer-Verlag, 2000, pp. 169-175.
8. Arnault F. "Rabin-Miller Primality Test: Composite Numbers Which Pass It." / Arnault F. // Math. Comput. - New York,1995 355-361.
9. Richard Crandall and Carl Pomerance. Prime Numbers: A Computational Perspective. / Richard Crandall and Carl Pomerance. – Springer : 2005. p.159–190.
10. Richard Crandall and Carl Pomerance. Prime Numbers: A Computational Perspective. / Richard Crandall and Carl Pomerance. – Springer : 2005. p.334–340.
11. ThePrimeGlossary — Режим доступа : http://primes.utm.edu/glossary/home.php. — Дата доступа : 02.05.15.

12. Arora Sanjeev and Barak Boaz. Computational Complexity – A Modern Approach. / Arora Sanjeev and Barak Boaz. – Cambridge: 2009.
13. Andrews, Gregory R. Foundations of Multithreaded, Parallel, and Distributed Programming. / Andrews, Gregory R. – Addison–Wesley: 2000.
14. Scott Guthrie: Silverlight and the Cross-Platform — Режим доступа: https://channel9.msdn.com/shows/Going+Deep/Scott-Guthrie-Silverlight-and-the-Cross-Platform-CLR — Дата доступа : 06.05.15.
15. Preimesberger, Chris Hadoop, Yahoo, 'Big Data' Brighten BI Future (англ.). – EWeek: 2011
16. Tryon R.C. Cluster analysis. — London: Ann Arbor Edwards Bros: 2014

ДОДАТОК А

ТЕКСТ ПРОГРАМИ

```
package org.apache.hadoop.examples;
import java.io.IOException;
import java.net.URI;
import java.util.*;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.ClusterStatus;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.mapreduce.lib.partition.InputSampler;
import org.apache.hadoop.mapreduce.lib.partition.TotalOrderPartitioner;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

```
public class Sort<K,V> extends Configured implements Tool {
  public static final String REDUCES_PER_HOST =
    "mapreduce.sort.reducesperhost";
  private Job job = null;
  static int printUsage() {
    System.out.println("sort [-r <reduces>] " +
                       "[-inFormat <input format class>] " +
                       "[-outFormat <output format class>] " +
                       "[-outKey <output key class>] " +
                       "[-outValue <output value class>] " +
                       "[-totalOrder <pcnt> <num samples> <max splits>] " +
                       "<input> <output>");
    ToolRunner.printGenericCommandUsage(System.out);
    return 2;
  }

public int run(String[] args) throws Exception {

    Configuration conf = getConf();
    JobClient client = new JobClient(conf);
    ClusterStatus cluster = client.getClusterStatus();
    int num_reduces = (int) (cluster.getMaxReduceTasks() * 0.9);
    String sort_reduces = conf.get(REDUCES_PER_HOST);
    if (sort_reduces != null) {
       num_reduces = cluster.getTaskTrackers() *
                       Integer.parseInt(sort_reduces);
    }
    Class<? extends InputFormat> inputFormatClass =
      SequenceFileInputFormat.class;
    Class<? extends OutputFormat> outputFormatClass =
```

```
  SequenceFileOutputFormat.class;
Class<? extends WritableComparable> outputKeyClass = BytesWritable.class;
Class<? extends Writable> outputValueClass = BytesWritable.class;
List<String> otherArgs = new ArrayList<String>();
InputSampler.Sampler<K,V> sampler = null;
for(int i=0; i < args.length; ++i) {
  try {
    if ("-r".equals(args[i])) {
      num_reduces = Integer.parseInt(args[++i]);
    } else if ("-inFormat".equals(args[i])) {
      inputFormatClass =
        Class.forName(args[++i]).asSubclass(InputFormat.class);
    } else if ("-outFormat".equals(args[i])) {
      outputFormatClass =
        Class.forName(args[++i]).asSubclass(OutputFormat.class);
    } else if ("-outKey".equals(args[i])) {
      outputKeyClass =
        Class.forName(args[++i]).asSubclass(WritableComparable.class);
    } else if ("-outValue".equals(args[i])) {
      outputValueClass =
        Class.forName(args[++i]).asSubclass(Writable.class);
    } else if ("-totalOrder".equals(args[i])) {
      double pcnt = Double.parseDouble(args[++i]);
      int numSamples = Integer.parseInt(args[++i]);
      int maxSplits = Integer.parseInt(args[++i]);
      if (0 >= maxSplits) maxSplits = Integer.MAX_VALUE;
      sampler =
        new InputSampler.RandomSampler<K,V>(pcnt, numSamples, maxSplits);
    } else {
      otherArgs.add(args[i]);
```

```
    }
  } catch (NumberFormatException except) {
    System.out.println("ERROR: Integer expected instead of " + args[i]);
    return printUsage();
  } catch (ArrayIndexOutOfBoundsException except) {
    System.out.println("ERROR: Required parameter missing from " +
        args[i-1]);
    return printUsage(); // exits
  }
}
// Set user-supplied (possibly default) job configs
job = Job.getInstance(conf);
job.setJobName("sorter");
job.setJarByClass(Sort.class);

job.setMapperClass(Mapper.class);
job.setReducerClass(Reducer.class);

job.setNumReduceTasks(num_reduces);

job.setInputFormatClass(inputFormatClass);
job.setOutputFormatClass(outputFormatClass);

job.setOutputKeyClass(outputKeyClass);
job.setOutputValueClass(outputValueClass);

// Make sure there are exactly 2 parameters left.
if (otherArgs.size() != 2) {
  System.out.println("ERROR: Wrong number of parameters: " +
      otherArgs.size() + " instead of 2.");
```

```
  return printUsage();
}
FileInputFormat.setInputPaths(job, otherArgs.get(0));
FileOutputFormat.setOutputPath(job, new Path(otherArgs.get(1)));

if (sampler != null) {
  System.out.println("Sampling input to effect total-order sort...");
  job.setPartitionerClass(TotalOrderPartitioner.class);
  Path inputDir = FileInputFormat.getInputPaths(job)[0];
  FileSystem fs = inputDir.getFileSystem(conf);
  inputDir = inputDir.makeQualified(fs.getUri(), fs.getWorkingDirectory());
  Path partitionFile = new Path(inputDir, "_sortPartitioning");
  TotalOrderPartitioner.setPartitionFile(conf, partitionFile);
  InputSampler.<K,V>writePartitionFile(job, sampler);
  URI partitionUri = new URI(partitionFile.toString() +
                             "#" + "_sortPartitioning");
  job.addCacheFile(partitionUri);
}

System.out.println("Running on " +
    cluster.getTaskTrackers() +
    " nodes to sort from " +
    FileInputFormat.getInputPaths(job)[0] + " into " +
    FileOutputFormat.getOutputPath(job) +
    " with " + num_reduces + " reduces.");
Date startTime = new Date();
System.out.println("Job started: " + startTime);
int ret = job.waitForCompletion(true) ? 0 : 1;
Date end_time = new Date();
System.out.println("Job ended: " + end_time);
```

```
    System.out.println("The job took " +
        (end_time.getTime() - startTime.getTime()) /1000 + " seconds.");
    return ret;
  }
  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new Sort(), args);
    System.exit(res);
  }
  public Job getResult() {
    return job;
  }
}
```