Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра _____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

_____другий (магістерський)_____
(рівень вищої освіти)

Дослідження моделі розподіленої обробки даних для обробки великих обсягів даних на комп'ютерних кластерах (MapReduce парадигма)
(тема)

Виконав: студент 2 курсу, групи ПЗСм-18-1___
спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)
Освітньо-наукової програми
_____Програмне забезпечення систем_____
(повна назва освітньої програми)

_____Чайковський В.Р._____
(прізвище, ініціали)
Керівник _____доц. Ревенчук І.А._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2019 р.

Харківський національний університет радіоелектроніки

Факультет Комп‘ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121- Інженерія програмного забезпечення

Тип програми освітньо-професійна програма

Освітня програма Програмне забезпечення систем

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«___»_____ 2019 р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові __Чайковському Владиславу Руслановичу_____

1. Тема роботи проекту «Дослідження моделі розподіленої обробки даних для обробки великих обсягів даних на комп'ютерних кластерах (MapReduce парадигма)» затверджена наказом по університету від «_ __»_ __ _2019р. № _____
2. Термін подання студентом роботи (проекту): _____ 2019р.
3. Вихідні дані до роботи: алгоритми роботи із Big Data. Використовувати ОС Linux, JVM.
4. Перелік питань, що потрібно опрацювати в роботі: мета роботи, аналіз проблемної галузі і постановка задачі, аналіз алгоритмів для роботи із великими даними, математичний аналіз роботи алгоритмів для обробки великих даних.

5. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Аналіз робемної області	доц. Ревенчук І.А.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів проекту (роботи)	Примітка
1.	Аналіз предметної галузі		
2.	Огляд існуючих методів		
3.	Методи роботи із Big Data		
4.	Аналіз методів роботи із Big Data		
5.	Аналіз алгоритмів роботи із Big Data		
6.	Підготовка презентації та доповіді		
7.	Попередній захист		
8.	Нормоконтроль, рецензування		
9.	Занесення диплома в електронний архів		
10.	Допуск до захисту у зав. кафедри		

Дата видачі завдання: «____»_____ 2019 р.

Студент _____ Чайковський В.Р.
(підпис)

Керівник роботи _____ доц. Ревенчук І.А.
(підпис)

РЕФЕРАТ/ABSTRACT

Атестаційна робота магістра містить: 71 с., 38 рис., 4 додатки, 18 джерел.

АЛГОРИТМ, АНАЛІЗ, MAPREDUCE, HIVE, BIG DATA, HADOOP, HDFS.

Метою роботи є дослідження моделі розподіленої обробки даних для обробки великих обсягів даних на комп'ютерних кластерах(MapReduce парадигма), аналіз технологій роботи із Big Data.

Методом вирішення є детальний аналіз технологій роботи із Big Data, пошук переваг та недоліків парадигми MapReduce, порівняння данної парадигми із іншими технологіями роботи із великими даними.

Результатом роботи є данні переваг та недоліків парадигми MapReduce, порівняння цієї технології з іншими, а також варіанти та ідеї вирішення тих чи інших проблем пов’язаних із даною технологією.

ALGORITHM, ANALYSIS, MAPREDUCE, HIVE, BIG DATA, HADOOP, HDFS.

The purpose of the study is to investigate a distributed data processing model for processing large volumes of data on computer clusters (MapReduce paradigm), to analyze Big Data technologies.

The solution method is a detailed analysis of Big Data technologies, finding the advantages and disadvantages of the MapReduce paradigm.

The result is an outline of the advantages and disadvantages of the MapReduce paradigm, a comparison of this technology with others, as well as options and ideas for solving particular problems associated with this technology.

ЗМІСТ

ВСТУП

Великі дані – це концепція великого спектру даних, яка створюється день у день. В останні роки обробка цих даних є найбільшою проблемою. Дві основні концепції hadoop – це розподілена файлова система (HDFS) Mapreduce та Hadoop. HDFS – це механізм зберігання, а mapreduce – мова програмування. Результати оброблюються швидше, ніж інші традиційні операції з базою даних. Pig та Hive – це дві мови, які допомагають нам програмувати mapreduce framework за короткий проміжок часу.

BigData містить структуровані та неструктуровані дані. Структуровані дані складаються з даних у текстовому та табличному форматі. Завдяки цьому їх можна легко структурувати та обробити за допомогою інструменту майнінгу даних. Неструктуровані дані не мають ідентифікованої внутрішньої структури, тому обробка цих даних с традиційної бази даних неможливі.

Обробка даних є найбільшою проблемою у BigData оскільки воно містить як типи даних, так і обчислення яке не може бути виконано звичною базою даних та технологією майнінгу даних. Дослідження стверджує, що вміст BigData генерується кожний день. IBM зазначає, що 2.5 мільярд гігабайт даних виробляються за один день[1].

BigData має кілька характеристик. Різноманітність посилається до різних форматів даних. Наприклад, розглянемо: банківську операцію, при цьому тут різноманітністю є чек, банкомат, платіжна картка тощо. Velocity означає швидкість виробництва даних з різної техніки, сенсорів, файлів журналу тощо. Складність в цьому є правильність поводження з великим обсягом даних.

Таким чином була поставлена задача провести дослідження існуючих способів роботи із великими даними, проаналізувати технології роботи із BigData, виявити переваги та недоліки парадигми MapReduce.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної області

Методику і інструменти роботи зі структурованими даними IT-індустрія створила давно – це реляційна модель даних і системи управління БД. Але сучасною тенденцією є потреба обробки великого обсягу неструктурованих даних, і це та область, де старі підходи працюють погано або взагалі не працюють. Саме ця потреба вимагає нової методики поводження з даними, і зараз все більш популярною стає модель роботи з Big Data. Займаючись проблемами Big Data, перед розробниками і вченими стоїть завдання – знайти програмне і технічне рішення, здатне легко інтегруватися в існуючу інфраструктуру ЦОД і забезпечити всі три етапи обробки інформації: збір, її організацію і аналіз.

В якості характеристик для великих даних Forrester визначає поняття Big Data як технологію в області апаратного та програмного забезпечення, яка об'єднує, організує, управляє і аналізує дані, які характеризуються «чотирма V»: об'ємом (Volume), різноманітністю (Variety), мінливістю (Variability) і швидкістю (Velocity). Ці характеристики є суттєвими проблемами технології Big Data. Розглянемо кожну з цих складових[2].

Швидкість надходження і обробки інформації (velocity).В області Big Data виділяють проблему: недостатньо висока швидкість обробки даних. В ряді завдань ця швидкість повинна бути дуже високою. Наприклад, біржовим гравцям іноді потрібно миттєво прийняти рішення, ґрунтуючись на великій кількості даних про стан ринку - за пару секунд ситуація вже може змінитися. Дуже велика швидкість надходження даних характерна для багатьох наукових завдань. Наприклад, тільки один експериментальний синхротрон Advanced Photon Source (APS) в Аргоннской лабораторії, який використовується в числі іншого для томографической зйомки об'єктів на субмікронних дозволах, може щодня генерувати 150 Тбайт інформації.

Обсяг даних (Volume). Проаналізувавши інформацію була створена діаграма на якій зображено які обсяги даних накопичили корпорації. Тільки в США це понад 100 Тбайт даних. При цьому в різних вертикальних індустріях обсяг даних істотно розрізняється, отже, актуальність застосування технології Big Data в них різна (рисунок 1).

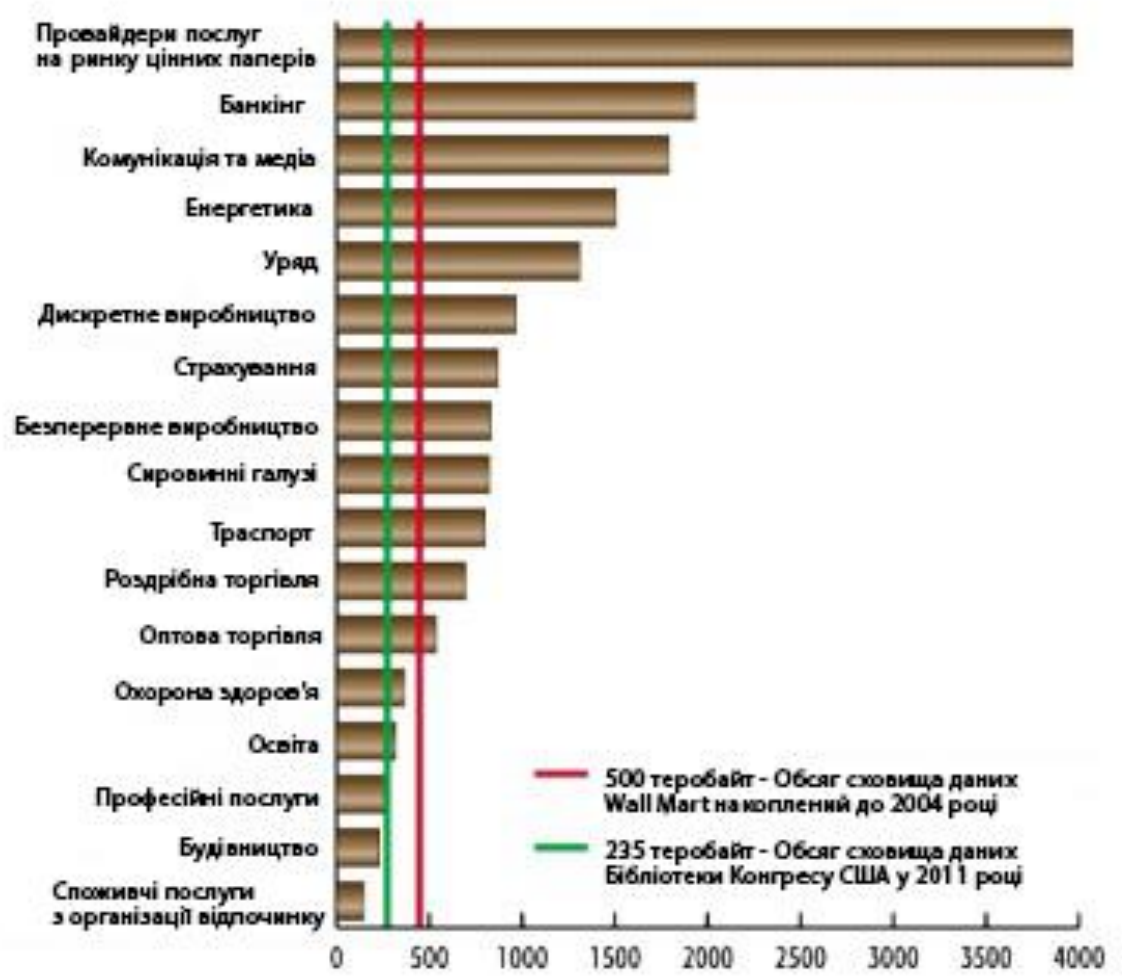

Рисунок 1 – Обсяг накопичених даних в корпораціях з різних сфер діяльності

Різноманітність форматів даних (Variety). Здатність додатки обробляти великі масиви даних, що надходять з різних джерел в різних форматах, є головним критерієм віднесення його до технології Big Data. Багато бізнес-завдання і наукові експерименти вимагають спільної обробки даних різних форматів - це можуть бути табличні дані в

СУБД, ієрархічні дані, текстові документи, відео, зображення, аудіофайли[3]. Приклад подібного роду завдання з області медицини: як знайти оптимальний курс лікування для конкретного пацієнта, базуючись на величезній кількості історій хвороб пацієнтів (які постійно змінюються), а також на базі даних медичних досліджень і геномних даних? Інший приклад – з області оптимізації бізнес-процесів: як провести аналіз структурованих даних з ERP (Enterprise Resource Planning) - додатки, а також слабоструктурованих даних в вигляді логфайлів і неструктурованого тексту з відгуків покупців?

Цінність для бізнесу (Value). компанія IDC теж виділяє «чотири V», що характеризують дані, проте параметр Variety (мінливість), який застосовує компанія Forrester, вона замінює на параметр Value (цінність). IDC підкреслює, що параметр Value – один з основних, дозволяють виділити Big Data як нове явище. Він відноситься до економічного ефекту, який технологія Big Data забезпечує користувачам. Інформація – це головний аспект успішного прогнозування зростання і складання маркетингової стратегії в умілих руках маркетолога. Big Data є найточнішим інструментом маркетолога для передбачення майбутнього компанії.

Після аналізу проблем можна виділити певні області, або конкретні завдання де буде доцільно використовувати технології роботи із Big Data.

Застосування технології Big Data могжет бути корисно для вирішення наступних завдань[4]:

- краще дізнаватися своїх споживачів, залучати аналогічну аудиторію в Інтернеті;
- оцінювати рівень задоволеності клієнтів;
- знаходити і впроваджувати нові способи, що збільшують довіру клієнтів;
- створювати проекти, які користуються попитом по прогнозування ринкової ситуації;
- маркетинг і оптимізація продажів;
- ефективно сегментувати клієнтів;

– удосконалювати якість товарів і послуг;
– приймати більш обґрунтовані управлінські рішення на основі аналізу Big Data;
– оптимізувати портфель інвестицій;
– підвищувати продуктивність праці.

1.2 Аналіз аналогів

Модель MapReduce відкрила немислиму першу можливість надійної обробки великих масивів даних на кластерах, зібраних з доступних серверів. Втім, при всій сенсаційної новизни не варто випускати з уваги ту обставину, що методи прискорення, що використовують розпаралелювання обчислень в багатопроцесорної середовищі і що базуються на розділенні процесу рахунки з наступною збіркою результатів, далеко не нові. Ще півстоліття тому, в 1963 році, Мелвін Конвей запропонував модель Fork / Join, яка, на відміну від MapReduce, краще пристосована до багатоядерним процесорам.

Про цю роботу, як і про роботу Дага Лі, згадували рідко, проте до них довелося повернутися, причому в той момент, коли у компанії Google виникла необхідність обробляти великі масиви на кластерах, зібраних з власних серверів.

Пізніше з'явився опис файлової системи Google File System (GFS), а рік по тому Даг Каттінг і Майкл Кафареллі за образом і подобою цієї моделі і файлової системи створили свою відкриту версію, назвавши її Hadoop.

Сьогодні система Apache Hadoop пішла далеко вперед від першого варіанту, тепер вона складається з бібліотек і утиліт, розподіленої файлової системи Hadoop Distributed File System (HDFS), платформи для управління ресурсами Hadoop YARN, що масштабується програмної моделі Hadoop MapReduce і ще безлічі компонентів. У

2014 році почався перехід на версію Hadoop 2.0, має цілий ряд відмінностей, серед яких дві нові, більш сучасні моделі паралельних обчислень Apache Tez і Apache Spark.

Створення Apache Tez і Apache Spark було викликано вимогами підтримки отримувавши широке поширення аналітичних додатків (машинне навчання, розпізнавання мови, методи добування інформації з даних). Ітераційна природа подібних додатків суперечить організації потоку робіт в пакеті - реалізація ітерацій засобами традиційного Hadoop MapReduce виявляється надзвичайно тривалою.

Причина обмеженості MapReduce при роботі з аналітичними додатками, в машинному навчанні і т. П. Криється в використанні ациклічної моделі потоків даних (Directed Acyclic Graph, DAG), яка не передбачає будь-якої можливості вкладених циклів на всьому протязі процесу від Map до Reduce, що суперечить суті подібних додатків. Зараз, щоб пристосувати модель MapReduce для цілей аналізу, доводиться багато разів повторювати основний цикл, а це призводить до помітного зниження продуктивності. Apache Tez і Apache Spark дозволяють в десятки разів підвищити швидкість обробки: по перше це відбувається за рахунок удосконалення процедури виконання графовой моделі, а по друге – завдяки переходу на принципово іншу алгоритмічну основу.

Більш висока продуктивність алгоритмів Tez в порівнянні з Hadoop MapReduce забезпечується двома факторами. По-перше, динамічним розпізнаванням графа, що відображає той факт, що зазвичай розподілені дані виявляються динамічними за своєю природою і багато з того, що з ними треба робити, визначається вже в процесі виконання. Динаміка знаходиться в протиріччі з пакетним режимом, де все слід визначити заздалегідь, а Tez включає в себе модулі модифікації вершин графа, що дозволяють знайти краще рішення.

По-друге, Tez використовує оптимальне управління ресурсами. Ця функція реалізується їм спільно з YARN, де є елемент динаміки - потреба в ресурсах змінюється, і на різних фазах рішення задачі вони можуть бути різними. Система Tez не призначена для кінцевих користувачів і орієнтована на розробників додатків,

підтримуючи сховище даних Apache Hive і аналітичну платформу з мовою високого рівня Apache Pig.

Spark можна розглядати як подальший розвиток ідеї DAG з можливістю циклічної обробки потоків даних, розміщених в пам'яті. Відхід від пакетного режиму підтриманий новим, що не має аналогів способом представлення даних, який отримав назву Resilient Distributed Dataset (RDD). Організація даних у вигляді RDD дозволяє розподіляти і перерозподіляти дані на різних стадіях паралельних обчислень, відкриваючи шлях до ітераційним процесам, неможливим в разі DAG. Ось чому RDD виявляється ефективним з точки зору як швидкості обробки, так і забезпечення високої надійності, необхідної при розподіленої роботі на кластерах.

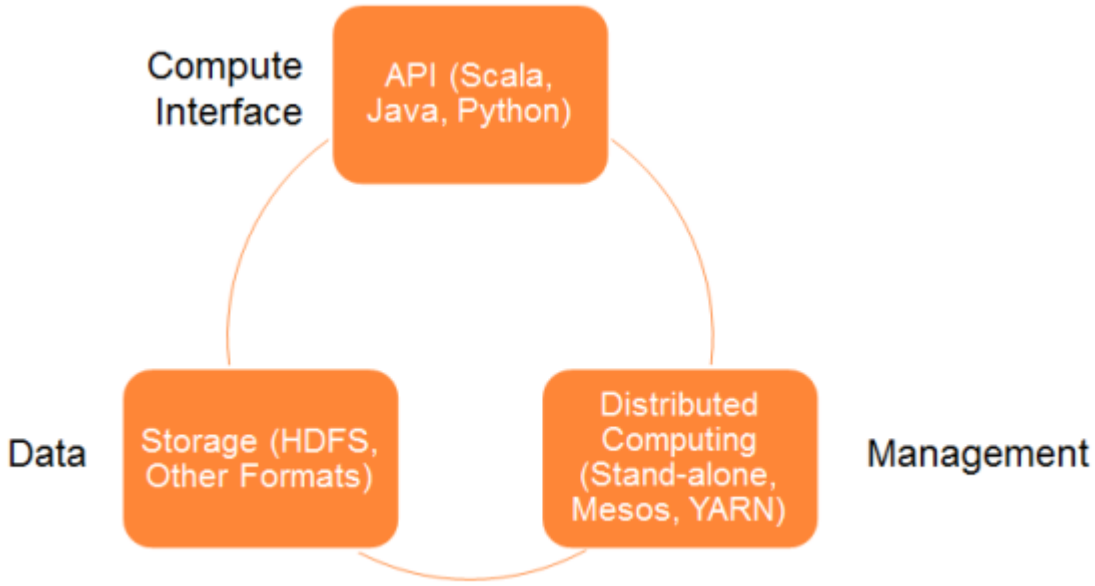

Рисунок 2 – Архітектура «Spark»

На даний момент є не так багато аналітичних систем, що працюють на Tez і Spark, але навіть вже відомі результати дозволяють віднести ці моделі, і особливо останню, до числа суперзірок Open Source, здатних визначити майбутнє індустрії Великих Даних.

1.3 Аналіз методів що використовуються

Великі дані – новий суттєвий елемент в інформаційній структурі сучасного суспільства. Саме тому, технології та методи їх аналізу настільки значущі.

Залежно від цілей, які ставляться перед аналізом великих даних, можна виділити наступні його основні різновиди:

Інформаційно-пошуковий і візуальний аналіз. Чи не набуваючи нових знань про предмет, ми отримуємо можливість розглянути його з різних точок зору і по частинах. Це досягається видачою конкретного запиту до реляційної бази даних. Такий вид аналізу рідко застосовують до великих даних, так як розкид варіацій для конкретного запиту буде дуже великий.

Оперативно-аналітичний аналіз або OLAP (OnLine Analytical Processing). В OLAP дані агрегуються до будь-якого ступеня узагальнення з будь-якого розрізу. В даному випадку вже є можливість виявити закономірності і тенденції в даних, які інакше були б не видно. Практично OLAP вводить нас в сферу узагальнених даних.

Інтелектуальний аналіз або Data Mining. Цей тип аналізу спрямований на виявлення прихованих закономірностей в даних (повторюваних шаблонів або кластерів даних), тобто, моделей, що лежать в основі структури даних і дають можливість краще розуміти дані і передбачати їхню поведінку. Саме Data Mining є безпосереднє виявлення знань.

Майер-Шенбергер В. і Кукьер К. в якості базового підходу до аналізу великих даних виділяють кореляційний аналіз. Саме він, на їхню думку, лежить в основі прогностичної аналітики, яка використовує великі дані. Суть її полягає у відповіді на питання «Що?», але при цьому вона не завжди в змозі відповісти на питання «Чому?». Метод кореляцій виявився мало застосуємо і володіє слабкою ефективністю при аналізі малих обсягів даних, але виявився кращим підходом для

аналізу великих обсягів інформації, де точність на обсязі величезної вибірки втрачає своє значення.

1.4 Постановка задачі

Метою даної роботи є дослідження застосовності і актуальності статистичної оптимізації в задачах MapReduce. Для цього необхідно:

- реалізувати збір статистики розподілу проміжних значень за проміжними ключам;
- реалізувати алгоритм, що забезпечує рівномірне навантаження на Reduce машини на основі зібраної статистики;
- реалізувати інструментарій для порівняння розподілів проміжних ключів по reduce-машинам;
- провести тестування та порівняльний аналіз оптимізованої версії на реальних завданнях.

2 АНАЛІЗ МЕТОДІВ ТА МОДЕЛЕЙ ЩОДО РЕАЛІЗАЦІЇ В ПРЕДМЕТНОЙ ГАЛУЗІ

2.1 Екосистема «HADOOP»

Як вже говорилося раніше, екосистема «Hadoop» є однією з найважливіших технологій «Big Data».

«Hadoop» – це програмна платформа з відкритим вихідним кодом, що знаходиться під управлінням «Apache Software Foundation». «Hadoop» використовується для надійних, масштабованих і розподілених обчислень, так як має велику обчислювальною потужністю, але може також застосовуватися і як сховище даних, яке може вмістити гігантську кількість інформації.

Дана платформа широко поширена в багатьох сферах, де має місце робота з «Big Data».

Навколо «Hadoop» утворилася ціла екосистема з пов'язаних проектів і технологій, багато з яких з самого початку розвивалися в рамках проекту, а пізніше перейшли в самостійні. Зараз йде активний процес комерціалізації технологій, кілька компаній будують бізнес, який повністю спрямований на створення комерційних дистрибутивів «Hadoop» і послуг з підтримки екосистеми.

Варто відзначити, що практично всі великі компанії, спеціалізуються на постачанні інформаційних технологій для організацій, включають «Hadoop» (в тому чи іншому вигляді) в свої лінійки рішень.

2.1.1 Використання і переваги

Екосистема «Hadoop» використовується в наступних компаніях:

– «IBM»;
– «ebaY»;
– «Twitter»;
– «Mail.ru group»;
– «Amazon»;
– «Adobe»;
– «Яндекс»;
– «Yahoo»;
– «Facebook»;
– «LinkedIn».

Особливості даної платформи:

– працює з «Big Data» на звичайних серверах;
– сильне відкрите співтовариство;
– безліч різних продуктів і засобів використовують «Hadoop».

Для розгортання кластера «Hadoop» зазвичай використовують звичайні сервера (не суперкомп'ютери або десктопи), з'єднані по мережі і розташовані в одному місці.

Переваги «Hadoop» [7]:

Можливість зберігати і обробляти величезну кількість даних.

З розвитком соціальних мереж та інших рішень, доводиться мати справу з постійно зростаючими обсягами даних. Тому це дуже важливий фактор.

Обчислювальна потужність.

Модель розподілених обчислень «Hadoop» обробляє великі дані швидко. Чим більше обчислювальних вузлів, які ви використовуєте, тим більше обчислювальної потужності ви маєте.

Відмовостійкість.

Дані та обробка додатків захищені від відмови обладнання. Якщо вузол втрачає працездатність, то завдання автоматично перенаправляються до інших вузлів, щоб

упевнитися, що розподілені обчислення не перестали працювати. Багаторазові копії (реплікація) всіх даних виробляються автоматично.

Гнучкість.

На відміну від традиційних реляційних баз даних, ви не повинні попередньо обробляти дані перш, ніж зберегти їх. Ви можете зберігати стільки даних, скільки ви хочете (різних типів) і вирішуєте, як використовувати їх пізніше. Це включає неструктуровані дані як текст, зображення і відео.

Низька вартість.

Платформа з відкритим вихідним кодом безкоштовна і використовує товарні апаратні засоби, щоб зберігати велику кількість даних.

Масштабованість (горизонтальна).

Ви можете легко наростити свою систему, щоб обробити більше даних, просто додавши вузли. І для цього не потрібно глибокого адміністрування.

Інкапсуляція складності реалізації.

«Hadoop» приховує багато складності розподілених і багатопоточних систем. Звільняє розробника від турботи про проблеми системного рівня, таких як очікування даних, організація передачі даних, розподіл даних, доставка коду. Дозволяє розробнику сфокусуватися на розробці додатків і реалізації бізнес-логіки.

2.1.2 Опис компонентів

Основними компонентами (ядро) «Hadoop» є[8]:

– «Hadoop Distributed File System» (HDFS) – розподілена файлова система, що дозволяє зберігати інформацію практично необмеженого обсягу;

– «YARN» - програмна модель для управління ресурсами і менеджменту завдань, в тому числі включає програмну модель «MapReduce» (використовуються в нових версіях «Hadoop» замість «MapReduce»);
– «Hadoop common» - бібліотеки і утиліти, які використовуються іншими модулями «Hadoop»;
– «Hadoop MapReduce» - програмний каркас для програмування розподілених обчислень в рамках парадигми «MapReduce».

На рисунку 3 зображено схему основних компонентів екосистеми «Hadoop»:

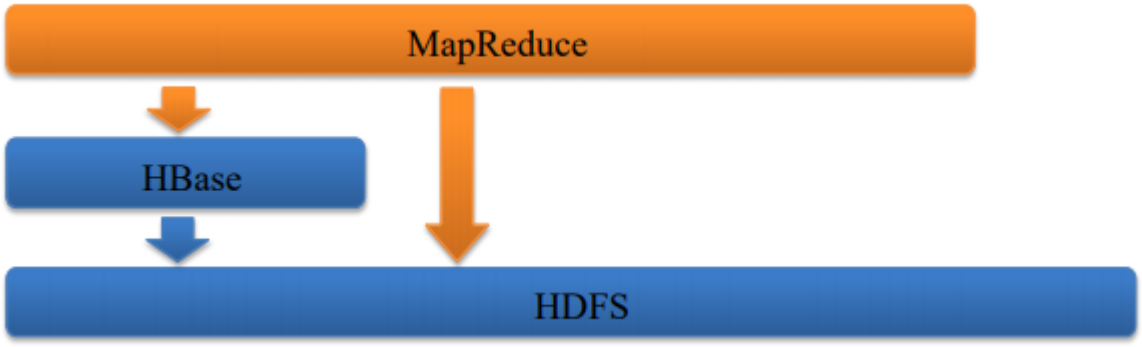

Рисунок 3 – Схема основних компонентів екосистеми «Hadoop»

Також існує велика кількість проектів, безпосередньо пов'язаних з «Hadoop», але який не входять до набір основних компонентів «Hadoop»[10]:

– «Hive» – інструмент для «SQL-like запитів» над великими даними (перетворює «SQL-запити» в серію «Map Reduce – задач »);
– «Pig» – мова програмування для аналізу даних на високому рівні. Одна рядок коду на цій мові може перетворитися в послідовність «MapReduce – завдань»;
– «HBase» – колоночная база даних, яка реалізує парадигму «BigTable»;
– «Cassandra» – високопродуктивна розподілена «key – value » база даних;
– «ZooKeeper» – сервіс для розподіленого зберігання конфігурацій і синхронізації змін цих змін;
– «Mahout» – бібліотека і движок машинного навчання на великих даних.

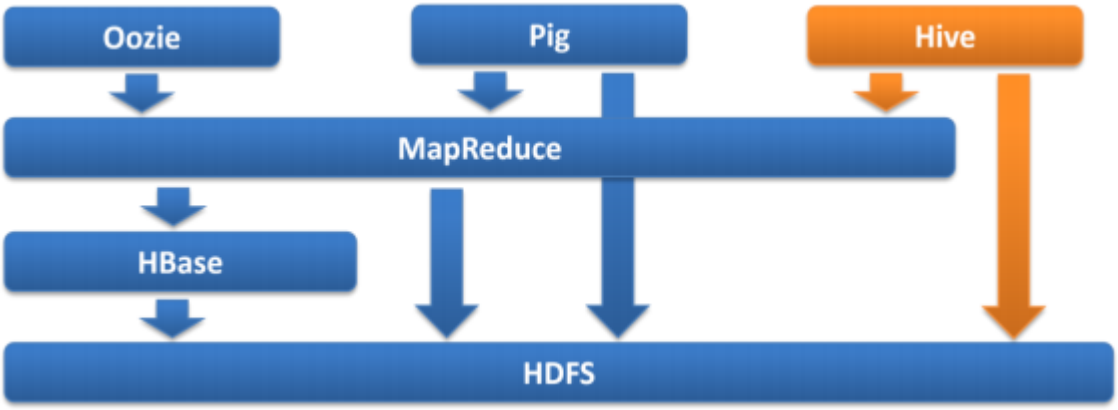

Рисунок 4 – Схема всіх компонентів екосистеми «Hadoop»

Дані модулі розширюють можливості платформи, що дозволяє розробникам зберігати і аналізувати різні типи даних.

2.2 Інструменти екосистеми Apache Hadoop

Навколо Hadoop будувалося багато спеціалізованих проектів, які згодом перетворилися в цілу екосистему. Найзначніші і добре підтримувані отримали статус офіційних під-проектів Apache Hadoop. До їх числа відносяться:

– «Pig» – високорівнева мова опису потоків даних;
– «Hive» – SQL-подібна інфраструктура для організації сховищ даних;
– «HBase» – розподілена СУБД зі зберіганням по стовпцях, влаштована за зразком Google BigtaBle;
– «ZooKeeper» – надійна система координації для управління станом, загальним для декількох розподілених додатків.

2.2.1 Pig

Pig підвищує рівень абстракції при обробці великих наборів даних. MapReduce дозволяє програмісту задати функцію відображення, а потім функцію згортки, але для адаптації механізму обробки даних до цієї схеми часто доводиться використовувати кілька стадій MapReduce, а це ускладнює задачу. При використанні Pig структури даних набагато складніше, з безліччю значень і багаторівневої ієрархією, а перетворення, застосовувані до даних, набагато потужніше.

Pig складається з двох основних частин:

– мова для опису потоків даних, званий Pig Latin;
– виконавча середовище для запуску програм Pig Latin. В даний час доступні два варіанти: локальне виконання на одній JVM і розподілене виконання в кластері Hadoop.

Програма Pig Latin складається з серії операцій (перетворень), які застосовуються до вхідних даних для отримання вихідних даних. В цілому ці операції описують потік даних, який перетворюється виконавчої середовищем Pig в який виконувався уявлення, а потім запускається для виконання. У внутрішній реалізації Pig трансформує перетворення в серію завдань MapReduce, однак ця трансформація в основному залишається прихованою від програміста, що дозволяє йому зосередитися на даних, а не на природі виконання.

Однією з цілей розробки Pig була хороша розширюваність. Налаштовуються практично всі стадії обробки: завантаження, зберігання, фільтрація, угруповання, з'єднання – кожна операція може бути змінена за допомогою призначених для користувача функцій (UDF, User-Defined Function). Ці функції працюють з вкладеною моделлю даних Pig, тому вони можуть дуже глибоко інтегруватися з операторами Pig. Крім того, призначені для користувача функції зазвичай краще підходять для

повторного використання, ніж бібліотеки, розроблені для написання програм MapReduce.

Основний об'єкт в Pig Latin – це «відношення». Саме з відносинами працюють всі оператори мови. У формі відносин представляються вхідні і вихідні дані.

Кожне відношення являє собою набір однотипних об'єктів – «Кортежів» (tuples). Аналоги в БД: кортеж – це рядок, ставлення – це таблиця. Кортежі можуть в свою чергу об'єднаються в колекції, звані bag.

Кортежі відповідно складаються з нумерованих або іменованих об'єктів - «полів», довільних базових типів (число, рядок, булева змінна і т.д.).

2.2.2 Hive

Hive – це пакет для організації сховищ даних, побудований на базі Hadoop. Орієнтований він на аналітиків, які впевнено володіють SQL і потребують засобі для виконання довільних запитів, агрегування і аналізу даних, обсяг яких такий, що потрібно Hadoop. Для взаємодії з Hive застосовується SQL-подібна мова запитів, званий HiveQL.

Hive видає із себе движок, який перетворює SQL-запити в ланцюжок map-reduce завдань. Движок включає в себе такі компоненти, як Parser (розбирає SQL-запити які отримує на вхід), Optimizer (оптимізує запит для досягнення більшої ефективності), Planner (планує завдання на виконання) Executor (запускає завдання на фреймворку MapReduce.

Для роботи Hive також необхідно сховище метаданих. Справа в тому що SQL передбачає роботу з такими об'єктами як база даних, таблиця, колонки, рядки,

клітинки. Оскільки самі дані, які використовує Hive зберігаються просто у вигляді файлів на HDFS - необхідно десь зберігати відповідність між об'єктами Hive і реальними файлами. Зазвичай мета-сховище розміщується в реляційної базі даних.

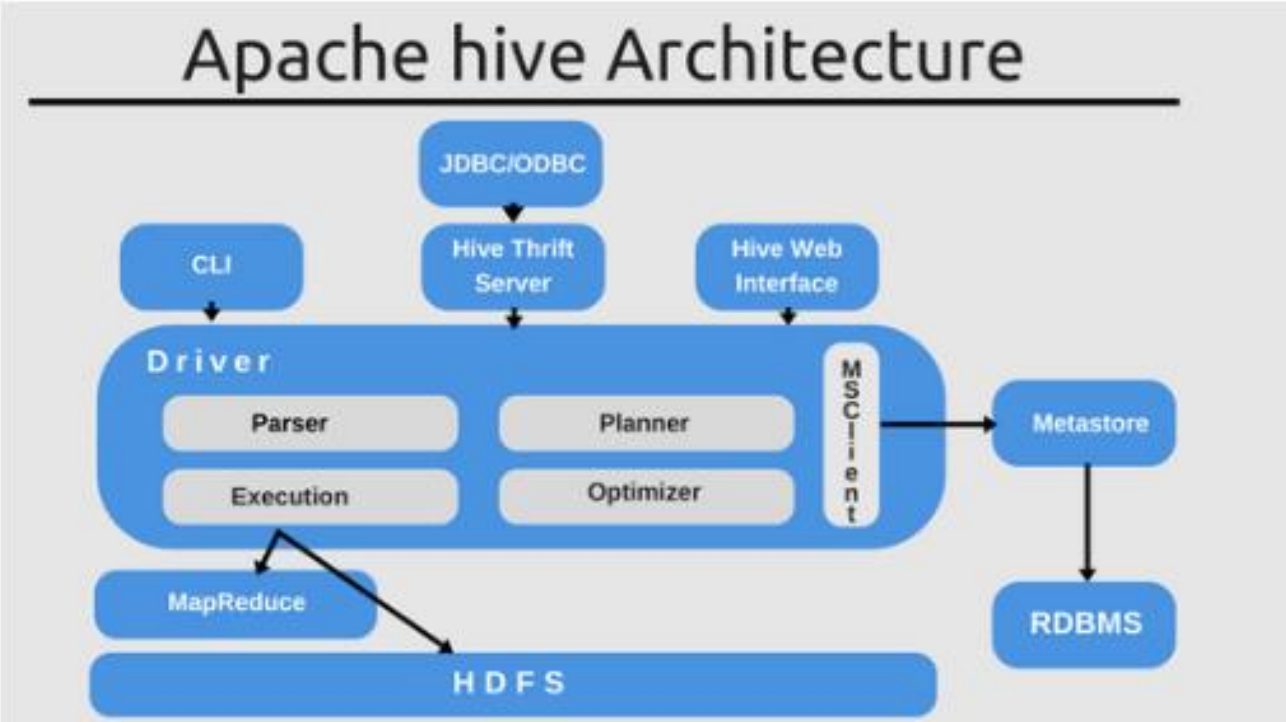

Рисунок 5 – Структурна схема «Apache Hive»

Pig і Hive мають приблизно однакові можливості і створені для однієї і тієї ж мети - абстрагувати розробника від безпосередньої розробки програм в рамках MapReduce. Але тим не менш кілька відмінностей існує.

– «Pig» реалізує процедурний підхід, в той час як Hive використовує декларативний SQL підхід. Тому при використанні складних вибірок запити в HQL стають дуже громіздкими, а Pig дозволяє розбивати логіку на блоки, кожен крок можна розгорнуто описувати коментарями;
– розробка на Pig складніша і передбачає вивчення Pig Latin, в той час як HQL практично ідентичний всім знайомому SQL.

Так як розвиток системи передбачає збільшення джерел даних, і складність запитів буде тільки зростати, було вирішено спочатку використовувати Apache Pig

2.2.3 HBase

HBase – розподілена на стовпці та орієнтована на них, побудована на основі HDFS. Ця програма Hadoop, що використовується в ситуаціях, коли вам необхідно організувати довільний доступ для читання / запису до дуже великих наборів даних в реальному часі.

Додатки зберігають дані в таблицях, що складаються з рядків і стовпців. Для елементів таблиці (перетину рядків і стовпців) діє контроль версії. За умовчанням як версії використовується тимчасова мітка, автоматично призначається HBase на момент вставки.

Вміст клітинки представляє неінтерпретуемий масив байтів. Ключі рядків таблиці також є байтовими масивами, тому теоретично ключем рядка може бути що завгодно – від рядків до довічних уявлень long і навіть серіалізованих структур даних. Рядки таблиці упорядковані відповідно до ключу рядків (первинному ключу таблиці). Сортування здійснюється в порядку проходження байтів. Всі звернення до таблиці виконуються по первинному ключу

Колонки організовані в групи колонок, звані Column Family. Як правило в одну Column Family об'єднують колонки, для яких однаковий патерн використання і зберігання. Записи фізично зберігаються в відсортованому по RowKey порядку. При цьому дані відповідні різним Column Family зберігаються окремо, що дозволяє при необхідності читати дані тільки з потрібного сімейства колонок.

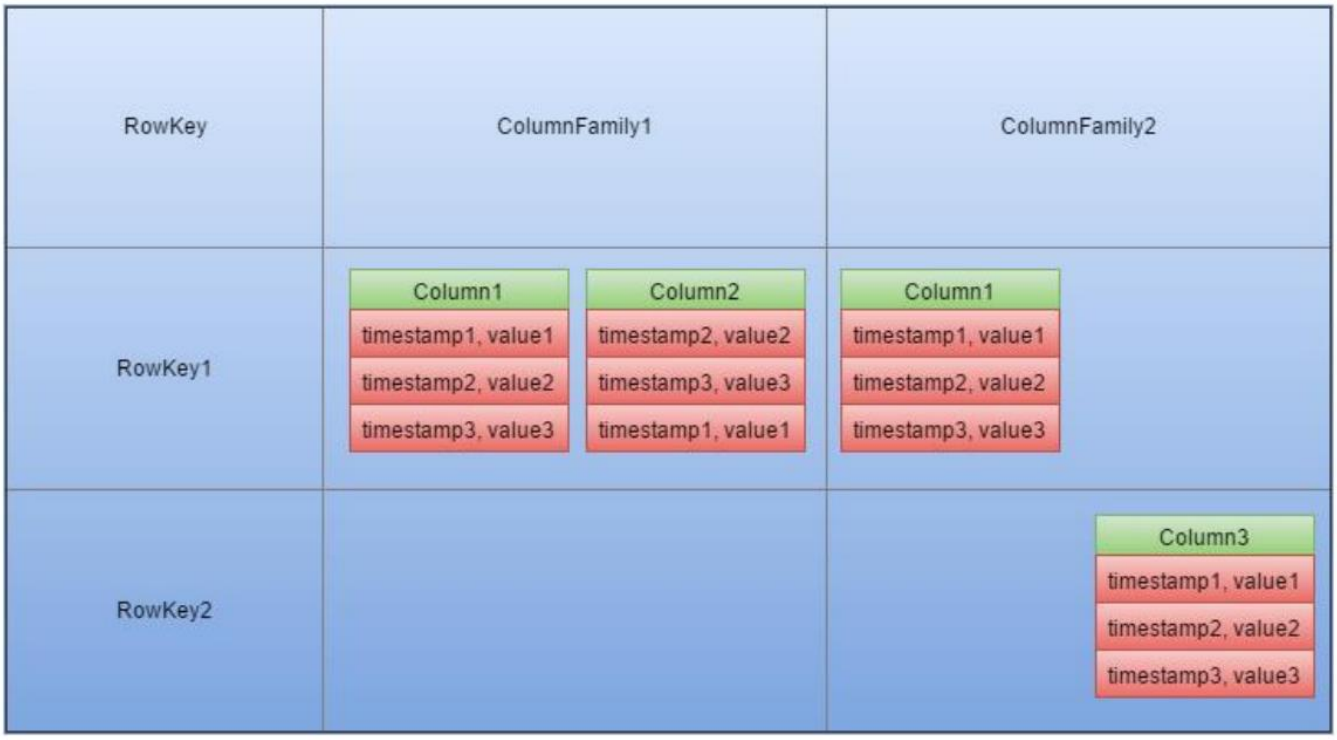

Рисунок 6 – Схема організації даних в «HBase»

HBase є розподіленою базою даних, яка може працювати на десятках і сотнях фізичних серверів, забезпечуючи безперебійну роботу навіть при виході з ладу деяких з них. Тому архітектура HBase є доволі складною у порівнянні з класичними реляційними базами даних.

HBase для своєї роботи використовує дві основні сутності.

Region Server – обслуговує один або кілька діапазонів записів які відповідають певному діапазону ключів (Регіонів). Кожен регіон містить:

– основне сховище даних. HBase працює поверх HDFS, причому використовує власний формат файлів – HFile, в якому дані зберігаються в відсортованому по ключу в лексеграфічному порядку;
– буфер на запис. Так як дані зберігаються в HFile в відсортованому порядку - оновлювати ці файли на кожну запис досить дорого, тому дані під час запису

потрапляють в спеціальну область пам'яті, де накопичуються, і через деякий час записуються в основне сховище;

– кеш на читання. Дозволяє суттєво економити час на даних які читаються часто;
– при використанні буфера на запис існує деякий ризик втрати даних через збій, тому, для забезпечення відмовостійкості, всі операції перед виконанням потрапляють в спеціальний журнальний файл - так званий Write Ahead Log.

Master Server – головний сервер в кластері HBase. Master управляє розподілом регіонів, веде їх реєстр, управляє запусками регулярних завдань.

Для координації дій між сервісами HBase використовує Apache ZooKeeper, спеціальний сервіс, призначений для управління конфігураціями і синхронізацією сервісів.

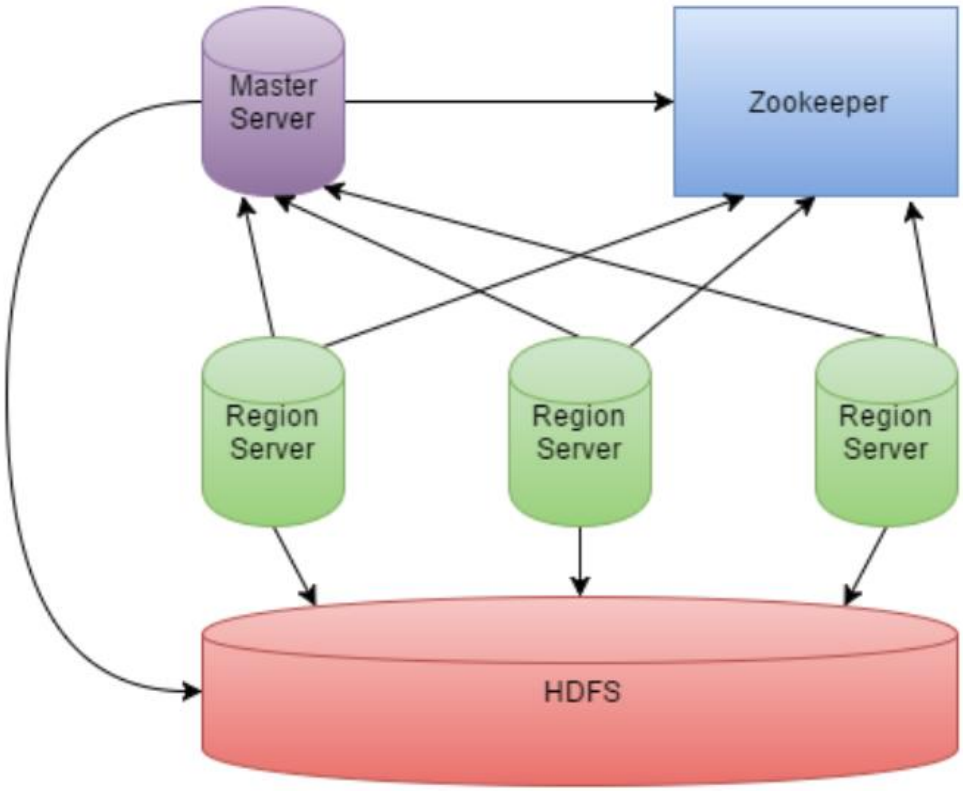

Рисунок 7 – Функціональна схема побудови кластера «HBase»

При збільшенні кількості даних в регіоні і досягнення певного розміру HBase запускає split, операцію розбиває регіон на 2. Для того щоб уникнути постійних поділів регіонів – можна заздалегідь задати кордон регіонів і збільшити їх максимальний розмір. HBase призначений головним чином для читання і доступу до великих даних. Використання HBase виправдано, коли дані часто оновлюються і видаляються і потрібен доступ до даних по певним ключам. В силу цих причин в рамках нашого завдання Apache HBase використовуватися не буде.

2.3 Розподілена файлова система «HDFS»

Експоненціальне зростання інформації привів до потреби в появі файлових систем, орієнтованих на забезпечення високої продуктивності, масштабованості, надійності і доступності. Так з'явилася розподілена файлова система «HDFS» – «Hadoop Distributed File System» (аналог «GSF» від «Google»). проект активно підтримується і розвивається компанією «Yahoo».

«HDFS» – розподілена файлова система, яка використовується в проекті «Hadoop». «HDFS – кластер» в першу чергу складається з наступних компонентів[11]:

– «NameNode» – основний сервер для управління простором імен файлової системи і доступом клієнтів до даних. Даний сервер, при першому запуску, фіксує всі транзакції, пов'язані зі зміною метаданих файлової системи, в файлі під назвою «EditLog» і застосовує ці зміни до образу «HDFS», розташований в файлі «FsImage». Потім записується вже новий образ із змінами, і система починає роботу з чистим «EditLog»;

– «DataNode» – сервер для безпосереднього зберігання даних. Передача даних відбувається безпосередньо між «DataNode» і клієнтом, щоб розвантажити «NameNode»;
– «Secondary NameNode» – сервер, який виконує функції «NameNode», що стосується файлів «EditLog» і «FsImage». При цьому дані файли зберігаються на основному сервері.

Дані в «HDFS» зберігаються у вигляді блоків (блок – одиниця зберігання файлів) на «DataNode» і управляється через «NameNode». Причому під час запису файлу відбувається процес реплікації (копіювання) даного файлу. Стандартний розмір блоків в «HDFS» – 64МБ або 128 МБ. Основним мотивом для визначення такого розміру є зменшення часу пошуку даних в порівнянні з їх передачею. При цьому відбувається процес реплікації (копіювання) даних.

Копії блоків зберігаються на декількох серверах, за замовчуванням – трьох. Їх розміщення відбувається наступним чином:

– перша репліка розміщується на локальному «DataNode»;
– друга репліка на іншому «DataNode» в цій же стійці;
– третя репліка на довільній «DataNode» іншої стійки;
– решта репліки розміщуються довільним способом.

На рисунку 5 зображена схема зберігання даних на «HDFS»:

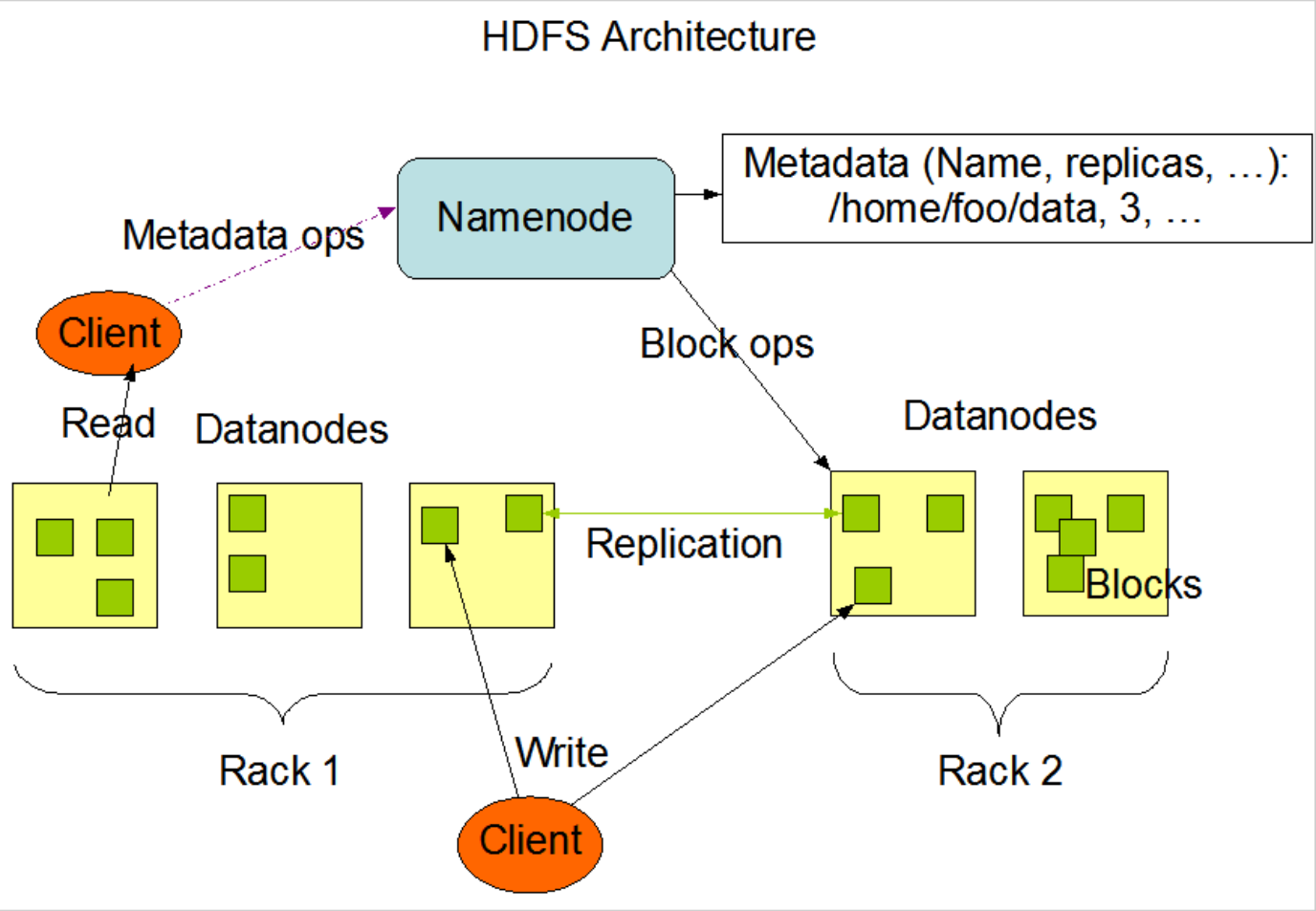

Рисунок 8 – Архітектура «HDFS»

При читанні даних клієнт вибирає найближчий до нього «DataNode» з реплікою.

2.4 Програмна модель «MapReduce»

«MapReduce» - програмна модель для виконання розподілених обчислень для великих обсягів даних, що представляє собою набір «Java – класів» і виконуваних утиліт для створення і обробки завдань на паралельну обробку.

Основні принципи «MapReduce» можна сформулювати так[15]:

– обробка та обчислення великих обсягів даних;

– масштабованість;

– автоматичне розпаралелювання завдань;

– робота на простому обладнанні;

– автоматична обробка відмов виконання завдань.

Роботу «Map Reduce» можна поділити на такі етапи[17]:

Читання вхідних даних.

Вхідні дані діляться на блоки даних визначеного розміру (від 16 Мб до 128 Мб), які називаються сплити. «MapReduce» закріплює за кожною функцією «Map» певний сплати.

Функція «Map».

Кожна функція Map отримує на вхід список пар «ключ/значення», обробляє їх і на виході отримує нуль або більше пар «ключ/значення», є проміжним результатом.

Всі операції «Map» виконуються паралельно і не залежать від результатів роботи один одного. Кожна функція «Map» отримує на вхід свій унікальний набір даних.

Функція «Reduce».

Програмна модель викликає функцію «Reduce» для кожного унікального ключа в списку значень. Операції «Reduce» виконуються паралельно і не залежать від результатів роботи один одного.

Таким чином, результати роботи кожної функції «Reduce» пишуться в окремий блок «HDFS».

Запис вихідних значень.

Результати, отримані на етапі «Reduce», записуються в файлові блоки в «HDFS». Кожен вузол «Reduce» пише в власний блок.

На рисунку 6 зображена схема роботи «MapReduce»:

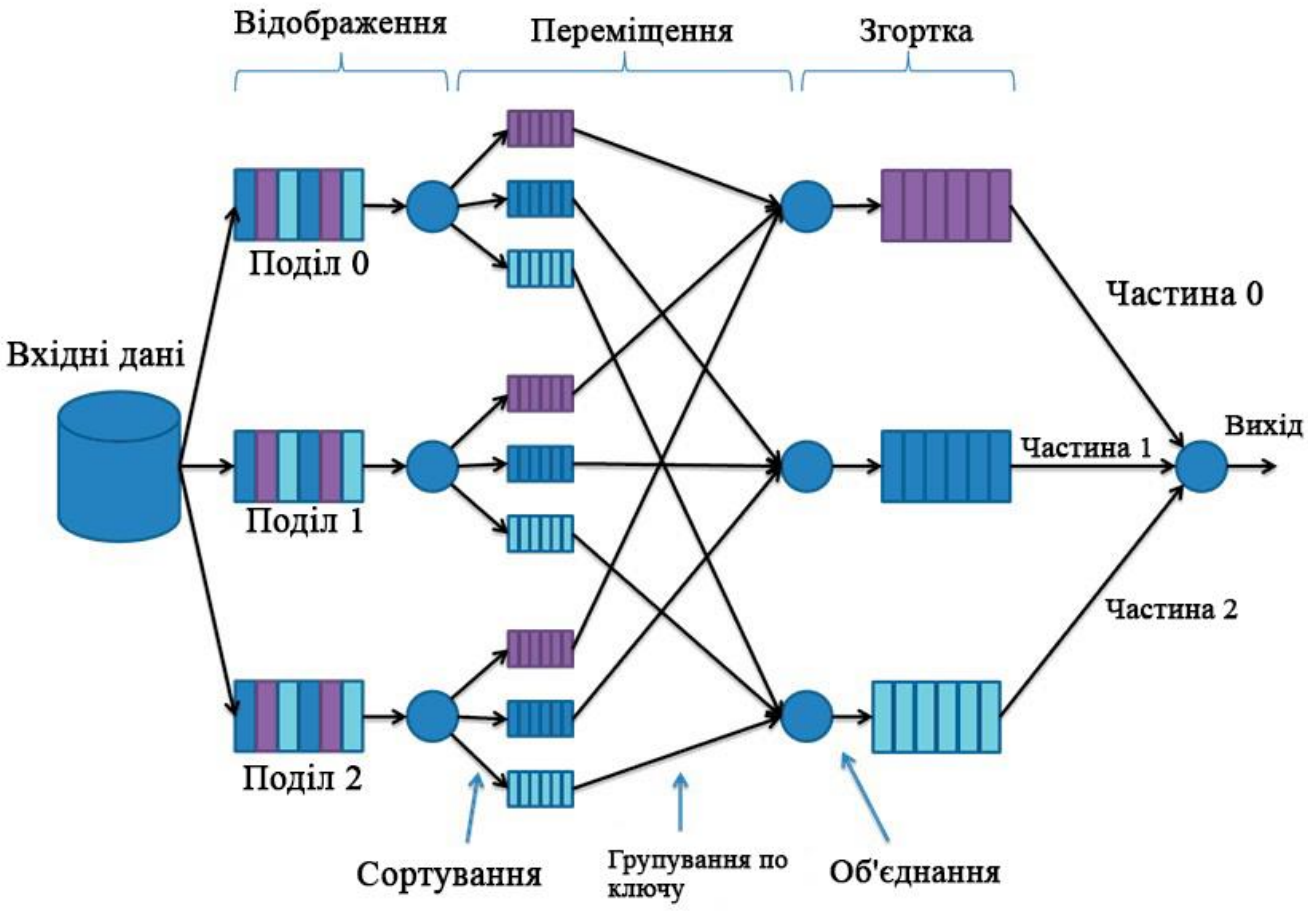

Рисунок 9 – Схема роботи «MapReduce»

Розробнику додатків для «Hadoop MapReduce» необхідно написати базовий обробник, який на кожному обчислювальному вузлі забезпечить освіту початкових пар «ключ / значення» в проміжний набір пар «ключ/значення'» (клас, який реалізує інтерфейс «Mapper»), і обробник, зводить проміжний набір пар в остаточний набір пар (клас, який реалізує інтерфейс «Reducer») [7].

Всі інші етапи виконуються програмної моделлю без додаткових зусиль розробника.

«Map Reduce» виконує наступні функції:

– планування завдань;

– розпаралелювання завдань;

– перенесення завдань до даних;

– синхронізація виконання завдань;
– обробка відмов виконання завдань і їх перезапуск;
– оптимізація мережевих взаємодій.

2.4.1 Архітектура « Hadoop MapReduce»

«MapReduce» використовує архітектуру «Master – Worker», де «Master» – єдиний екземпляр керуючого процесу («JobTracker»), запущений на окремій машині. «Worker» – це довільна кількість процесів «TaskTracker», що виконуються на «DataNode».

«JobTracker» і «TaskTracker» знаходяться над рівнем зберігання «HDFS», і запускаються за такими правилами:

– примірник «JobTracker» виповнюється на «NameNode»;
– примірники TaskTracker виконуються на «DataNode».

Вищеописані принципи розташування «JobTracker» і «TaskTracker» дозволяють істотно скоротити обсяги переданих по мережі даних та мережеві затримки, пов'язані з передачею цих даних. «JobTracker» є єдиним вузлом, на якому виконується додаток «MapReduce», яке викликається клієнтом.

«JobTracker» виконує наступні функції:

– планування індивідуальних (по відношенню до «DataNode») задач «Map» і «Reduce»;
– координація завдань;
– моніторинг виконання завдань;
– перепризначення провалилися завдань інших вузлів «TaskTracker».

У свою чергу, «TaskTracker» виконує наступні функції:

– виконання завдань «Map» і «Reduce»;

– управління виконанням завдань;
– відправлення повідомлень про статус завдання вузлу «JobTracker» ;
– відправлення діагностичних повідомлень вузлу «JobTracker».

Взаємодія «TaskTracker» з «JobTracker» йде за допомогою «RPC – викликів», причому вони йдуть тільки від« TaskTracker ». Таке рішення зменшує залежність керуючого процесу «JobTracker» від процесів «TaskTracker».

Взаємодія «JobTracker» з клієнтом проходить за наступною схемою[16]:

– «JobTracker» приймає завдання від клієнта і розбиває завдання на безліч «Map» завдань і безліч «Reduce» завдань. Вузол «JobTracker» використовує інформацію про файлових блоках, розташовану в «NameNode», щоб вирішити, скільки завдань потрібно створити на вузлах «TaskTracker»;
– «TaskTracker» отримує від «JobTracker» список завдань, завантажує код і виконує його. З певною періодичністю «TaskTracker» відсилає «JobTracker» статус про виконання завдань.

Взаємодії «TaskTracker» з клієнтом відсутні.

У разі виникнення збою «TaskTracker» «JobTracker» перепризначає завдання іншого вузла «TaskTracker».

У разі несправності «JobTracker», для продовження виконання «MapReduce-завдання», необхідний перезапуск «JobTracker». при перезапуску «JobTracker» зчитує з журналу дані, про останню успішної контрольної точці, відновлює свій стан на момент запису і продовжує роботу.

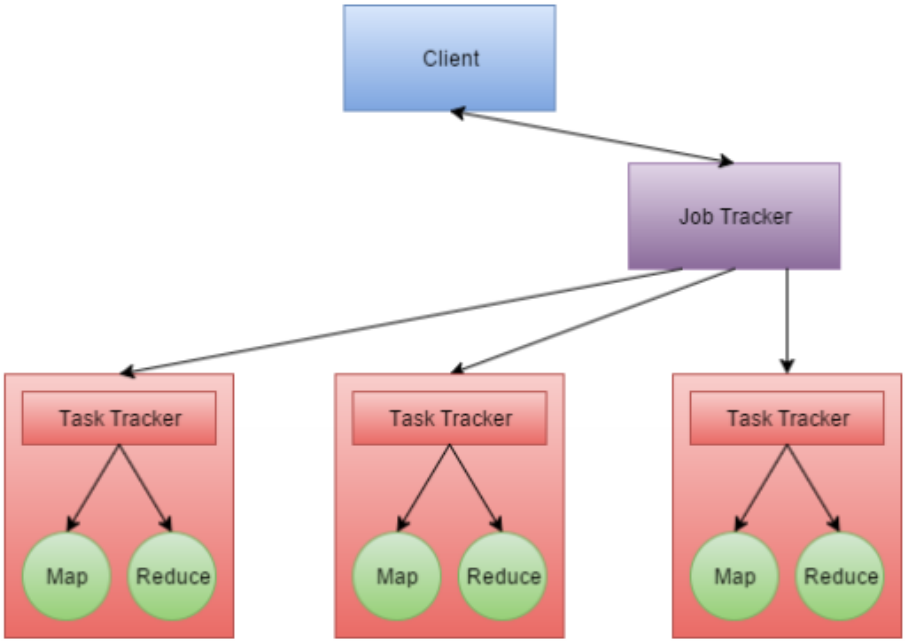

Рисунок 10 – Ієрархія демонів обчислення Hadoop

Основні переваги:

– ефективна робота з великим об'ємом даних;
– масштабованість;
– відмовостійкість;
– відкритий вихідний код.

Недоліки «MapReduce»:

– ефективність застосування «MapReduce» знижується при малому кількості комп'ютерів;
– не можна визначити закінчення стадії «Map» ;
– затримки у виконанні будь-якої запущеної «Map» завдання веде до затримки виконання завдання цілком;
– збій вузла «JobTracker» призводить до простою всього кластера.

Отже, якщо підвести підсумки то можна сказати, що дана платформа надає великі можливості по аналізу і зберігання даних будь-якого розміру і типу.

3 АНАЛІЗ ПРОГРАМНИХ РІШЕНЬ ЩОДО РЕАЛІЗАЦІЇ В ПРЕДМЕТНОЙ ГАЛУЗІ

3.1 Існуючі роботи по оптимізації

Розглянемо деякі роботи по оптимізації MapReduce програм. У більшій частині робіт оптимізації SQL запитів адаптуються для MapReduce. Прикладом такої роботи може бути работа [5], в якій представлений інструмент Manimal, вирішальний проблему знаходження в програмах MapReduce операцій проекції, вибірки, агрегації, які Переміщення з іншого програмної логікою, і застосовує оптимізації для цих операцій. В іншій роботі [6] розглянута оптимізація Join в задачах MapReduce. Стратегії, представлені в ній, добре працюють в разі join-а однією дуже великий таблиці з декількома маленькими.

3.2 Відкрита реалізація MapReduce. Проект Hadoop

Найбільш поширеною реалізацією моделі MapReduce є HadoopTM Project [7], основний внесок в який зробили розробники з компаній Facebook і Yahoo! Hadoop був успішно впроваджений в цих компаніях [8]. Також цей проект офіційно підтримується на Amazon EC2, де доступні офіційні AMI (Amazon Machine Images), які відразу містять в собі Hadoop. Крім того, в дистрибуційному пакеті Hadoop знаходяться інструменти, які дозволяють порівняно легко розгорнути кластер с Hadoop на Amazon EC2 і запускати MapReduce програми. На жаль, ці інструменти і AMI на даний момент давно не оновлювалися і сильно застаріли. Про це докладніше буде сказано в частині 5.1.

З причини широкої підтримки цього проекту і можливості розгорнути кластер на Amazon EC2 було вирішено проводити дане дослідження на його основі. Розглянемо підпроекти, складові його:

– Hadoop Common: загальні утилітарні класи, необхідні для інших підпроектів Hadoop;
– Hadoop Distributed File System (HDFS): розподілена файлова система [9] з високою пропускною здатністю;
– Hadoop MapReduce: фреймворк для написання і виконання програм MapReduce на обчислювальних кластерах.

Найбільший інтерес для нас представляє останній підпроект: Hadoop MapReduce. Опишемо ключові компоненти, що входять до нього, і які будуть використовуватися в подальшому в цій роботі. найбільш важливий клас для даного дослідження - це Partitioner, абстрактний клас, який відповідає за розподіл ключів, отриманих на стадії Map, по Reduce машинам. Він має всього лише один метод з сигнатурою:

```
abstract class Partitioner <KEY, VAL>{
    abstract int getParrtition(KEY key, VAL value, int k)
```

Цей метод визначає, на яку з k машин буде відправлена пара (key, value) для згортки. Реалізацією за замовчуванням цього класу є клас HashPartitioner, в якому метод getPartition бере хеш код ключа key по модулю k. Варто зазначити, що ця реалізація не є специфічною для проекту Hadoop, вона згадувалася ще в найпершій публікації про парадигмі MapReduce [2].

Двома іншими основоположними класами є абстрактні класи Mapper і Reducer. Класи, наслідники від Mapper, реалізують призначену для користувача функцію map, аналогічно класи спадкоємці Reducer реалізують призначену для користувача функцію reduce.

4 МАТЕМАТИЧНА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ ДЛЯ ПРЕДМЕТНОЇ ГАЛУЗІ ТЕМИ ДИПЛОМА

4.1 Побудова алгоритму

Припустимо, що ми вміємо збирати статистику по проміжним ключам і на основі її передбачати кількість проміжних значень, відповідних кожному проміжному ключу або групі ключів. За допомогою цих даних треба завантажити reduce машини так, щоб reduce стадія на всіх них займала однаковий час. Будемо виходити з природного припущення, що час виконання reduce завдання на одній машині відповідно сумарній кількості проміжних значень по всіх проміжних ключам, оброблюваних на даній машині.

Таким чином, хочеться домогтися того, що ця кількість на всіх машинах було однаковим. Якщо пронумерувати ключі, то ми можемо отримати масив, в якому на i-ій позиції стоїть кількість проміжних значень відповідних i-ому проміжного ключу, і цей масив потрібно розбити на k якомога ближчих за сумою множин.

4.2 Математична постановка задачі

Дан масив позитивних чисел $a_1,\ldots,a_n$ і його треба розбити на k множин $M_1,\ldots,M_k$ так, щоб різниця max (S_1,..., S_k) – min (S_1,..., S_k), була мінімальною, де

$$S_k = \sum_{a \in M_k} a.$$

Це один з окремих випадків відомої K – balance partition problem, яка в житті зустрічається в різних формулюваннях таких як: розподіл мережевих запитів до серверів або завдань по процесорам; вона є NP-повної [10]. За рішенням цієї та

аналогічних їй завдань з різними обмеженнями на вхідні дані існує безліч дослідних робіт [11, 12], метою ж даної роботи була розробка або поліпшення існуючих підходів.

У поточній реалізації використовувався найпростіший алгоритм, аналогічний алгоритму First Fit Descending [13].

Найменше можливе B надалі позначатимемо OPT.

Задача пакування в ємності може бути задана як задача лінійного програмування наступним чином:

Мінімізувати:

$$B = \sum_{i=1}^{n} y_i$$

При обмеженнях:

$$\sum_{j=1}^{n} a_j x_{ij} \leq V y_i, \forall \in \{1, \dots, n\}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \forall \in \{1, \dots, n\}$$

$$y_i \in \{0,1\}, \forall i \in \{1, \dots, n\}$$

$$x_{ij} \in \{0,1\}, \forall i \in \{1, \dots, n\}\ \forall j \in \{1, \dots, n\}$$

де $y_i = 1$, якщо ємність i використовується;

$x_{ij} = 1$, якщо предмет j поміщено в ємність i.

Таким чином, якщо у нас є ємності B, принаймні B – 1 ємність більш ніж наполовину заповнена. Тому

$$\sum_{i=1}^{n} a_i > \frac{B-1}{2} V$$

$$\frac{\sum_{i=1}^{n} a_i}{V}$$

є нижньою межею оптимального значення OPT, отримуємо, що $B - 1 < 2OPT$ і тому $B \leq 2OPT$

Кроки цього алгоритму:

– відсортуємо масив a по спадаючій;
– на i-ой ітерації береться елемент a_i і кладеться в безліч з мінімальною сумою на даний момент.

4.3 Архітектура

Цей розділ буде присвячений наступним питанням: як же її можна збирати, який з варіантів збору був реалізований і які точки розширення є в даній реалізації.

Отже, першим і основним питанням, на які треба відповісти: які можливості надає API Hadoop для збору статистики. Дані, необхідні для статистики, доступні з кінця Map стадії і до початку стадії Reduce. Розглянемо, де можна в цьому проміжку додати збір статистики:

– кінець стадії Map, на основі класу Mapper. Першим шляхом є спроба перехоплення пари (проміжний ключ, проміжне значення) під час їх запису в екземпляр класу Mapper.Context. Для цього треба було б створити клас, проксірующій клас Mapper.Context, за тим винятком, що метод write (KOUT key, VOUT value) ще б і оновлював статистику. Наступним кроком було б створення нового класу, що посяде Mapper і який передається в метод map (KEYIN key, VALUEIN value, Mapper.Context context), що не вихідний context, а його вищеописаний проксі. Цей підхід має той мінус, що для того, щоб випробувати нашу оптимізацію на вже існуючі програмі, необхідно вносити зміни в код, а саме міняти батьківський клас для Mapper;
– кінець стадії Map, на основі класу Partitioner. Це другий шлях, що дає можливість побудувати статистику в кінці Map стадії. На цьому шляху створюється абстрактний клас, що успадковує від Partitioner, який перед

викликом getPartition (KEY key, VALUE value, int k) буде оновлювати статистику. На жаль, цей метод також містить свої підводні камені: Partitioner не має методів життєвого циклу. Це може бути погано, якщо статистика буде зберігатися просто в файл, так як нам необхідно дізнаватися, коли Partitioner закінчив свою роботу, і можна виконати запис зібраної статистики в файлову систему;

– початок стадії Reduce, на основі класу Reducer. Останній варіант – це оновлювати статистику в спадкоємця класу Reducer перед тим, як викликати функцію reduce. Розглянемо його мінуси: по-перше, цей варіант має той же самий мінус, що і перший варіант, по-друге, додаткове ітерірованіє по всіх проміжних значень вкрай небезпечно, так як в разі їх великої кількості буде виконано багато дискових операцій, що може звести нанівець спроби оптимізації.

Для полегшення подальшого використання даної розробки було вирішено реалізувати збір статистики на основі Partitioner. (Для його використання необхідно прописати новий Partitioner в конфігураційних файлах, що довелося б робити в будь-якому випадку, так як ми замінюємо і сам алгоритм розподілу ключів). Наступним етапом після збору статистики є її зберігання, і на цьому етапі теж є два альтернативних шляхи:

– зберігати статистику в файлової системі;
– зберігати статистику в базі даних.

З причини того, що другий шлях набагато більш складніший, так як тягне в проект нові залежності і вимагає додатково розгортання сервера бази даних в кластері, і не має настільки ж очевидних і вагомих плюсів, було вирішено піти першим і найпростішим шляхом.

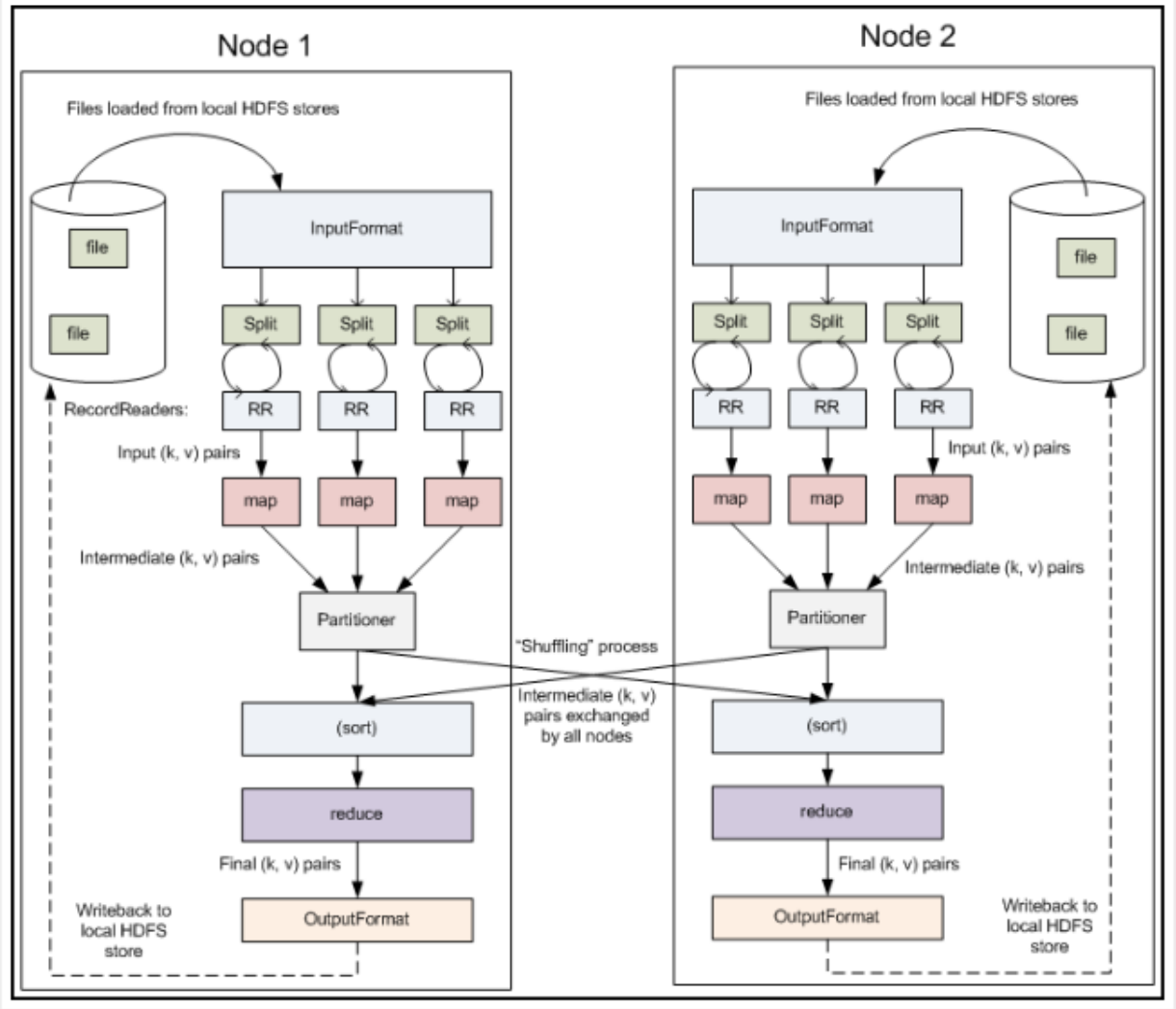

Рисунок 11 – Схема роботи алгоритму

Після того, як зроблений вибір на основі яких компонентів буде збиратися статистика, і де вона буде зберігатися, звернемо увагу на саму статистику. Очевидним і неприємним фактом є те, що зберігати її у вигляді пари проміжного ключа і кількості відповідних йому проміжних значень неможливо, в зв'язку з тим, що MapReduce обробляє величезні обсяги даних, і як наслідок:

– проміжних ключів може бути дуже багато;
– самі ключі можуть бути важкими, тобто займати багато місця на диску. Наприклад, проміжними ключами може бути довгі рядки.

Через ці проблеми доводиться об'єднувати ключі в групи, і розподіляти вже ці групи ключів по reduce машинам. І переходячи до програмної реалізації даної роботи, ця необхідність була врахована в інтерфейсі Statistic, який повинні реалізувати класи, що збирають і обробні статистику.

```
public interface Statistic<K,V,M> extends Writable {
    abstract Statistic<KEY, VALUE, M> newStatistic();
    void add(K key, V value);
    Converter<K, M> getConverter();
    Map<M, Integer> export();
}
interface Converter<K,M> {
    M convert(K key);
}
```

Statistic <K, V, M> правильно читати наступним чином: статистика приймаюча ключі класу K, значення класу V, і об'єднує ключі в групи класу M. Нижче буде описана стандартна реалізація цього інтерфейсу, але спочатку коротко опишемо, навіщо потрібні ті чи інші його методи. Зауважимо, що цей інтерфейс успадковує інтерфейс Writable пакета org.apache.hadoop.io, який містить методи, необхідні для сериализации об'єктів цього класу при збереженні його на диск або передачі по мережі. Повернемося до методів інтерфейсу Statistic:

– add – додає в статистику key і value;
– getConverter – повертає функцію, яка зіставляє ключу групу, в яку буде входити ключ. Ця функція повинна для одного і того ж ключа повертати одну і ту ж групу, так в іншому випадку пари з однаковим проміжним ключем можуть виявитися на різних reduce машинах. Так само треба мати на увазі, що ця функція буде викликана для кожної пари (проміжний ключ, проміжне значення);

– export – повертає Map, в якій кожній групі ключів порівнювати очікувану кількість проміжних значень, зіставлених всім проміжних ключам в цій групі. На основі цих даних і виконується алгоритм з частини 4.2.

Клас HashStatistic є реалізацією за замовчуванням інтерфейсу Statistic. Він групує ключі по хеш-коду взятому по модулю деякого числа N, яке сильно перевершує кількість машин.

На основі вище описаного алгоритму і інтерфейсу статистики, створений абстрактний клас StatistiscalPartitioner, який має один абстрактний метод:

```
Abstract Statisitic <KEY, VALUE, M> newStatisitc();
```

Клас StatisticalPartitioner за допомогою цього методу зчитує статистику з попередніх запусків і на її основі розподіляє проміжні ключі. HStatisticalPartitioner є спадкоємців StatisticalPartitioner, використовує клас HashStatistic, як реалізацію інтерфейсу Statistic.

HStatisticalPartitioner є найпростішим прикладом Partitioner, який працює на основі зібраної статистики розподілу проміжних значень.

5 ТЕСТУВАННЯ ТА АНАЛІЗ РОБОТИ РОЗРОБЛЕНОГО АЛГОРИТМУ

5.1 Конфігурація тестового стенду

Amazon EC2 офіційно підтримує Hadoop Project і надає офіційні публічні AMI для розгортання кластера, в зв'язку з чим було вирішено проводити експерименти на цьому майданчику. Крім того, на Amazon доступні публічні набори даних (Public Data Sets) з різних наукових областей таких як біонформатіка, алгебра, астрономія[14].

Щоб почати користуватися цими даними треба лише виконати пару команд. Ці набори знімають з дослідників завдання пошуку і складання тестових наборів для своїх досліджень. Один з них - Freebase Simple Topic Dump) буде активно використовуватися в тестах, описаних в 5.2 та 5.3.

Кластер під час проведення всіх експериментів складався з машин типу Small Instance, які мають 2 гігабайт оперативної пам'яті і один EC2 compute unit, який еквівалентний процесору 1.5 - 2.1 GHz 2009 opteron або процесору 2009 Xeon.

Наша реалізація заснована на API Hadoop версії 0.20.203. На жаль, офіційні образи Amazon містили лише застарілу версію 0.18.0, тому була використана публічна, але не офіційна AMI з номером ami-975390fe, яка містила Hadoop с API 0.20.203.

5.2 Word count

Класичним і одночасно найпростішим прикладом MapReduce програми є Word count, завданням якої є підрахунок кількості входження слова в даний текст. Це

завдання з’явилося як частина програм, які рахували TF-IDF [15] – важливість цього слова в документі, який є частиною декількох наборів документів.

Опишемо роботу цієї програми: на стадії Map на вхід приходить пара з номером рядка в тексті і самої строчки, функція map розбиває рядок на слова і видає проміжні пари у вигляді (слово, 1). На стадії reduce проміжні пари групуються по слову і всі одиниці підсумовуються, і результатом є пара (слово, кількість його входжень в текст).

Подальші результати наведені на тестах, де вхідними даними були 5000000 останніх рядків з Freebase Simple Topic Dump, які сумарно займали 319 мегабайт, програми виконувалися на машинах в вищеописаної конфігурації. Для збору статистики оптимізована версія спочатку запускалася на перших 1500000 рядках Freebase Simple Topic Dump.

Опишемо два типи гістограм, які будуть зустрічатися в цьому і подальших тестах:

– «Кількість проміжних значень», де на осі Y відображається кількість проміжних значень, а по осі X reducer-и. Цей тип зіставляє кожній reduce машині кількість проміжних значень, які вона опрацювала. Reducer-и на гістограмі відсортовані за спаданням кількості оброблених значень;
– «Час», де по осі Y – секунди, по осі X – reducer-и. Цей тип відображає скільки секунд виповнювалося reduce завдання на машині. Reducer-и на гістограмі відсортовані за спаданням часу виконання reduce завдання.

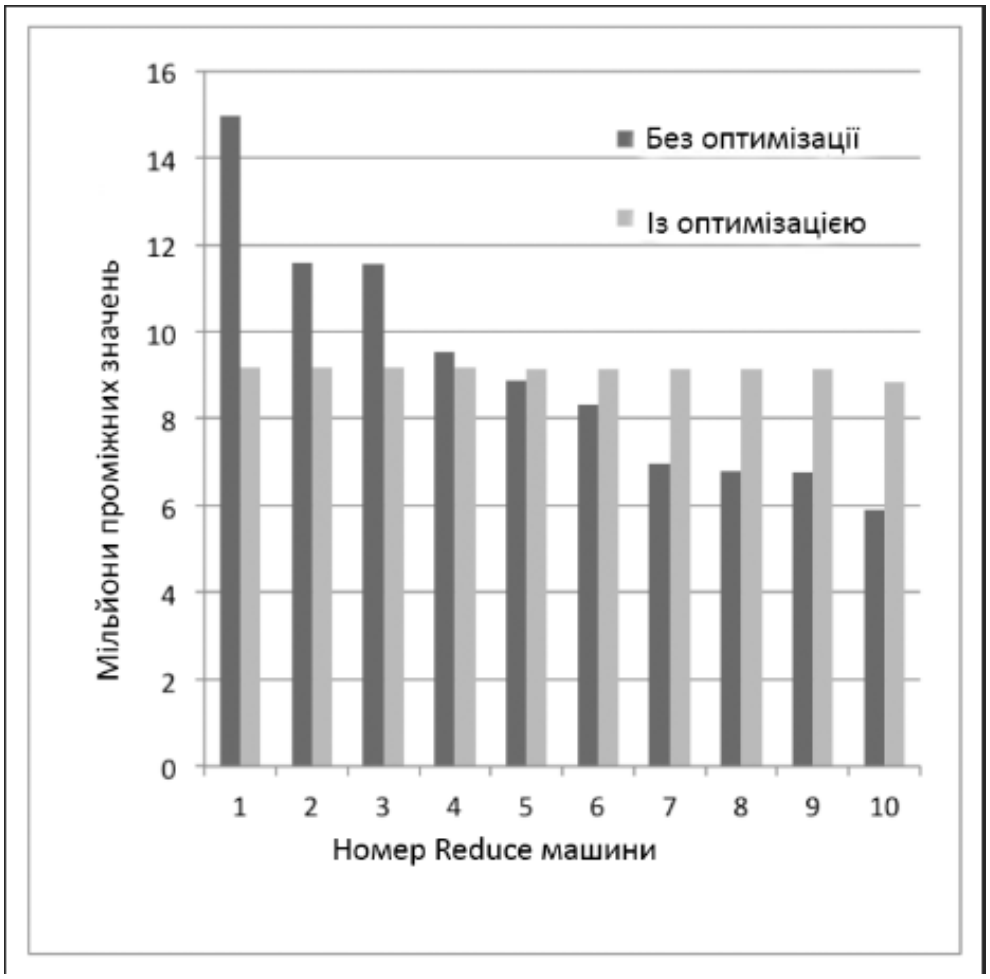

Рисунок 12 – Кількість проміжних
значень в експерименті з 10-ю завданнями reduce

З гістограми на рисунку 12, оптимізована версія дійсно краще розподілила проміжні ключі по reduce машинам: кількість оброблюваних проміжних значень на кожній з машин приблизно однаково, в той час як кількість оброблюваних проміжних значень в стандартній реалізації коливається набагато сильніше, і максимальну кількість перевищує мінімальну більш ніж в 2 рази. Але з гістограми на рисунку 13 видно, що на кінцевому часі виконанні стадії Reduce, і, як наслідок, MapReduce програми у цілому, це не позначається.

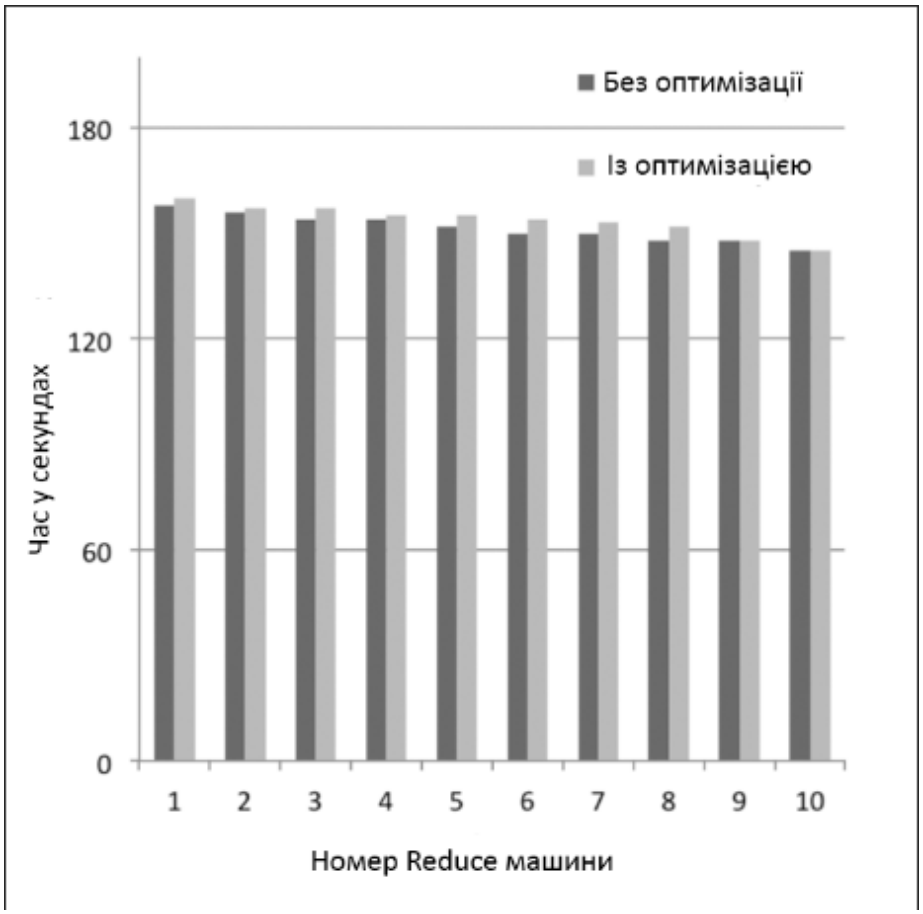

Рисунок 13 – Час у тесту із 10-ю завданнями reduce

Такі результати можна пояснити тим, що на стадії Reduce в Word count виконується лише процедура складання, тобто дуже проста операція. Наша ж оптимізація повинна працювати, коли на стадії Reduce виконується безліч операцій. Класичний тест Word Count є прикладом, коли наша оптимізація не потрібна.

5.3 First Character

У наступному експерименті проводиться підрахунок кількості слів, що починаються з кожної з букв. Програма була штучно уповільнена так, що кожні

500000 проміжних ключів обробляються задані користувачем t мілісекунд. Важність цього експерименту для нас полягає в тому, що, слова які починаються з різних букв алфавіту, вживаються з різною частотою: очевидно, що слова, які починаються з літери «x», набагато рідше вживаються, ніж слова, що починаються з «s».

Таким чином стандартний Partitioner повинен спрацювати досить погано, і кількість оброблюваних проміжних значень на різних машинах повинно сильно відрізнятися. І при збільшенні t ця різниця повинна значно відбитися на часі виконання відповідних Reduce завдань.

Тест проводився з наступними параметрами: програми обробляли перші 1500000 рядків з Freebase Simple Topic Dump, версія із статистичної оптимізацією, перед цим запускалася і збирала статистику з останніх 500000 рядків того ж дампа.

При малих t програма очікувано введе себе ідентично Word Count, тому діаграми з малим t не мають жодного інтересу для нас і не будуть приведені в даний роботі. Розглянемо діаграми на рисунку 14, 15, 16, 17, відповідні до запуску тесту при t = 30000 на восьми і десяти reduce машинах, відповідно. На цих діаграмах видно, що проміжні ключі, як і в випадку Word Count, стандартний Partitioner розподілив невдало, але на відміну від попереднього випадку це відбилося на часі виконання стадії Reduce. Як результат версія з оптимізацією, що розподіляє проміжні ключі на основі, виграє майже в два рази. А загальний час виконання програм:

– у разі восьми reducer-ів, без оптимізації дорівнює 12 хвилин 46 секунд, із оптимізацією - 8 хвилин 5 секунд;
– в разі десяти reducer-ів, без оптимізації - 11 хвилин 40 секунд, із оптимізацією - 6 хвилин 40 секунд;

Також можна зазначити, що при збільшенні кількості Reducer загальний час виконання програм очікувано знизився.

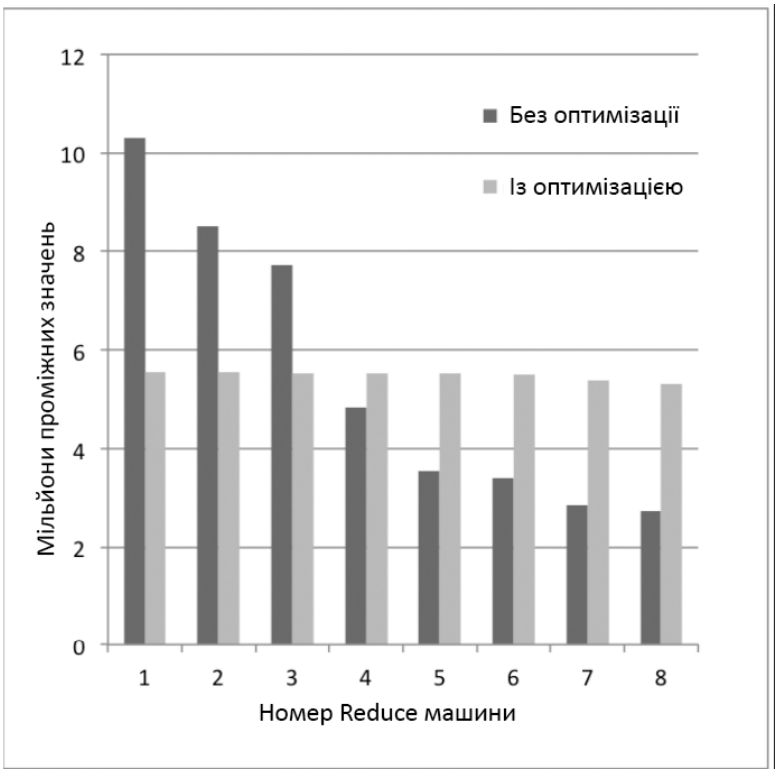

Рисунок 14 – Кількість проміжних значень в тесті з 8-ю завданнями reduce

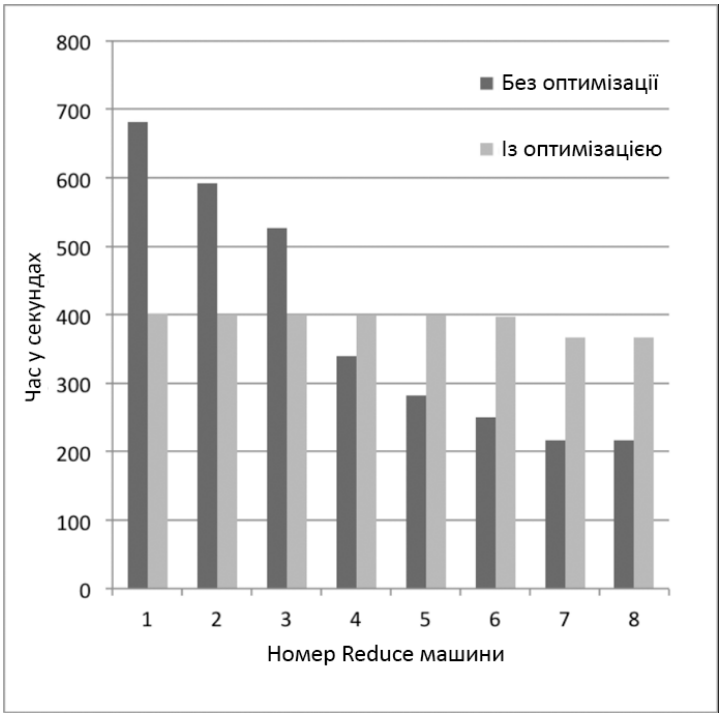

Рисунок 15 – Час у тесті із 8-ю завданнями reduce

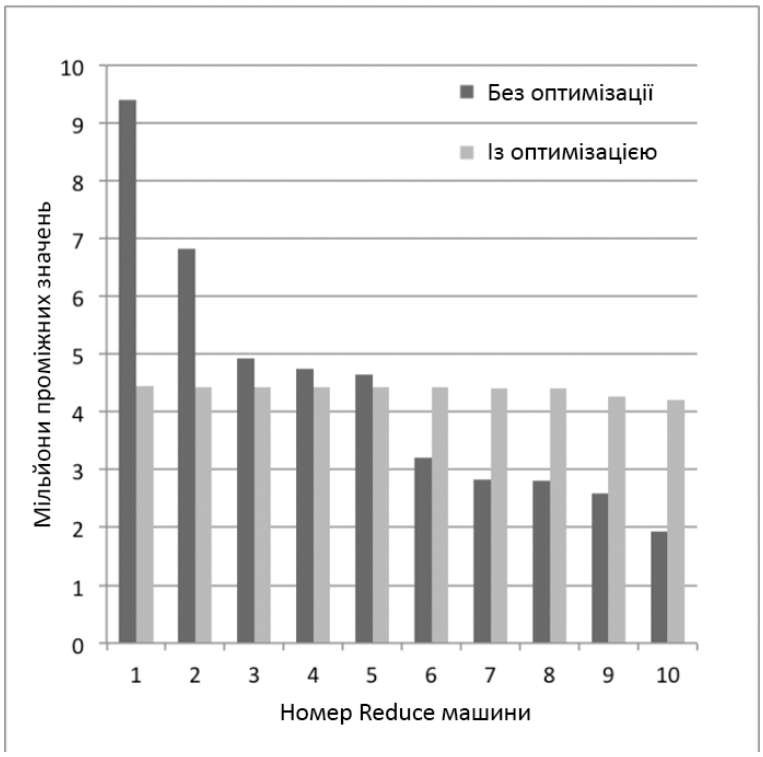

Рисунок 16 – Кількість проміжних значень в експерименті з 10-ю завданнями reduce

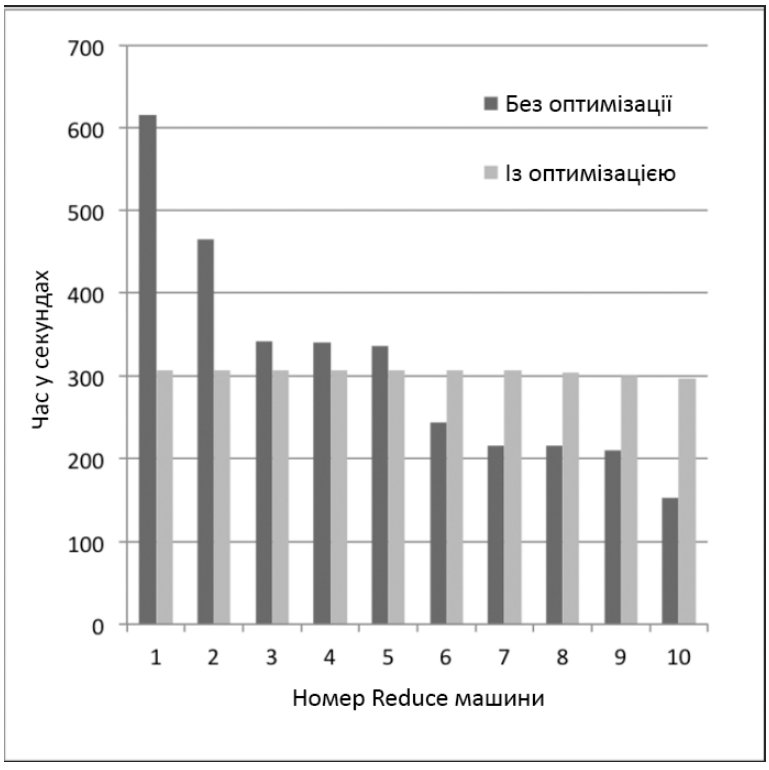

Рисунок 17 – Час у час тесту із 10-ю завданнями reduce

На даному штучному тесті наша оптимізація дала дуже гарні результати відносно не оптимізованої версії, трохи менше 50%. З цього тесту можна зробити висновок, що в разі нерівномірно розподілених проміжних ключів і довгій стадії Reduce в більш життєвих MapReduce програмах можна очікувати поліпшення від даної оптимізації.

5.4 Середня довжина сесії

В останньому експерименті буде розглянуто одне з класичних застосувань MapReduce технології – завдання аналізу логів. У нашому випадку буде проведено аналіз географії та поведінки користувачів сайту, а точніше нас буде цікавити наступна статистика: скільки в середньому сторінок за одну сесію переглядають користувачі кожної з країн, наприклад із України або користувачі з Італії і так далі.

Коротко опишемо підхід до її вирішення: на вхід в програму приходять логи запитів до сайту за певний період часу, на стадії Map кожному запиту підставляється країна користувача за допомогою ip адреси, що знаходиться в журналі. Таким чином результатом стадії Map є пара (країна, запит). На стадії Reduce запити, згруповані по країнам, розбиваються на сесії, і підраховується відношення запитів сторінок до кількості сесій, це саме те що нам потрібно. Більш докладний опис реалізації, представленої в даній роботі, буде наведено трохи пізніше в цьому розділі.

Даний тест у великій мірі залежить від того, в якому форматі і яка інформація міститься у вхідних даних. Реалізації в даній роботі обробляє логи з сайту проекту GanttProject [16], що приходять в форматі Combined Logs для Apache Server 1.3. Ці логи не містять cookies для простого визначення сесії запитів до сервера, тому сесія визначається за допомогою IP адреси, User Agent і часу запиту. Сесії вважаються

різними, якщо між двома запитами з одним і тим же IP адресою і User Agent пройшло півгодини.

Для того, щоб ефективно визначати чи належать запити однієї сесії, вони повинні бути відсортовані за часом до початку стадії Reduce. Цього можна досягти, зробивши час вторинним ключем і та відсортувавши згруповані проміжні результати за значенням вторинного ключа. Такий прийом називається Secondary Sort. Його наявність в даній реалізації додає цінності цього експерименту, так як для використання Secondary Sort в Hadoop потрібно змінити поведінку Partitioner, і в теорії накладення двох змін в Partitioner могло б привести до проблем. Але як буде далі показано в цій частині, використовувати StatisticalPartitioner можливо і для Secondary Sort.

Тести проводилися на логах запитів до сайту проекту GanttProject за березень 2018 року. версія з оптимізацією перед запусками мала зібрану статистику за перші 9 днів січня 2018 року.

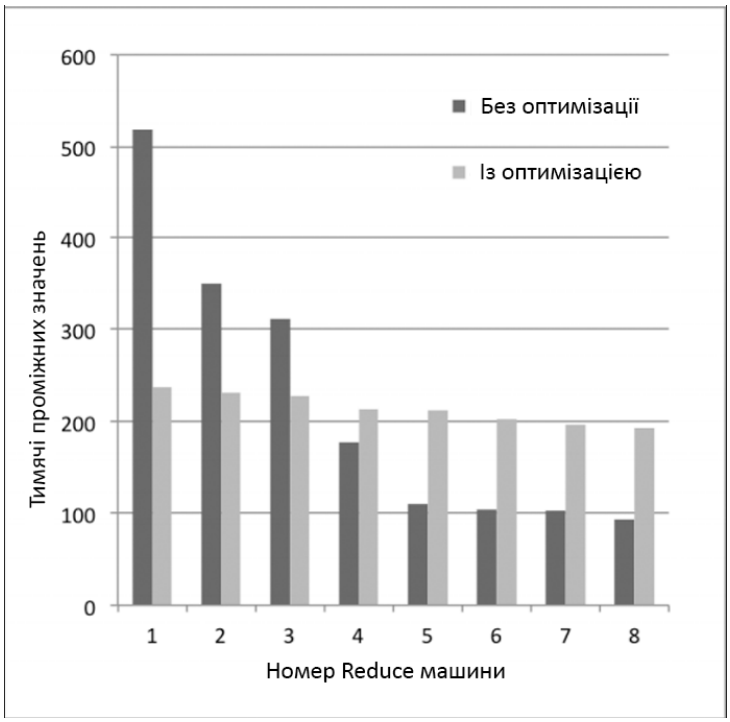

Рисунок 18 – Кількість проміжних
значень в експерименті з 8-ю завданнями reduce

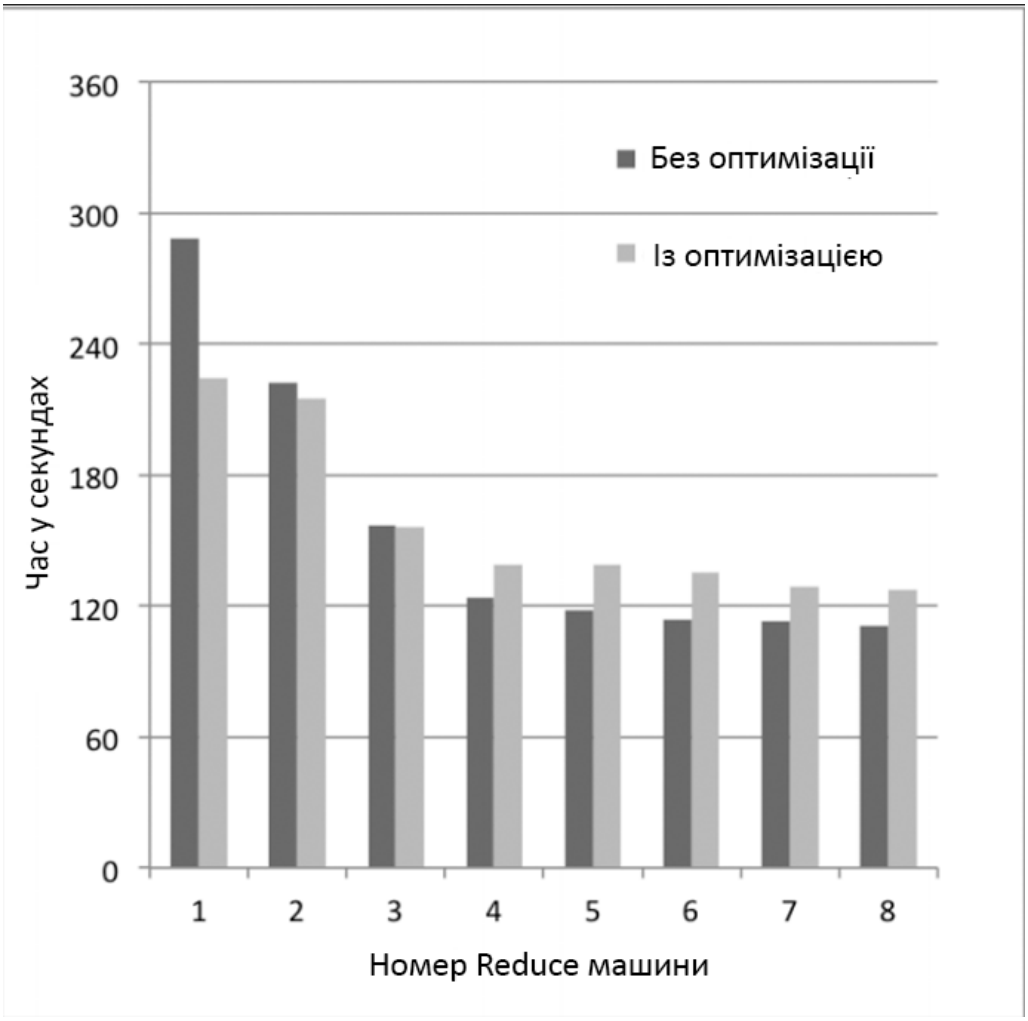

Рисунок 19 – Час у час тесту із 8-ю завданнями reduce

Розглянемо гістограми, зображені на рисунку 19 і рисунку 21. На них очевидний виграш оптимізованої версії, різниця між найдовшими reduce завданнями в оптимізованої і не оптимізованої версіями більше хвилини, при загальній довжині в 5 хвилин, то є виграш близько 20%. Але в цих результатах можна помітити ще цікаві факти. Наприклад, в неоптимізованої версії стадія Reduce на 10 машинах виконувалася довше, ніж на 8.

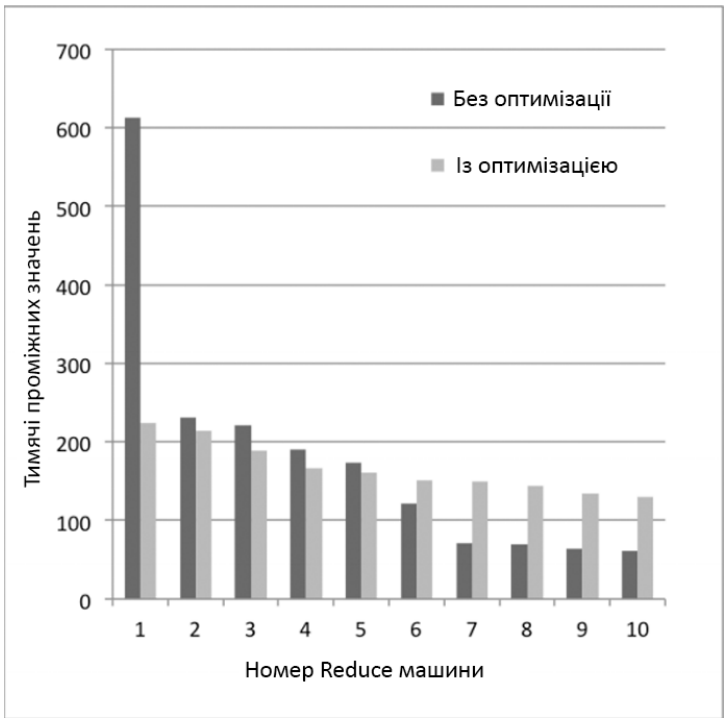

Рисунок 20 – Кількість проміжних значень в експерименті з 10-ю завданнями reduce

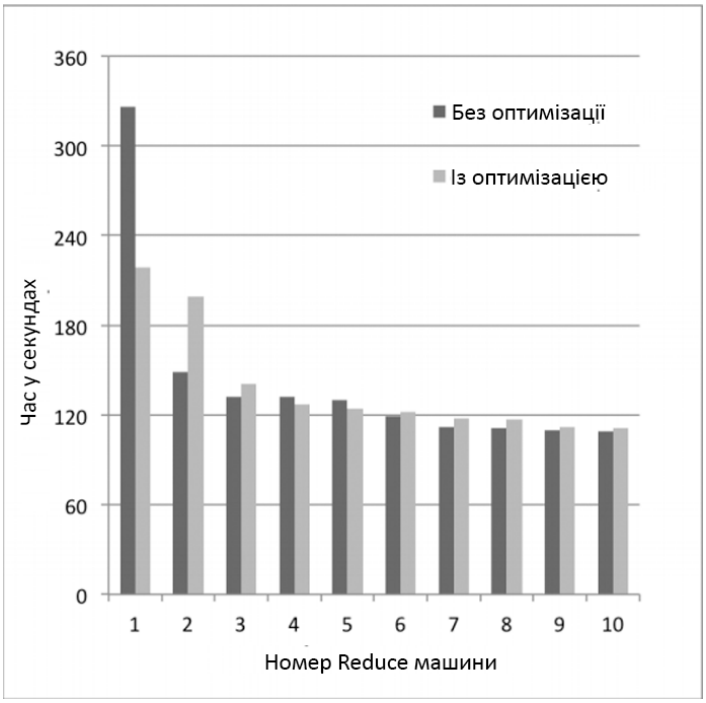

Рисунок 21 – Час у час тесту із 10-ю завданнями reduce

Якщо поглянути на гістограми на рисунку 18 і рисунку 20, картина прояснюється: для 10 reducer стандартний Partitioner розподілив ключі дуже невдало, на одному з reducer виявилося сильно більше проміжних значень для обробки. Також із тестів можна побачити, що не відбулося поліпшення швидкості виконання стадії Reduce між запусками на 8 і 10 reducer-ах і в оптимізованої версії. Можна сказати що існуючі програми обробки Big Data не дозволяють контролювати етапи введення даних, збирати статистику і підбирати оптимальні структури для зберігання індексів, оптимізувати розміщення даних на диску для забезпечення високої швидкості. Статистична оптимізація показала хороші результати в разі нерівномірного розподілу проміжних значень за проміжними ключам і Reduce стадії, яка виконує велику кількість операцій.

ВИСНОВКИ

В ході виконання роботи була поставлена задача дослідити и проаналізувати методи та технології роботи із Big Data, зокрема парадигми MapReduce.

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. Проаналізовано основні компоненти Apache Hadoop, які використовувалися у роботі. Що стосується методів аналізу для обробки Big Data, існуючі на сьогодні інструменти і найбільш поширені методи аналізу масивів даних поки не повністю задовольняють вимогам додатків обробки Big Data. В одному випадку вони не придатні для обробки великих даних, в іншому – важко їх застосовність при побудові автоматичної класифікації безлічі об'єктів в умовах відсутності апріорної інформації про кількість класів, в третьому – алгоритм має високу трудомісткість.

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 я проаналізував алгоритм та вдосконалив його роботу, провів математичний аналіз роботи алгоритму для обробки великих даних. Далі побудував архітектуру роботи алгоритму розписавши дії кожного класу та методів.

У п’ятому розділі проводилось тестування розробленого алгоритму на підібраних задачах, щоб перевірити роботу алгоритму. Можна сказати, що для паралельної обробки великої кількості неструктурованих даних часто використовують модель обчислень MapReduce. Одним з її недоліків є нерівномірне навантаження на reducer-и через невдалий розподілу проміжних ключів. В даному дослідженні розглядалася оптимізація, що розподіляє проміжні ключі на підставі даних, отриманих на попередніх запусках цієї MapReduce програми. Ця оптимізація була випробувана на деяких життєвих завданнях, і частина з цих завдань була істотно прискорена. Статистична оптимізація показала хороші результати в разі нерівномірного розподілу проміжних значень за проміжними ключам і Reduce стадії, яка виконує велику кількість операцій.

Можна сказати що існуючі програми обробки Big Data не дозволяють контролювати етапи введення даних, збирати статистику і підбирати оптимальні структури для зберігання індексів, оптимізувати розміщення даних на диску для забезпечення високої швидкості введення / виводу, для виконання аналітичних запитів немає можливості провести глибокий статистичний аналіз і виробити оптимальний план виконання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Stonebraker M., Abadi D., DeWitt D. J. et al. MapReduce and parallel DBMSs: friends or foes? // Commun. ACM. 2010. — . Vol. 53, no. 1. Pp. 64–71. URL: http://doi.acm.org/10.1145/1629175.1629197.

2. Dean J., Ghemawat S. Mapreduce: Simplified data processing on large clusters // OSDI. 2004. Pp. 137–150.

3. MapReduce. URL: http://en.wikipedia.org/wiki/Partition_ problem (дата звернення: 4.12.2019).

4. Managing Optimizer Statistics. URL: http://docs.oracle.com/ cd/B13789_01/server.101/b10752/stats.htm (дата звернення: 4.12.2019).

5. Jahani E., Cafarella M. J., R´e C. Automatic Optimization for MapReduce Programs // CoRR. 2011. Vol. abs/1104.3217.

6. Afrati F. N., Ullman J. D. Optimizing Joins in a Map-Reduce Environment: Technical report: National Technical University of Athens, Stanford University, 2009. — December. URL: http://ilpubs.stanford.edu:8090/952/.

7. Hadoop Project. URL: http://hadoop.apache.org/ (дата звернення: 4.12.2019).

8. Vance A. Hadoop, a Free Software Program, Finds Uses Beyond Search. URL: http://www.nytimes.com/2009/03/17/technology/ business-computing/17cloud.html?_r=1 (дата звернення: 4.12.2019).

9. Distributed File System. URL: http://ru.wikipedia.org/wiki/ Distributed_File_System (дата звернення: 5.12.2019). 31

10. Partition problem. URL: http://en.wikipedia.org/wiki/Partition_ problem (дата звернення: 4.12.2019).

11. Korf R. E. A complete anytime algorithm for number partitioning // Artificial Intelligence. 1998. Vol. 106. Pp. 181–203.

12. Mertens S. A complete anytime algorithm for balanced number partitioning // CoRR. 1999. Vol. cs.DS/9903011.

13. Zulawinski B. W., Iii W. F. P., Goodman E. D. The grouping genetic algorithm (gga) applied to the bin balancing problem.

14. Public Data Sets. URL: http://aws.amazon.com/publicdatasets/ (дата обращения: 20.05.2012).

15. Tf-idf weighting. URL: http://nlp.stanford.edu/IR-book/html/ htmledition/tf-idf-weighting-1.html (дата звернення: 6.12.2019).

16. GanttProject. URL: http://www.ganttproject.biz/ (дата звернення: 4.12.2019).

17. Фаулер Рефакторинг. Улучшение существующего кода. 2010. Pp. 352 – 353.

18. Gruzdo I.V., Torbiievskyi O., Shyrokopetlieva M.. METHODS OF STATISTICAL DATA ANALYSIS AND TOOLS FOR THEIR IMPLEMENTATION // SCIENCE, RESEARCH, DEVELOPMENT.Technics and technology.#15, 30.03.2019 – 31.03.2019 Rotterdam (The Netherlands). Warszawa, 2019. – 128 str. – p.p.79-97.