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 7.050102, 8.050102 Інформаційні технології проектування
7.050103, 8.050103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2015 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Качко Микиті Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)) Прикладні високопродуктивні кластерніобчислення в HADOOP
керівник проекту (роботи))Харченко Костянтин Васильович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02» квітня 2015 р. № 30/1-ст

2. Строк подання студентом проекту (роботи) 08.06.2015

3. Вихідні дані до проекту (роботи)

Використовувати віртуальну машину CentOSз встановленим фреймворком Hadoop
Реалізувати програмне забезпечення під Hadoop для розв'язування трьох типів задач: генерація простих чисел, знаходження чисел Серпінського та задача комівояжера.
Для порівняння написати програму що не розподіляє дані та виконує прямий алгоритм для кожної задачі.

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

1. Розглянути існуючі системи розподілених обчислень.
2. Дослідитиматематичні основи роботи.
3. Дослідити можливі варіанти реалізації архітектури додатку.
4. Описати використання програмного додатку.
5. Розробити опис задач для тестування.
6. Протестувати програмний додаток та порівняти дві концепції вирішення поставленої задачі.
7. Зробити висновки щодо перспективності використання фреймворку HADOOPв прикладних програмних додатках.

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

1. UML-діаграма класів (архітектура) додатку – плакат.
2. Таблиці порівняння часу виконання програм обох концепцій – плакат.
3. Графікипорівняння часу виконання програм обох концепцій – плакат.
4. Схема запропонованого розподілення даних між вузлами– плакат.

6. Консультанти розділів проекту (роботи)*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	ГусєвА.М., доцент		
Основна частина			

7. Дата видачі завдання01.02.2015

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2015	
2	Збір інформації	15.02.2015	
3	Вивчення варіантів реалізації та вибір варіанту для розробки	28.02.2015	
4	Розробка алгоритму та структури програми	10.03.2015	
5	Розробка плану оцінювання	15.03.2015	
6	Розробка програмної моделі	25.03.2015	
7	Розробка опису існуючих продуктів	25.04.2015	
8	Тестування додатку та отримання даних	30.04.2015	
9	Оформлення дипломної роботи	31.05.2015	
10	Отримання допуску до захисту та подача роботи в ДЕК	08.06.2015	

Студент _____ М.А.Качко
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ К.В.Харченко
(підпис) (ініціали, прізвище)

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

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)

"___"_____2015 р.

Дипломна робота

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

зі спеціальності7.050102, 8.050102 Інформаційні технології проектування
7.050103, 8.050103 Системне проектування
(код та назва спеціальності)

на тему: Прикладні високопродуктивні кластерніобчислення в HADOOP____
_____
_____

Виконав: студент 4 курсу, групи ДА-12
(шифр групи)

Качко Микита Андрійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцентХарченко К.В. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант охорона праці доцентГусєв А.М. _____ __ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль ст.. викладач Британов О.А.

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

Київ – 2015 року

АНОТАЦІЯ

бакалаврської дипломної роботи Качко Микити Андрійовича
на тему: «Прикладні високопродуктивні кластерні обчислення в HADOOP»

Дипломна робота присвячена використанню програмного продукту HADOOP для прискорення обчислення об'ємних завдань. У роботі наведено аналіз існуючих алгоритмів для тестування числа на предмет простоти. Досліджено час роботи програми без і з допомогою системи HADOOP на прикладі задачі генерації простих чисел.

Актуальність роботи полягає в тому що розподіл обчислень на кілька комп'ютерів зменшить час і зусилля, необхідні для виконання завдання.

В якості результату розроблено програмне забезпечення з підтримкою платформи Hadoop. Для порівняння було розроблено програмне забезпечення але з іншою, прямою структурою. Був запропонований інноваційний алгоритм складання таблиці простих чисел і спосіб розподілу завдань.

Шляхи подальшого розвитку предмета дослідження - використання більшої кількості більш потужних систем для складних обчислень.

Загальний обсяг роботи: 74 сторінки, 21 рисунок, 5 таблиць, 1 додатокна 3 стр., 31 посилання.

Ключові слова: РОЗПОДІЛ ОБЧИСЛЕНЬ, АЛГОРИТМ, HADOOP, ПРОСТІ ЧИСЛА, ЧИСЛА СЕРПІНСЬКОГО.

АННОТАЦИЯ

бакалаврскойдипломнойработыКачкоНикитаАндреевича
на тему:« Прикладные высокопроизводительные кластерные вычисления в HADOOP»

Дипломная работа посвященаиспользованию программного продуктаHADOOP для ускорения вычисления объемных задач.В работе приведен анализ существующих алгоритмов для тестирования числа на предмет простоты. Исследовано время работы программы без и с помощью системы HADOOP на примере задачи генерации простых чисел.

Актуальность работы состоит в том что распределение вычислений на несколько компьютеров уменьшит время и усилия, необходимые для выполнения задачи.

В качестве результата разработано программное обеспечение с поддержкой платформы Hadoop. Для сравнения было разработано другое программное обеспечение но с другой, обыкновенной структурой. Был предложен инновационный алгоритм составления таблицы простых чисел и способ распределения задач.

Пути дальнейшего развития предмета исследования - использование большего количества более мощных систем для сложных вычислений.

Общий объем работы: 74 страницы, 21рисунок, 5 таблиц, 1приложениена 3 стр., 31 библиографическое наименование.

Ключевые слова:РАСПРЕДЕЛЕНИЕ ВЫЧИСЛЕНИЙ, АЛГОРИТМ, HADOOP, ПРОСТЫЕ ЧИСЛА, ЧИСЛА СЕРПИНСКОГО.

ABSTRACT

Bachelor thesis Kachko Nikita A.

on«Applied high-performance cluster computing on HADOOP»

Thesis is devoted to the use ofHADOOP software to speed up the complex tasks. The paper provides an analysis of existing algorithms to test number for the primeness. The time of execution of the program with or without the help ofHADOOP via an example of the problem generating prime numbershas been researched.

Relevance of the work is that the distribution of computation on several computers will reduce the time and resourses required to complete the task.

As a result the software was developed with the support of the platform Hadoop. For comparison, we developed other software but on the other, the straightforward structure. Innovative algorithm of generating prime tables and way of distribution of tasks were proposed.

Through further development of the research subject - using more powerful systems for more complex tasks.

The total amount of work: 74 pages, 21 Figures, 5 tables, 1 appendix for 3 p., 31 references.

Keywords: DISTRIBUTED COMPUTING, ALGORITHM, HADOOP, PRIME NUMBERS,SIERPINSKI NUMBERS.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

ІТ– інформаційні технології

ОС – операцыйна система

ПП – програмний продукт

СКІТ–суперкомп’ютерний комплекс інституту кібернетики

СЛАР–система алгебраїчних лінійних рівнянь

ACID – atomicity, consistency, isolation, durability

BOINC–berkeley open infrastructure for network computing

GPU–graphics processing unit

HDFS –hadoop distributed file system

VDS– virtual disk service

SMP – shared-memory multiprocessing

ВСТУП

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

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, що 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.

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.

Другий розділ присвячено дослідженню існуючих методів для розв’язання проблеми, побудовано математичну модель, описано за якими критеріями визначалась якість результату роботи системи, наведено алгоритм роботи програми для тестуваня даної системи.

У третьому розділі було обґрунтовано вибір платформи та мови реалізації програмного продукту, було зроблено аналіз архітектури системи. Крім того, були розроблені експерименти за допомогою яких можна було проводити аналіз результатів, отриманих в роботі.

У четвертому розділі проводиться аналіз отриманих результатів, а саме порівняння двох концепцій програмних додатків – з та без підтримання фреймворку HADOOP.

П’ятий розділприсвячено охороні праці. У ньому проведено аналіз умов праці, аналіз шкідливих та небезпечних чинників, з якими стикається робітник під час роботи. В результаті було описано, що потрібно зробити для того, щоб дане приміщення відповідало необхідним нормам і було комфортним для робітника.

1. ДОСЛІДЖЕННЯ ЗАДАЧІ РОЗПОДІЛЕННЯ ОБЧИСЛЕНЬ

1.1 Актуальність задачі

Проекти розподілених обчислень (Distributed Computing) мають вагому історію та серйозний обчислювальний потенціал. Сьогодні власникам персональних комп'ютерів пропонують взяти участь у рішенні найрізноманітніших науково-дослідних задач. Найчастіше до технології розподілених обчислень дослідники звертаються тоді, коли наукові проекти вимагають величезної кількості обчислень, які недоцільно та надто дорого виконувати за допомогою суперкомп'ютерів або кластерних обчислювальних мереж. Особливисть таких проектів полягає в тому що вони вирішують проблему одночасного моделювання громіздких систем.

Приклад: моделювання фолдінга (згортання в тривимірну структуру) важливих білків людського організму. Це завдання становить найбільш складну проблему молекулярної динаміки білків, тому що вирішальна стадія фолдінга білка відбувається практично миттєво при взаємному впливі безлічі факторів. Вирішення подібного завдання, у вигляді відпрацьовування методів моделювання на експериментально підтверджених моделях згортання білків, відкриває перспективи дослідження першопричин порушень цього процесу, що призводять до багатьох важких захворювань. Ці дослідження зрештою повинні призвести до розробки нових ліків і способів лікування багатьох форм рака, хвороб Альцгеймера, Паркінсона та ін. [1]

Приклад 2: при аналізі сучасних складних математичних моделей, що описуються системами диференційних рівнянь з частинними похідними виникаєпотреба в значних обчислювальних ресурсах для роботи з розрідженими системамилінійних алгебраїчних рівнянь (СЛАР). Зокрема, вирішення задач тепловогопроектування електронних пристроїв складної

конструкції методами скінченнихелементів чи скінченних різниць зводиться до розв’язування СЛАР з розрідженимиматрицями надвеликих розмірностей.

Розмірність таких систем в окремих випадках є настільки великою, що їхрозв’язання ускладнюється або й стає неможливим без використання спеціальнихобчислювальних ресурсів та засобів, таких як обчислювальні кластери аборозподілені системи. Саме системи розподілення обчислень стали останнім часомособливо використовуваними. Така популярність обумовлена, в першу чергу:

- зростанням потреб у вирішенні обчислювальних завдань високої складності;
- стрімким підвищення швидкодії систем розподілення обчислень;
- на порядок нижчою собівартістю таких систем в порівнянні з суперкомп’ютерами.

Розробка методів та засобів розподілення обчислень, що виконуються при розв’язуванні СЛАР великої розмірності дасть змогу, завдяки ефективномувикористанню наявної обчислювальної інфраструктури: по-перше, зменшити часвирішення задач теплового проектування електронних пристроїв та, по-друге,скоротити обсяг необхідних фінансових ресурсів.

Таким чином, розробка методів та засобів, які дали б можливістьрозподіленого вирішення задач теплового проектування електронних пристроїв, щопотребують розв’язування СЛАР великої розмірності є актуальною науковоюзадачею.[2]

Ще одна проблема: практично всім науковим організаціям недостатньо фінансування для придбання спеціальних ресурсів - суперкомп'ютерів або кластерних обчислювальних мереж - для проведення ґрунтовних наукових досліджень. Розподілені обчислення, що проводяться за допомогою групи добровільних користувачів Internet, співставні і навіть перевершують потужності суперкомп'ютерів і тому можуть стати альтернативою на окремих етапах актуальних наукових досліджень.[3]

Розглянемо застосування схем на нашій Батьківщині

Завдяки використанню суперкомп'ютерів науково-дослідні установи НАН України отримали важливі фундаментальні і прикладні результати збіофізики, біохімії, фізичної хімії, теоретичної фізики, матеріалознавства,медицини, геології/геофізики, нанотехнологій тощо. В рамках виконаннятематичних планів досліджень, програм держзамовлень, участі у вітчизняних таміжнародних наукових проектах українські науковці постійно застосовуютьсуперкомп'ютерний комплекс СКІТ Інституту кібернетики (ІК)ім.В.М.Глушкова НАН України для розв'язання задач моделюваннямолекулярної динаміки, квантово-хімічних розрахунків, моделюваннятеплофізичних, гідродинамічних і геофізичних процесів. На суперкомп'ютерахпроводяться розрахунки властивостей хімічних сполук, сплавів, біологічноактивних речовин і нанооб'єктів, обробка супутникових зображень ірезультатів сейсмогеологічних досліджень, медичні популяційні дослідження ідослідження геному, моделювання фізичних і соціально-економічних процесів.

Суперкомп'ютерні обчислення сполучуються грід-обчисленнями, хоча не заміняють одне іншого. Грід – це інфраструктура розподілених обчислень, коли задача поділяється на кілька окремих підзадач, підзадачі розв'язуються незалежно, навіть не одночасно, їх результати записуються в файли, а вже потім об'єднуються. Суперкомп'ютер спрямований на паралельні обчислення,тобто підзадачі виконуються одночасно і взаємодіють через передачуповідомлень з даними. Взагалі Грід може використовувати і використовуєсуперкомп'ютери як вузли. Але для багатьох задач великої розмірності відсутніалгоритми поділу на незалежні підзадачі, а умови стійкості обмежують розміри елементарних одиниць, отже, і обсяги необхідної пам'яті. Такі задачі взагалінеможливо розв'язати без суперкомп'ютера.

Можливості сучасних суперкомп’ютерів дозволяють створитиінформаційні технології, які з позиції системного (комплексного) аналізу
та оптимізації розкривають сутність природно взаємопов’язаних досліджуванихявищ. Внаслідок розвитку суперкомп’ютерних технологій вартість і часчисельного моделювання стали нижчими, ніж в разі експериментальноговипробування реальних об’єктів. Сучасні ІТ та суперкомп’ютери, якінструменти розв’язання класів надскладних задач стали одним з визначальнихчинників росту конкурентоспроможності економіки і безпосередньо впливаютьна спроможність країни забезпечувати потреби населення та суб’єктівгосподарювання у інформаційному суспільстві.

Створення суперкомп’ютерних ІТ потребує розробки математичних моделей, ефективних методів їх аналізу, алгоритмів та надійних програм.Складність комплексних взаємозв’язків предметної області та зростаючівимоги до ефективності паралельної реалізації сприяють успіхуінтелектуальних ІТ, здатних самостійно налаштовуватись на властивості задачіта обчислювального середовища. Фундаментальна наукова складова цієїтематики полягає в дослідженні і розробці ефективних математичних методівта алгоритмів розв’язання класів задач, що за вимогами до об’єму пам’яті таобчислювальної продуктивності перевищують можливості доступних ЕОМ, аждо задач трансобчислювальної складності.

Подальший прогрес застосування обчислень у фундаментальній та прикладній науковій діяльності НАН України потребує розвиткусуперкомп’ютерних потужностей. Кластерний комплекс СКІТ, який єнайбільшим обчислювальним ресурсом НАН України та основою ресурсного центру Українського національного Гріду (УНГ), з 2005 р. забезпечує набезоплатній основі високопродуктивні обчислення організацій та установ НАНУкраїни, закладів МОН і НКА. Три покоління суперкомп’ютерів СКІТ (СКІТ-1,СКІТ-2, СКІТ-3) розвинуто та успішно застосовано у дослідженнях за

програмою «Інтелект» протягом 2007-2009 рр., а за 2010-2011 рр. Впровадженогібридний сегмент СКІТ-GPU для експериментальних досліджень новихпідходів до розпаралелювання задач. На його основі у 2012 р. розроблено тареалізовано у суперкомп'ютері СКІТ-4 новий комплексний архітектурнийпроект побудови кластерних обчислювальних систем, що дозволило досягтивдвічі більшої за суперкомп'ютер попереднього покоління СКІТ-3продуктивності - майже 12 Тфлопс. Крім того, це дозволило суттєво підвищитийого енергоефективність, електроспоживання якого (15 кВт-годин) вчетвероменше за СКІТ-3. СКІТ-4 за енергоефективністю відповідає 99 позиції усвітовому рейтингу найбільш екологічних суперкомп'ютерів (Green500).

Передбачається подальше нарощування його потужностей. Суперкомп'ютер СКІТ-4 зараз є найпотужнішим обчислювальнимзасобом України. Він підключений як складова частина до комплексу СКІТ, щоє основою Ресурсного центру Українського національного гріду, та пройшовсертифікацію Європейської грід-ініціативи (EGI).

Серед користувачів кластерного комплексу СКІТ найбільшу часткуресурсів у 2010–2012 рр. споживали:

1 – Інститут молекулярної біології і генетики (близько 30%);

2 – Інститут кібернетики ім. В.М. Глушкова (15%);

3 – Український національний грід (за 3 роки його частка зросла з 5,5 до 18%);

4 – Інститут фізики конденсованих систем;

5 – Фізико-технічний інститут низьких температур ім. Б.І. Вєркіна;

6 – Інститут математичних машин і систем;

7 – Інститут гідромеханіки;

8 – Ужгородський національний університет;

9 – Інститут органічної хімії;

10 – Запорізький інститут державного та муніципального управління.

Крім того, суперкомп’ютери СКІТ застосовували Інститут геохімії, мінералогії та рудоутворення ім. М.П. Семененка, Інститут металофізикиім. Г.В.Курдюмова, Донецький фізико-технічний інститут ім. О.О. Галкіна,Харківський національний університет ім. В.Н. Каразіна. Близько 15% обчислювальнихресурсів використано для розв’язання важливих державних задач (зовнішнярозвідка, державний бюджет, оптимізація обслуговування державного боргу).Задачі, які вирішують за допомогою суперкомп’ютерів, часто взагалінеможливо розв’язати в інший спосіб із потрібною точністю чи детальністю.Насамперед це стосується задач моделювання, де критичним параметром єрозмір сітки. Наприклад, дослідження довготривалого впливу забруднення наякість питної води неефективно проводити на окремій ділянці з огляду наневизначеність умов на краях моделі, якщо вони не відповідають природниммежам системи вододілу поверхневих і підземних потоків. Отже, масштабтакого дослідження – тисячі й десятки тисяч квадратних кілометрів, аглибинність – сотні метрів. Потужність водоносного шару – від кількох метрів,ширина більшості річок не перевищує 10–20 м. Тож йдеться про сітки змільйонів чи десятків мільйонів комірок. Обрахунок таких сіток потребує накожному часовому кроці розв’язку системи рівнянь відповідного розміру (кілька рівнянь на комірку). Використання суперкомп’ютера дає змогукардинально підвищити детальність моделі, а отже, і достовірність результатів.[4]

1.2 Особливості й проблематика задачі розподілення обчислень

Проаналізувавши ринок хмарних обчислень, можна сказати, що основна його маса зосереджена на продажу віртуальних машин в хмарі. Однак, хмари, це не тільки маркетинг. Існуючі технології тимчасового збільшення ліміту споживання ресурсу не здатні вирішити проблему динамічної зміни ресурсів. Тобто машина недоотримує ресурси тоді, коли вони їй найбільше потрібні.

Друга проблема - коли ресурси, фактично, довелося сплатити, але використовувати не вдалося. Віртуальній машині вночі просто не потрібно стільки ресурсів. Вона простоює.

Виникає проблема: людина змушена замовляти ресурсів більше, ніж потрібно в середньому, для того, щоб переживати без проблем піки. В інший час ресурси простоюють. Провайдер бачить, що сервер не навантажений, починає продавати ресурсів більше, ніж є (це називають «оверселл»). У якийсь момент, наприклад, через пік навантаження на декількох клієнтів, провайдер порушує свої зобов'язання. Він обіцяв 70 людям по 1 ГГц - але в нього є тільки 40 (2.5 * 16 ядер). Недобре. Продавати смугу чесно (без оверселла) невигідно (і ціни неринкові виходять). Оверселлити - знижувати якість сервісу, порушувати умови договору. Ця проблема не пов'язана з VDS або віртуалізацією, це загальне питання: як чесно продавати простоюють ресурси? [6]

Під час розподіленної обробки даних велика задача поділяється на кілька менших, які одночасно виконуються на кількох вузлах комп'ютерної системи. Відтак вихідна задача вирішується значно швидше. Розподіл даних системи за багатьма вузлах здійснюється для того, щоб дати можливість багатьом обчислювальним одиницям отримувати одночасний доступ до даних, розміщених на різних носіях. Цим забезпечується паралелізм під час виконання операцій обчислення/введення/виведення. Відомі два варіанти розміщення даних на багатьох елементах:

- розміщення на різних вузлах таблиць;
- розміщення на різних вузлах рядків однієї таблиці, або горизонтальний поділ таблиць.

Разом із цим постає проблема нерівномірного розподілу даних. Бажано, щоб кортежі відношення розподілялися за вузлами рівномірно, оскільки саме в такий спосіб досягатиметься рівномірний розподіл навантаження на елементи під час виконання операцій обчислення/введення/виведення. А це, в свою чергу, сприяє ефективнішому розподіленню. Однак можливі ситуації, коли одні

елементи містять багато даних, а інші — мало. Такий розподіл, звичайно, є небажаним. Нерівномірний розподіл може бути обумовлений такими факторами:

- нерівномірний розподіл значень атрибутів. Значення деяких атрибутів у відношенні можуть розподілятися нерівномірно. Наприклад, атрибут Зарплата взагалі не має значень поза межами певного діапазону, а всередині нього, скоріш за все, значення розподілені нерівномірно. Якщо такий атрибут використовується в хешуванні чи ранжуванні, то кортежі відношення будуть розподілятися за вузлами нерівномірно;
- нерівномірний розподіл значень у групах ранжування. Застосування методу ранжування за неправильного формування груп може призвести до нерівномірного розподілу кортежів.[5]

1.3 Приклад існуючих систем для вирішення задачі

На даний момент часу, ринок програмного забезпечення систем розподілених обчислень представлений великою кількістю різних програмних продуктів. Розглянемо деякі з них:

1.3.1 Folding@Home

Folding@Home – це проект розподілених обчислень для проведення комп’ютерного моделювання згортання молекул білка. Проект запущений 1 жовтня 2000 року ученими зі Стенфордського університету.

Мета проекту – за допомогою моделювання процесів згортання/розгортання молекул білка отримати краще розуміння причин виникнення хвороб, що викликаються дефектними білками, таких як хвороби Альцгеймера, Паркінсона, діабету II типу, хвороби Крейтцфельдта - Якоба (коров’ячий сказ), склероз і різних форм онкологічних захворювань. До теперішнього часу проект Folding@home успішно змоделював процес

згортання білкових молекул протягом 5-10 мкс – що в тисячі разів більше попередніх спроб моделювання.

За результатами експерименту опубліковано трохи менше 200 наукових робіт.

1.3.2 Rosetta@Home

Проект добровільних обчислень, спрямований на вирішення однієї з найбільших проблем в молекулярній біології – обчислення третинної структури білків з їх амінокислотних послідовностей. Завдяки недавно завершеному проекту «Геном людини» відомі амінокислотні послідовності всіх білків в людському організмі. Дослідження за даним проектом також допоможуть у проектуванні нових, неіснуючих білків. В разі успішного вирішення даних проблем людство зможе боротися з такими хворобами як рак, малярія, хвороба Альцгеймера, сибірська виразка та іншими генетичними і вірусними захворюваннями.

Результати обчислень Rosetta@Home не доступні безпосередньо. Так само, не можна використовувати результати обчислень власного комп'ютера. Однак вони використовуються для великої кількості наукових публікацій.

По суті Rosetta – це комп’ютерна програма, основними завданнями якої є:

- пошук структури з найменшою енергією для заданої амінокислотної послідовності для передбачення структури білка;
- вирішення оберненої задачі – пошук амінокислотної послідовності з найменшою енергією для заданої білкової структури;
- розрахунок взаємодії комплексу білок-білок.

В даному проекті використовується зворотній зв'язок з прогнозуванням та отриманим результатом, щоб покращувати потенційні функції і алгоритми пошуку.[7]

1.3.3Einstein@Home

Einstein @ Home — проект добровольчих обчислень на платформі BOINC по перевірці гіпотези Ейнштейна про існування гравітаційних хвиль, які досі нікому з учених так і не вдалося зафіксувати.

Проект стартував в рамках Всесвітнього року фізики 2005 (англ.) і координується університетом Вісконсіна-Мілуокі (англ.) (Мілуокі, США) і Інститутом гравітаційної фізики ім. Макса Планка (англ.) (Ганновер, Німеччина), керівник — Брюс Аллен (англ.).

З метою перевірки гіпотези проводиться складання атласу гравітаційних хвиль, випромінюваних неосесиметричними зірками, що швидко обертаються: нейтронними зірками (пульсарами), хитними зірками (англ. wobbling star), аккреціюючими (англ. accreting star) і пульсуючими зірками (англ. oscillating star).[8] Дані для аналізу надходять з Лазерно-інтерферометричної гравітаційно-хвильової обсерваторії (LIGO) і GEO600.

Крім перевірки загальної теорії відносності Ейнштейна та отримання відповідей на питання «Чи поширюються гравітаційні хвилі з швидкістю світла?» і «чим вони відрізняються від електромагнітних хвиль?»[9], пряме виявлення гравітаційних хвиль буде також являти собою важливий новий астрономічний інструмент (більшість нейтронних зірок не випромінюють в електромагнітному діапазоні та гравітаційні детектори здатні привести до відкриття цілої серії раніше невідомих нейтронних зірок[10]). Наявність же експериментальних доказів відсутності гравітаційних хвиль відомої амплітуди від відомих джерел поставить під сумнів саму загальну теорію відносності та розуміння сутності гравітації.

1.3.4PrimeGrid

PrimeGrid — проект добровільних розподілених обчислень на платформі BOINC і PRPNet, метою якого є пошук простих чисел різного виду. Проект стартував 12 червня 2005 року.

Основна мета проекту — нести задоволення пересічному користувачу від знаходження простих чисел. Другою метою PrimeGrid є надання відповідних навчальних матеріалів про прості числа.

І нарешті, прості числа грають центральну роль в криптографічних систем, які використовуються для комп'ютерної безпеки. Через вивчення простих чисел можна показати, скільки вимагається обчислень, щоб зламати код шифрування і таким чином визначити, чи є поточні схеми безпеки достатньо безпечними.

Участь у проекті PrimeGrid можлива у два способи: через BOINC і через Project Staging Area (PSA) [11]

1.4 Постановка задачі

Метою даної роботи було аналіз продуктивності програмного забезпечення HADOOP. Для реалізації цього завдання потрібно:

1. Дослідити та вивчити основні принципи роботи програм для розподілення обчислень.
2. Розробити прикладну задачу, на прикладі якої можна ефективно дослідити завдання розподілених обчислень і яка є гнучкою для реалізації на різних мовах програмування та різних операційних системах.
3. Зробити критичний огляд відомих програм для побудови паралельних систем.
4. Проаналізувати існуючі платформи 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.

1.5 Висновки

В даному розділі розглянуто та досліджено задачу побудови системи розподіленх обчисдень. В 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. МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1 Дослідження області простих чисел

За визначенням, просте число — це натуральне число, яке має рівно два різних натуральних дільники (лише 1 і саме число). Решту чисел, окрім одиниці, називають складеними[12]. Таким чином, всі натуральні числа, більші від одиниці, розбивають на прості і складені.

Існує багато алгоритмів, за допомогою яких можна перевірити число на простоту. Кожного року з'являються нові модифікації існуючих алгоритмів. Але серед всього різноманіття алгоритмів, можна виділити основні, які можна використати для перевірки. Їх можна розділити на такі категорії:

1. Наївні методи:
2. Ймовірністні тести.
3. Швидкі детерміновані тести.

Коротко опишемо кожен з них:

2.1.1 Наївні методи

Найпростіший тест простоти полягає в такому: коли задане число n, перевірити чи якесь ціле m від 2 до $n-1$ ділить n. Якщо n ділиться на певне m, то n складене, в іншому разі воно просте. Замість перевірки всіх m до $n-1$, досить лише перевірити m до $\sqrt{n}$: якщо n складене, то його можна розкласти на два множники, принаймні один з яких не перевищує $\sqrt{n}$. Можна також покращити ефективність, пропускаючи всі парні m , за винятком 2, бо коли якесь парне число ділить n , то 2 також ділить. Можна далі вдосконалити зауважуючи, що всі прості числа, за винятком 2 та 3, мають вигляд 6k ± 1. Дійсно, всі цілі можна подати як (6k + i) для деякого k та для i = -1, 0, 1, 2, 3, або 4; 2 ділить (6k + 0), (6k + 2), (6k + 4); а 3 ділить (6k + 3). Спочатку перевіряємо чи n ділиться на 2 або 3, тоді пробігаємо всі числа вигляду 6k ± 1≤ $\sqrt{n}$. Це у 3 рази швидше від попереднього методу. Насправді, всі прості мають вигляд $c\#k + i$ для $i<c\#$ де і

належить до чисел, взаємно простих з *c#* (приморіал) . Фактично, коли $c \to \infty$ кількість значень, які *c#k + i* може набувати в певному діапазоні, зменшується, а, отже, час тестування *n* зменшується. Для цього методу, слід ділити на всі прості менші ніж *c*. Спостереження, аналогічні до попереднього, можна застосувати рекурсивно, отримуючи решето Ератосфена [13]. Вдалим способом пришвидшення цих методів (*i* всіх інших згаданих далі) є попередній обрахунок і зберігання списку всіх простих до певної межі, скажімо всіх простих до 200. (Такий список можна обчислити за допомогою решета Ератосфена). Тоді, перед тестуванням *n* на простоту з використанням серйозного методу, спочатку перевіряємо чи *n* не ділиться на якесь просте із цього списку.

2.1.2 Ймовірністні тести

Найпопулярнішими тестами простоти є ймовірнісні тести. Ці тести використовують, крім тестованого числа n, деякі інші числа a , які випадково вибираються з певного набору; звичні рандомізовані тести простоти ніколи не оголошують прості числа складеними, але можливе для складених чисел оголошення їх простими. Імовірність помилки можна зменшити, повторюючи тест з ріними незалежно вибраними a; для двох найчастіше вживаних тестів, для будь-якого складеного n принаймні половина авизначає складеність *n* , тому *k* повторень зменшують імовірність помилки до щонайбільше 2^{-k}. Останню величину можна зробити як завгодно малою, збільшуючи *k*.

Концепцію побудови генетичних алгоритмів схематично можна представити таким чином [10]:

1. Випадково вибрати число a.
2. Перевірити певну рівність, що містить a та задане число n. Якщо рівність не виконується, то n є складене число, а називають свідком складеності, і тест зупиняється.
3. Виконувати крок 1, поки не буде досягнуто потрібної певності.

Після низки повторень, якщо не отримано, що *n* є складене число, то його можна оголосити імовірнісним простим.

Найпростішим ймовірнісним тестом простоти є тест простоти Ферма. Це лише евристичний тест; деякі складені числа (числа Кармайкла) будуть оголошені "ймовірнісними простими" незалежно від того, якого свідка обрати. Проте, він деколи використовується з метою швидкої перевірки числа, наприклад, на фазі утворення ключа криптографічного алгоритму з відкритим ключем RSA.

Тест простоти Міллера–Рабіна або тест Міллера–Рабіна — це тест простоти. Його початкова версія, яку розробив професор Міллер, є детерміністичною, але детермінізм покладається на недоведену узагальнену гіпотезу Рімана; Міхаель Рабін змінив її, щоб отримати безумовний імовірнісний алгоритм [14].ТестпростотиМіллера-РабінататестпростотиСоловея-Штрассенаєвдосконаленимиваріантами, яківизначаютьвсіскладенічисла (цеозначає: длякожногоскладеногочислаn, принаймні 3/4 (Міллер-Рабін) або 1/2 (Соловей-Штрассен) чиселаєсвідкамискладеностіn). На ці методи часто падає вибір, бо вони набагато швидші, ніж інші загальні тести простоти.

Леонард Адлеман та Хуанг запропонували варіант без помилки (але лише з очікуваним поліноміальним часом виконання) тесту простоти на основі еліптичних кривих. На відміну від інших імовірнісних тестів, цей алгоритм дає сертифікат простоти, а тому може бути використаний для доведення простоти числа. На практиці виявилося що цей алгоритм є занадто повільним.

2.1.3 Швидкі детерміновані тести

Близько початку 20 сторіччя дослідження показали, що тези з малої теореми Ферма можна використовувати для перевірки на простоту. Це призвело до появи тесту на простоту Поклінґтона. Однак, через те, що цей тест вимагає

часткову факторизацію n-1 його часова складність у найгіршому випадку все ще дуже велика. Першим детермінованим тестом простоти значно швидшим, ніж наївні методи, був циклотомічний тест; для часу його виконання отримано оцінку $O((\log n)^{c\log(\log(\log(n)))})$, де n тестоване на простоту число, а c константа, незалежна від n. Це повільніше, ніж поліноміальний час.

Для тесту простоти на основі еліптичних кривих можна отримати оцінку $O((\log n)^6)$, але лише коли використовуємо деякі ще не доведені (але які як правило припускаються вірними) положення аналітичної теорії чисел. Це один з з найчастіше вживаних на практиці детермінованих тестів.

Реалізація цих двох методів досить важка, бо є великий ризик помилок при програмуванні; це одна з причин, чому їм не віддають перевагу.

Якщо вважається вірною узагальнена гіпотеза Рімана, то тест Міллера-Рабіна можна звести до детермінованої версії з часом виконання $O((\log n)^4)$. На практиці, цей алгоритм повільніший, ніж два інших для величин чисел, з якими можна реально оперувати.

У 2002, Маніндра Агравал, Нітін Саксена та Нірай Кайал описали новий детермінований тест простоти, AKS тест простоти, який як доведено виконується за $O((\log n)^{12})$ [15]. Крім того, якщо вірна гіпотеза Харді-Літлвуда, яку вважають справедливою, то він виконується за $O((\log n)^6)$. Отже, маємо перший детермінований тест простоти з доведеним поліноміальним часом виконання. На практиці, цей алгоритм повільніший, ніж імовірнісні методи.

2.2 Числа Серпінського як приклад використання визначення простих чисел

За визначенням, числом Серпінського називається таке непарне натуральне число k, що для довільного натурального n число $k\times 2^n+1$ не є простим [16].В 1960 році Серпінський показав, що множина таких чисел k є нескінченою, але він не навів явного чисельного прикладу [17]. Також,

Серпінський захотів з’ясувати, яке найменше *k* можливе. На даний момент є гіпотеза, що число 78557, доказане число Серпінського знайдене в 1962 році Джоном Селфріджем, є найменшим. Воно «покриваїться» такою множиною простих чисел:

{3, 5, 7, 13, 19, 37, 73}

Тобто, кожне з чисел $78557 \cdot 2^n+1$ може бути розділене на одне з цієї множини без залишку.

Для того щоб доказати що число є Серпінським, треба для *k*знайти таку ступінь *n*що перетворює $k \cdot 2^n+1$ в просте. На даний момент залишилось 6 чисел, менших за 78557, що можуть відповідати властивості: 10223, 21181, 22699, 24737, 55459, та 67607.Наразі проблематико. визначення чисел Серпінського займається проект «SeventeenorBust», що вже успішно викреслив наступні кандидати на найменше число:4847, 5359, 19294, 27653, 28433, 33661, 44131, 46157, 54767, 65567та 69109. Просте число, що спростовує 19249 має довжину в 3918990 десяткових чисел. Очікується, що аналогічні прості числа для залишкових шести будуть ще більшими[17].

2.3 Дослідження принципу розподілення обчислень

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

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, що 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 [18].

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.

У цій роботі, обчислювальні сутності називаються комп'ютери або вузли.

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.

2.3.1 Історія виникнення терміну

Використання паралельних процесів, що взаємодіють один з однимшляхом передачі повідомлень має свої корені в операційній системі архітектур що вивчались в 1960 році. Перші поширені розподілені системи представляли собою локальні мережі, такі як Ethernet, що був винайдений в 1970-х роках.

Мережа ARPANET була введена в кінці 1960-х років, а електронна пошта від цієї компанії була винайдена на початку 1970-х років. Електронна пошта стала найуспішнішим застосуваннямARPANET, і це, ймовірно, найбільш ранній приклад великомасштабного розподіленого програмного забезпечення. На додаток до ARPANET, і його наступника, Інтернет, іншими ранніми комп'ютерними мережами по всьому світу можна вважатиUsenet і FidoNet з 1980-х, кожен з яких використовувався для підтримки розподілених систем зв'язку.

Дослідження розподілених обчислень перетворилось у власну гілку інформатики в кінці 1970-х років і на початку 1980-х років. Перша конференція в області, симпозіум про принципи розподілених обчислень (PODC), відбулася в 1982 році, і його європейський аналог Міжнародний симпозіум з розподілених обчислень (DISC) вперше був проведений в 1985 році [19].

2.3.2Теоретичні основи концепції

2.3.2.1 Моделі

Багато задач, які ми хотіли б автоматизувати за допомогою комп'ютера мають тип питання-відповідь: на поставлене запитання комп'ютер повинен видавати відповідь. У теоретичній інформатиці, такі завдання називаються обчислювальні задачі. Формально, обчислювальна задачаскладається з екземпляру завдання та вирішення кожного з цих завданнь.Екземпляри це питання, які ми можемо задати, а вирішення це відповіді на ці питання.

Теоретична інформатика прагне зрозуміти, які обчислювальні проблеми можна вирішити за допомогою комп'ютера (теорія обчислень) і наскільки ефективно (теорії складності обчислень). За замовчуванням вона свідчить, що проблема може бути вирішена за допомогою комп'ютера, якщо ми можемо розробити алгоритм, який виробляє правильне рішення для будь-якогоекземпляра. Такий алгоритм може бути реалізований у вигляді комп'ютерної програми, що виконується на комп'ютері загального призначення: програма зчитує екземпляр проблеми в якості входу, виконує деякі обчислення, і видає рішення в якості виходу.

Областіпаралельних і розподілених обчислень досліджують схожі питання у випадку якдекількох комп'ютерів, так і комп'ютера, який виконує набір взаємодіючих процесів: які обчислювальні завдання можуть бути вирішені в такій мережі і наскільки ефективно? Проте, це зовсім не очевидно, що мається на увазі під поняттям "вирішення проблеми" у випадку паралельних або розподілених систем: наприклад, яке завдання розробника алгоритму і який еквівалент паралельних або розподілених обчислень до послідовного комп'ютеру загального призначення?

Як правило, три точки зору використовуються:

1. Паралельні алгоритми в контексті моделі загальної пам'яті

Всі комп'ютери мають доступ до загальної пам'яті. Розробник алгоритму обирає програму, що буде виконуватись на кожному комп'ютері. Використовується тільки одна теоретична модель - паралельні машини випадкового доступу (PRAM).Проте, класична модельPRAM передбачає синхронний доступ до загальної пам'яті.Програми з загальною пам'яттю можуть бути розширені до розподілених систем, якщо використовувана операційна система інкапсулює зв'язок між вузлами і практично об'єднує пам'ять усіх окремих систем.

2. Паралельні алгоритми вконтексті моделі передачі повідомлень

Розробник алгоритмуобирає структуру мережі, а також програми, що будуть виконуватися на кожному комп'ютері. Використовуються такі моделі, як булеві схеми і мережа сортування. Булева схема може розглядатися як комп'ютерна мережа: кожнийшлюз може бути представлений як комп'ютер, щовиконує надзвичайно проступрограму. Аналогічно, мережу сортування можна розглядати як комп'ютерну мережу: кожен компаратор може бути представлений як комп'ютер.

3. Розподілені алгоритми в моделі передачі повідомлень

Розробник алгоритму обирає тільки комп'ютерну програму. Всі комп'ютери виконують одну і ту ж програму. Система повинна працювати правильно, незалежно від структури мережі. Зазвичай використовується модель графуіз скінченим автоматом представленим у вигляді вузла.

У разі розподілених алгоритмів, обчислювальні завдання, як правило, пов'язані з графіками. Часто граф, що визначає структуру комп'ютерної мережі є екземпляром проблеми.[20]

2.3.2.2 Міра складності

В паралельних алгоритмах, ще один ресурс, на додаток до часу та простору це кількість комп'ютерів. Дійсно, часто треба йти на компроміс між часом та кількостю комп'ютерів: проблема може бути вирішена швидше, якщо є кілька комп'ютерів, що працюють паралельно. Якщо проблема може бути вирішена в полілогаріфмічний час з використанням поліноміальноїкількості процесорів, то кажуть, що проблема знаходиться в класі NC.Клас NC може бути визначений однаково добре за допомогою PRAM формалізму або Булевих схем - PRAM машини можуть ефективно імітувати Булеві схеми і навпаки.

При аналізі розподілених алгоритмів, більше уваги зазвичай виділяється на операції зв'язку, ніж обчислювальним крокам. Можливо, найпростіша модель розподілених обчислень є синхронною системою, де всі вузли працюють «в одну ногу». Під час кожного раунду зв'язку, всі

вузлипаралельноодержують останні повідомлення від сусідів, виконуютьпевні локальний обчислення, і надіслають нові повідомлення для своїх сусідів. У таких системах, центральною мірою складності виступає кількість синхронних раундів зв'язку, необхідних для виконання завдання.

Ця міра складності тісно пов'язана з діаметром мережі. Нехай *D* діаметр мережі. З одного боку, будь-якаобчислювальна проблема може бути вирішена тривіально в синхронній розподілений системі приблизно за 2*D* раундів зв'язку: зібрати всю інформацію в одному місці (*D*раундів), вирішити проблему, і повідомити кожен вузол про рішення (*D*раундів).

З іншого боку, якщо час роботи алгоритму є набагато менше ніж *D* раундів зв'язку, то вузли мережі повинні обробляти свою задачу, не маючи можливість отримати інформацію про далекі частини мережі. Іншими словами, вузли повинні глобально правильні та стійкі рішення на основі інформації, яка доступна в їх локальнійоколиці. Відомі багато розподілених алгоритмів з тривалістю набагато менше, ніж D раундів, і розуміння, які проблеми можуть бути вирішені за допомогою таких алгоритмів є одним з центральних питань дослідження області.

Інші часто вживані засоби виміреня складності - загальна кількість бітів, переданих в мережі [21].

2.3.2.3 Властивості розподілених систем

До даного моменту увага була зосереджена саме на розробці розподіленої системи, яка вирішує проблему. Завдання додаткового дослідження вивчає властивості даної розподіленої системи.

Проблема зупинки - аналогічний приклад з області централізованих розрахунків: ми дали комп'ютерну програму і його задача – вирішити, зупиняється вона чи працює вічно. Проблему зупинки неможливорозв’язати в загальному випадку, ірозуміння поведінки комп'ютерної мережі такеж складне, як ірозуміння поведінки одного комп'ютера.

Тим не менш, є багато цікавих особливих випадків, які можна розв'язати. Зокрема, можна міркувати про поведінку мережі кінцевих автоматів. Однин із прикладів, це визначення чи дана мережа взаємодіючих (асинхронних та недетермінірованих) кінцевих автоматів може зайти в глухий кут. Ця проблема PSPACE-повна, тобто загалом вирішувана, але швидше за все не існує ефективного (централізованого, паралельного або розподіленого) алгоритму, який вирішує проблему у випадку великих мереж.

2.3.3 Архітектури розподілених мереж

Для розподілених обчислень використовуються різні апаратні і програмні архітектури. Розподілене програмування як правило потрапляє в одну з декількох основних архітектур чи категорій: клієнт-серверних, 3-рівнева архітектура, *n*-рівнева архітектура, розподілених об'єктів, слабкого зв'язку, або тісного зв'язку.

- Клієнт-сервер: клієнтський вузолзв’язується з сервером,форматує дані і відображає їхкористувачеві. Введення в клієнтрезультує у переміщенні вводу до сервера.
- 3-рівнева архітектура: трьохрівневі системи переміщують логіку клієнта на середній рівень, так що можуть бути використаніклієнти без будь-якого стану. Це спрощує розгортання додатків. Більшість веб-додатків побудовані на архітектурі 3-рівня.
- *n*-рівнева архітектура: відноситься, як правило, до веб-додатків, які надалі передають свої запроси на інші сервіси. Цей тип програм є одним з найбільш відповідальних за успіх серверів додатків.
- тісно з’єднані (кластерні): відноситься, як правило, до груп машин, що тісно працюють разом, паралельно виконуючи спільний процес. Завдання підрозділяється на частини, які розроблені індивідуально для кожного з них і потім додаються назад разом, щоб знайти остаточний результат.

- Peer-to-peer: архітектура, де немає ніяких спеціальних машин або таких, які забезпечують обслуговування або управління мережевими ресурсами. Замість цього всі обов'язки рівномірно ділиться між усіма машинами, відомих як peers. Peers можуть служити як у якості клієнтів, так і серверів.
- Просторово-орієнтовані: відносяться до інфраструктури, що створює ілюзію (віртуалізацію) одного адресного простору. Дані явно відтворені відповідно до потреб програмного забезпечення. Це дозволяє досягти незалежності у часі, просторі і посиланні.

Інший основний аспект розподіленої обчислювальної архітектури це метод спілкування та координації роботи між паралельними процесами. За допомогою різних протоколів передачі повідомлень, процеси можуть спілкуватися один з одним безпосередньо, як правило, у форматі майстер / відомий. Крім того, база даних, орієнтована на архітектуру може дозволити розподіленим обчисленням бути зробленими без будь-якої форми безпосереднього спілкування між процесами, а шляхом використання загальної бази даних [22].

2.4 Висновки

В даному розділі було розглянуто основні алгоритми, які використовують для знаходження та тестування простих чисел. В ході дослідження було описано особливості кожного алгоритму, перелічено їх переваги та недоліки. На основі цього дослідження було вибрано алгоритм, який буде використовуватись для задачі.

Крім того були описані особливості класифікації та теоретичні основи побудови розподілених систем. Також були детально описанівластивості та можливі архітектури розподілених систем, були представлені способи вирішення поставленої проблеми.

3. АРХІТЕКТУРА

3.1 Вибір архітектури програмного забезпечення реалізації

Першим питанням, що постає перед розробником є вибір архітектури програмного забезпечення. Над цим питанням замислювалися багато років назад, вже тоді, коли починали писати перші програми. Перші програми виконувалися без операційної системи. Вони були написані на мові асемблера і працювали безпосередньо з обладнанням. В 1960-их роках почали з’являтися операційні системи. Разом з їх появою, з програміста були зняті частина завдань - управління головкою жорсткого диска, організація доступу до окремих секторів і організація файлової системи. Однак питання формату зберігання даних і пошуку даних вирішувалося самим програмістом. У 1970-ті роки набувають поширення системи управління базами даних - системи, які знімали з програміста основні питання по зберіганню та вибірки даних. Подальший розвиток СКБД дозволив працювати з однією базою даних відразу декільком користувачам. Причому ці користувачі могли працювати на різних комп'ютерах, з'єднаних локальною мережею. Зазвичай розділяють два типу клієнт - серверних додатків: з товстим клієнтом і тонким клієнтом. Відмінність полягає в тому, хто виконує основну роботу з обробки даних. Якщо основна робота проводиться на сервері, а клієнт тільки відображає дані - прийнято називати його «тонким». Якщо ж клієнтська програма займається обробкою даних, а сервер - тільки організацією доступу до бази даних - то такий клієнт називається «товстим».

Наступним кроком у розвитку архітектури програмного забезпечення була поява дворівневої архітектури. До дворівневих архітектур відносять системи, в яких є СКБД і програма для роботи з даними. В дворівневих архітектурах кожен користувач має свій екземпляр програми, але всі вони використовують єдине сховище даних.

На цьому розвиток архітектури не зупинився і була запропонована трьохрівнева архітектура програмного забезпечення.

У трьохрівневій архітектурі додається ще один додаток, який працює з базою даних і служить буфером між додатками користувачів і СКБД. У цьому додатку зручно помістити всю обробку даних (розрахунок зарплат, видачу стипендій, обробку та зберігання будь-яких даних). Якщо раптом зміняться вимоги до програми (наприклад, зміниться алгоритм розрахунку прибуткового податку), достатньо буде зробити зміни лише в одній програмі, яка знаходиться на сервері. «Тонкі» клієнти користувачів можна залишити без зміни. Також програми, що використовують трьохрівневу архітектуру є безпечними. Якщо ми надаємо клієнтського додатку доступ до бази даних (як в 2-х рівневій архітектурі), то користувач може «підмінити» клієнтську програму і провести небажані зміни в даних. В трьохрівненвій архітектурі програмісту потрібно писати лише головну частину - серверну. А клієнти використовують стандартну програму для доступу до серверного додатку. Така стандартна програма називається ультра-тонким клієнтом. І вона є практично на кожному комп'ютері - це інтернет-браузер. Завдяки стандартним протоколам TCP/IP і HTTP і стандартнїій мови HTML інтернет-браузер може спілкуватися з нашим сервером і надавати користувачу інтерфейс. Покажемо схематично трьохрівневу архітектуру на (рис. 3.1).

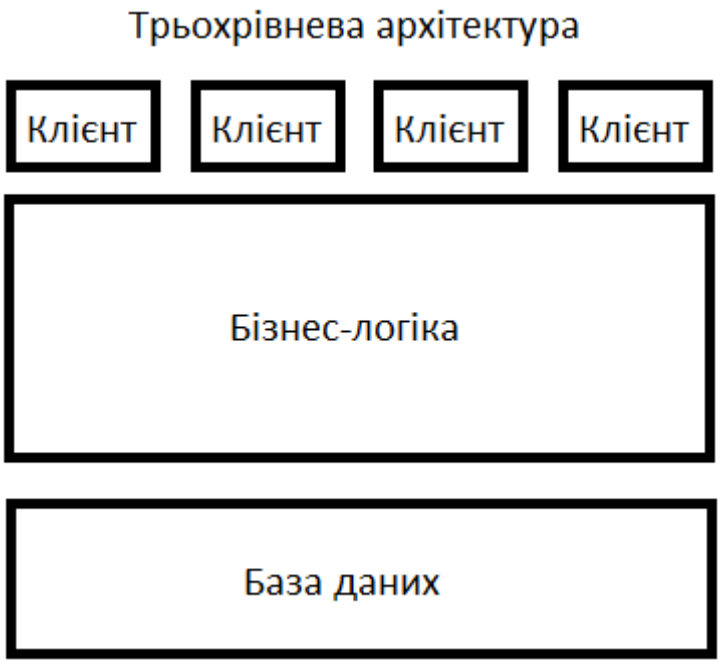

Рисунок 3.1 - Трьохрівнева архітектура

Для виконання поставленої задачі ми будемо використовувати Hadoop. 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 безліч дрібніших відособлених фрагментів, кожен з яких може бути запущений на окремому вузлі кластера. До складу Hadoop входить також реалізація розподіленої файлової системи Hadoop Distributed Filesystem (HDFS), котра автоматично забезпечує резервування даних і оптимізована для роботи MapReduce-застосунків. Для спрощення доступу до даних в сховищі Hadoop розроблена БД HBase і SQL-подібна мова Hive, яка є свого роду SQL для MapReduce і запити якої можуть бути розпаралелені і оброблені кількома Hadoop-платформами [23].

Для того щоб скористатися цим інструментарієм треба встановитиекземпляр (вузол) на комп’ютер з однією з підтримуваних операційних систем. Далі, треба дати цьому вузлу «роботу» на опрацювання. В

деталях розробнику треба описати що буде виконуватись при розділенні задачі, та що при зборі даних назад.

Вже на цьому показовому описі ми можемо переконатись у зручності трьохрівневого подходу – розробнику не треба думати про те, що відбувається на рівень нижче: як дані розподілені, де вони знаходяться та яким чином буде зроблена передача. Все що нам потрібно – це віртуальна абстракція «дискового простору», до якого ми можемо отримати доступ через HadoopDistributedFileSystem.Крім того, вона має низку переваг і задовольняє всім необхідним вимогам:

- масштабованість;
- конфігурованість - ізольованість рівнів один від одного дозволяє швидко і простими засобами переконфігурувати систему при виникненні збоїв або при плановому обслуговуванні на одному з рівнів;
- висока безпека;
- висока надійність;
- низькі вимоги до продуктивності та технічним характеристикам терміналів, як наслідок зниження їх вартості.

3.2 Вибір мови програмування для створення додатків

Далі постає питання про вибір тих технологій, за допомогою яких можна реалізувати програму. Хоча найбільш поширеною мовою для створення Hadoop-додатків є Java, існує спосіб у котрий можна використативиконуваний модуль MapReduceнаписаний на інших мовах. Зараз користуються популярністю дві платформи: JEE і .NET. Порівняємо їх і виберемо ту платформу, за допомогою якої ми будемо досліджувативиокопродуктивні обчислення.

3.2.1 Кросплатформеність

Однією з найбільших переваг JEE над .NET є кросплатформеність. Але, якщо вірити Microsoft, то це більше не є прерогативою JEE. Microsoft позиціонує .NET як платформу з двоступеневою компіляцією, що дозволяє створювати середовище виконання для будь-якої платформи, подібно Java. На даний момент Microsoftпредставила специфікації для CLItaC# до ECMAта ISO, зробивши їх стандартами, але не відомо коли треті особи презентують відповідну версію такого програмного забезпечення. Таким чином, можемо зробити висновок, що найбільш вдалим вибором буде JEE.

3.2.2 Багатомовна підтримка

Єдиною мовною основою JEE є Java, що сильно відрізняється від .NET, де підтримується велика кількість мов програмування, включаючи Fortran, COBOL, C++ та Visual Basic. Можна, звичайно, посперечатися щодо того, що єдина мова є більш елегантним рішенням, однак треба визнати, що .NET пропонує більш просте рішення для організацій, які хочуть користуватися знаннями, які вже є у їх розробників. Адже для тих розробників, які використовують мови, відмінні від Java, .NET дає можливість створювати Web-служби звичною для них мовою з мінімальними витратами на перенавчання.

3.2.3 Документація

Зробимо аналіз документації для кожної з платформ. На відміну від JEE, платформа .NET не має гарної документації. Є специфікації на мову C# і на середовище виконання CLR. Але цього недостатньо, щоб зрозуміти всі можливості .NET. Дехто може зауважити, що .NET має MSDN. Але MSDN не є специфікацією. Це погано структурований збірник інформації, що складається з довідки для різних API, статей, прикладів, але цього недостатньо. Тому можна зробити висновок, що платформа JEE краще задокументована.

3.2.4 Інструментарій для розробки

На сьогоднішній момент єдиним середовищем розробки, який є повністю придатним для реалізації проектів промислового масштабу на платформі .NET, можна вважати лише Visual Studio.NET фірми Microsoft. Що стосується JEE, то тут існує величезна маса інструментів, у тому числі інтегрованих середовищ розробки. Також є відкриті проекти, які наразі повністю безкоштовні. Наведемо список основних або найбільш відомих IDE:

- Oracle JDeveloper;
- Borland JBuilder;
- IntelliJ IDEA;
- NetBeans IDE;
- Eclipse IDE.

Крім того, є ще одне питання, пов'язане з підтримкою інструментарію самої платформою JEE. Але і ця сторона платформи теж відкрита і стандартизована, на відміну від .NET. Також, треба прийняти до уваги те, що самі середовища крос-платформні, на відміну до VisualStudio, де існує жортка прив’язка до операційної системи MicrosoftWindows.

Отже, більш гнучким та універсальним буде вибір інструментарію для розробки до платформи JEE.

3.2.5 Ціна

Розповсюджена думка, що рішення на базі JEE є дорогими. При цьому всі забувають, що є безкоштовні засоби, на базі яких можна побудувати промислову систему. Для прикладу, якщо ви не хочете платити за Oracle iAS, то ви можете використовувати JBoss разом із вихідними кодами на додачу. А якщо при цьому ви будете використовувати, наприклад, ОС Linux і СКБД MySQL, то отримаєте безкоштовну платформу Enterprise-класу. Для Windows і .NET це нереально до тих пір, поки не буде вільної реалізації .NET і всієї необхідної інфраструктури Enterprise-класу.

На основі цього порівняння можна зробити висновок, що для написання програмного продукту більш доречно використовувати саме технологію JEE.

3.3 Вибір способу запуску фреймворку Hadoop

Далі постає питання завантаження інтрументарію та розгортання Hadoop. На вибір у нас є образи віртуальних машин від Cloudera та Hortonworks, а також варіант встановлення програмного забезпечення на певну операційну систему вручну. Зробимо зауваження, що найбільш стабільною операційною системою на яку можна поставити Hadoop– це Linux. Порівняємо можливі альтернативи і зобразимо результати у вигляді таблиці 3.1

Таблиця 3.1 – Порівняння способів

Назва методу	Час розгортання та обслуговування *	Інтерфейс користувача	Наявність додаткового ПО
Ручний	Декілька годин, в залежності ознайомленості користувача з Linux-подібними системами	Той, що вибере користувач (гнучкий вибір серед дистрибутивів ОС Linux)	Можна встановити самому, але це негатино вплине на час розгортання
Hortonworks	Після завантаження готова до користування	Текстовий, в стилі терміналу. Можливість з’єднуватися через веб-інтерфейс.	Нема
Cloudera	Після завантаження готова до користування	Графічний, на базі CentOS	Існує підтримка веб-сервісів що з будь-якого вузла відстежувати та керувати виконаням задач

*Зазначимо що до часу розгортання не відноситься часзавантаження та встановлення програмного забезпечення для зчитування віртуальних образів операційних систем, та самих ОС.

Беручи до уваги всі плюси та мінуси кожного з методів обираємо спосіб встановлення фреймворку через віртуальну ОС від Cloudera.

3.4 Аналіз архітектури системи

Як вже зазначалось, даний програмний продукт реалізовує трьохрівневу архітектуру. Покажемо взаємозв'язок між рівнями на рис 3.2.

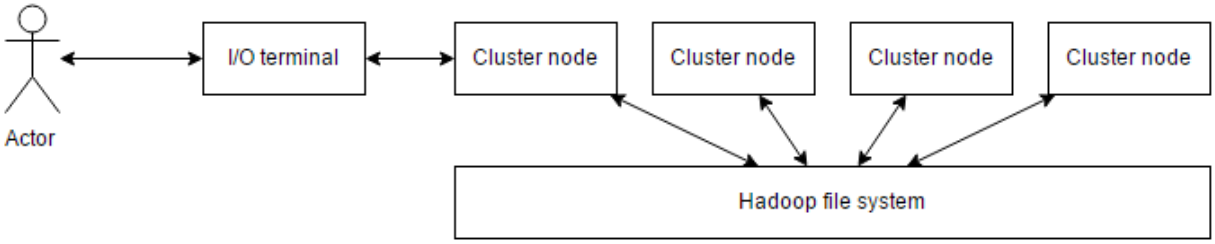

Рисунок 3.2. Взаємозв'язок між рівнями фреймворку

Для того, щоб почати працювати з дистрибутивом розподіленого обчислення задач необхідно мати доступ до будь-якого вузла середовища. При цьому компіляція може бути зроблена будь-де, за наявністю необхідних пакетів та бібліотек.Введення відкомпільованогоJava байт-коду повинно бути зроблено за допомогою терміналу введення/виведення, через який можна дати завдання всьому кластеру.В якості параметру до задачі задається шлях до вхідних даних що повинні знаходитися в HDFS.Відповідно початку вхідні дані треба туди помістити за допомогою терміналу. Результати роботи будуть збережені всередині розподіленої файлової системи, за шляхом також заданим в якоті параметру до задачі.

Для того щоб мати можливіть працювати з даними які з легкітю можуть підлягати компресійній обробці при передачі, користувачеві був предоставлений набір спеціальних типів даних що відповідають стандартним в середовищі Hadoop.

Для того щоб фреймворк мав можливість виконувати декілька компонентів однієї задачі на різних пристроях в потенційно тей самий час, треба описати за яким принципом буде проводитися розподілення, що буде виконуватися на кожному з вузлів та як буде проводитися злиття результатів. За замовчуванням виділяється по одному вузлу на кожен файл вводу. Для наших тестових прикладів цей спосіб є прийнятним. Для того щоб завдати логіку розділення/злиття потрібно створити 2 класи що наслідуються від класів фреймворку MappertaReducer відповідно.Результуючу архітектуру покажемо на рис 3.3

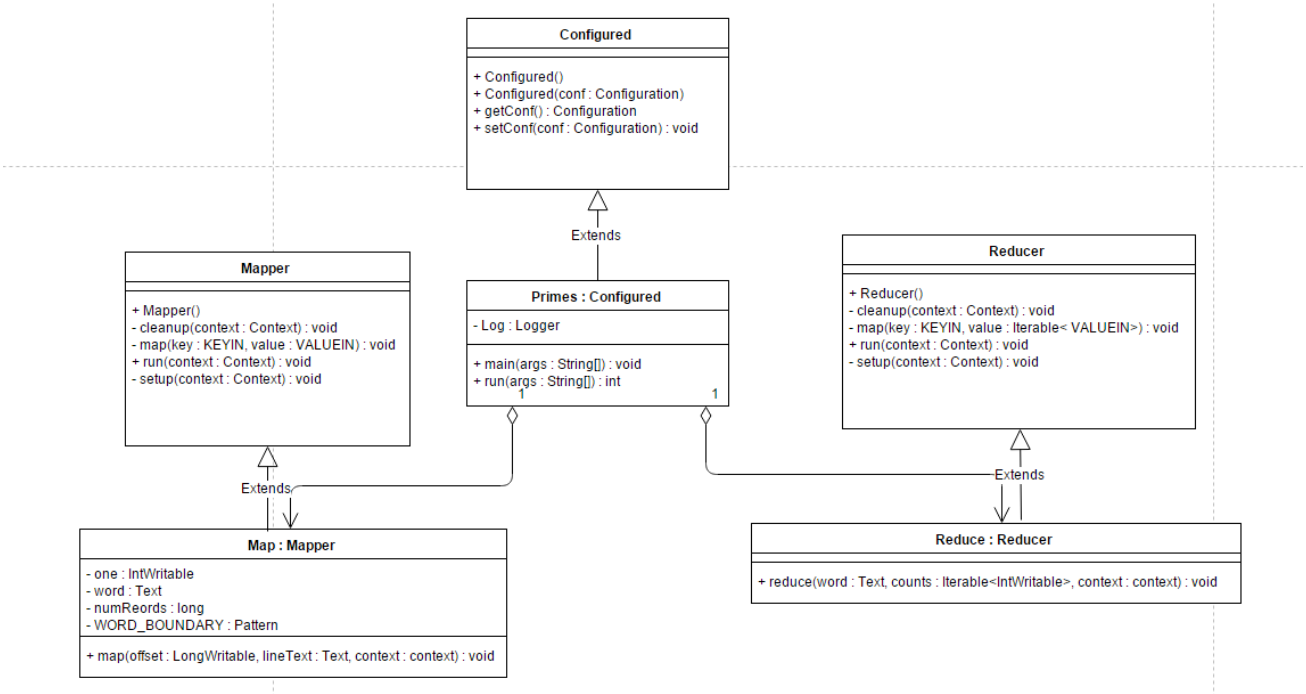

Рисунок 3.3 – Архітектура додатку

3.5 Керівництво користувача

Після того як ми встановили все необхідне програмне забезпечення ми можемо розпочинати роботу. Для цього треба запустити віртуальну машину із файлом ОС на якій встановлений фреймворк Hadoop (рис 3.4).

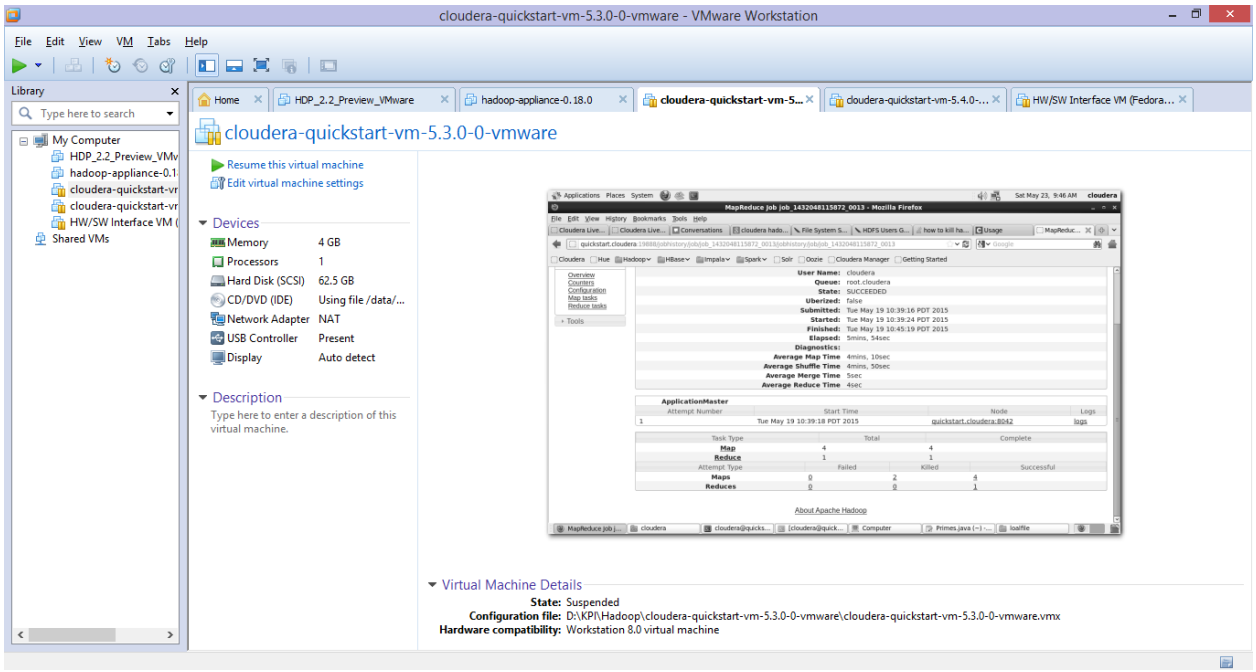

Рисунок 3.4 - Головна сторінка віртуальної машини

Якщо ми вибрали варіант з предвстановленою операційною системою, то після цього можна одразу приступати до написання коду (рис 3.5).

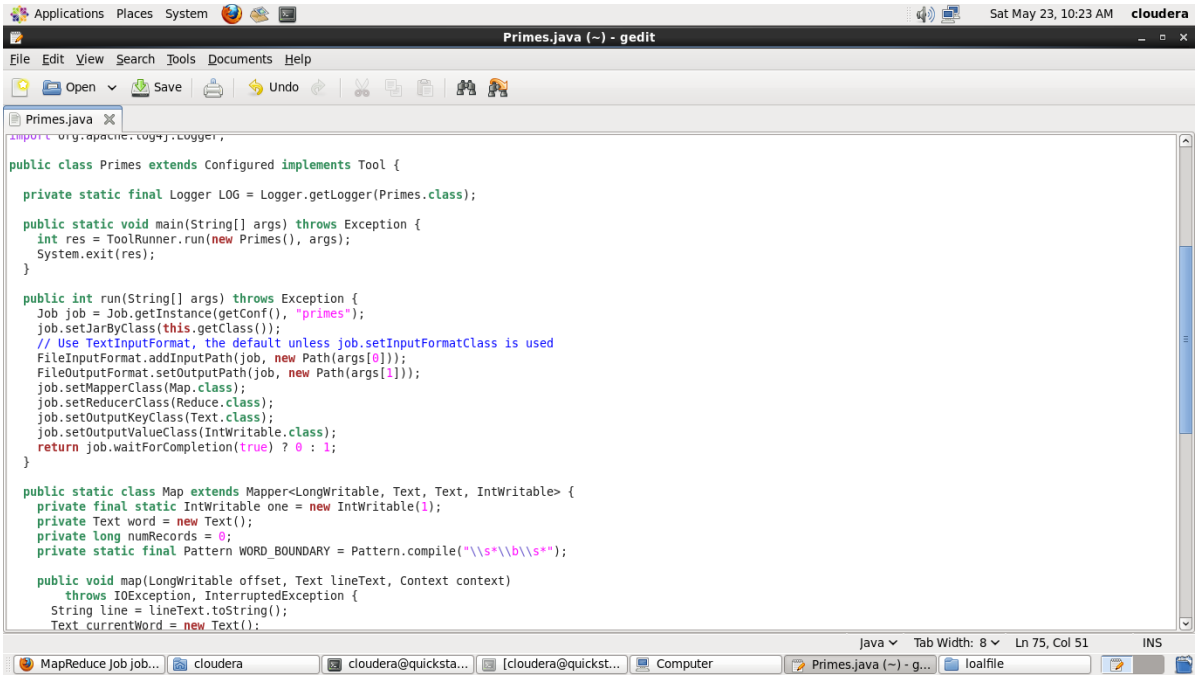

Рисунок 3.5 – Написання коду у віртуальній машині

Цілком можливо налаштувати інтегроване середовище розробки Eclipse, але для показовості прикладу наберемо код у звичайному текстовому редакторі.

Тепер, як ми написали «як» обробляти, ми повинні створити «що» обробляти. Для цього треба створити довільну кількість файлів із вхідними даними. Даний процес зображено на (рис. 3.6).

Рисунок 3.6–Файли вхідних даних

Але самої лише наявності файлів на дисковому просторі недостатньо. Треба завантажити їх до розподіленої файлової системи фреймворку. Для того щоб ініціалізувати роботу з Hadoop треба відкрити термінал та прописати послідовність команд зображену на (рис 3.7).

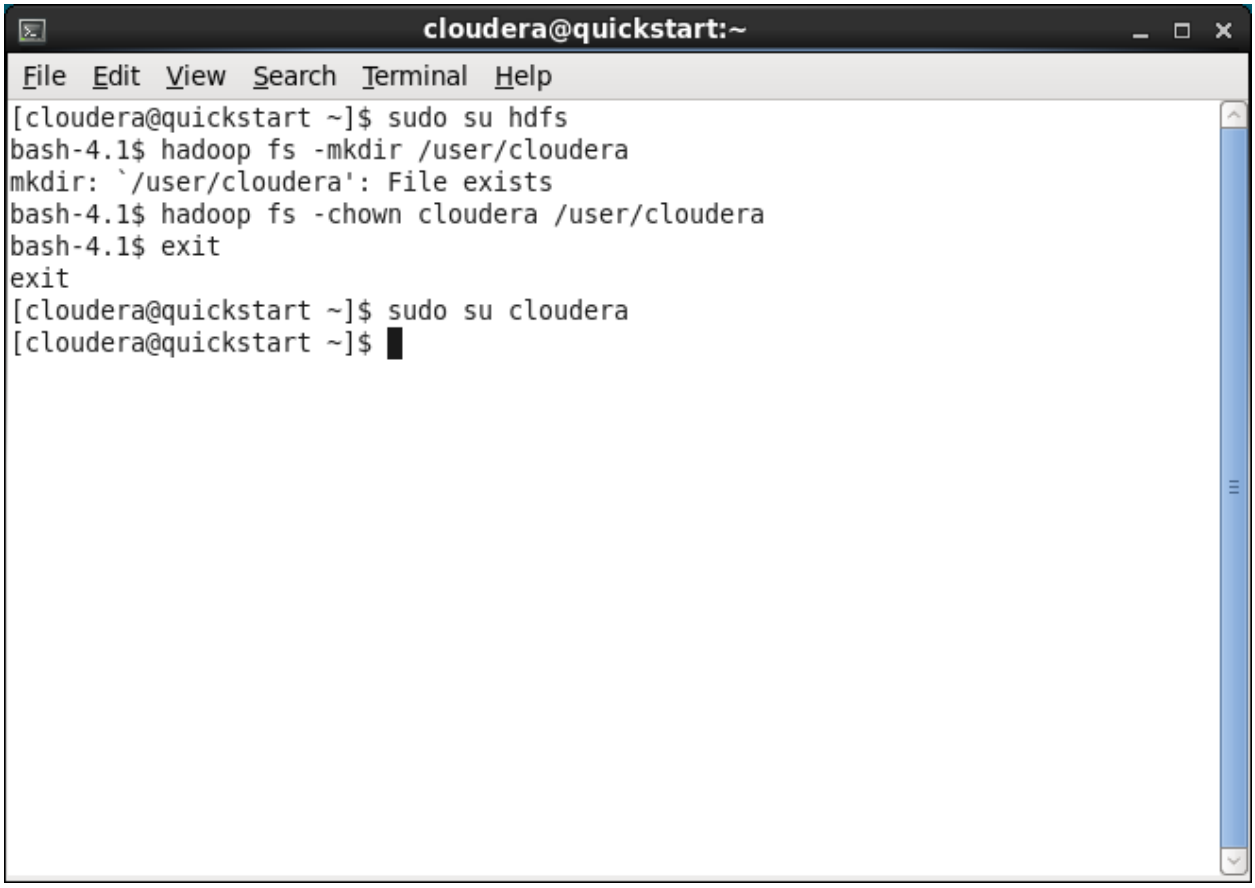

Рисунок 3.7–Послідовність команд

Тепер треба підготовити робочий простір та створити папки вхідних та вихідних даних як зображено на (рис. 3.8)

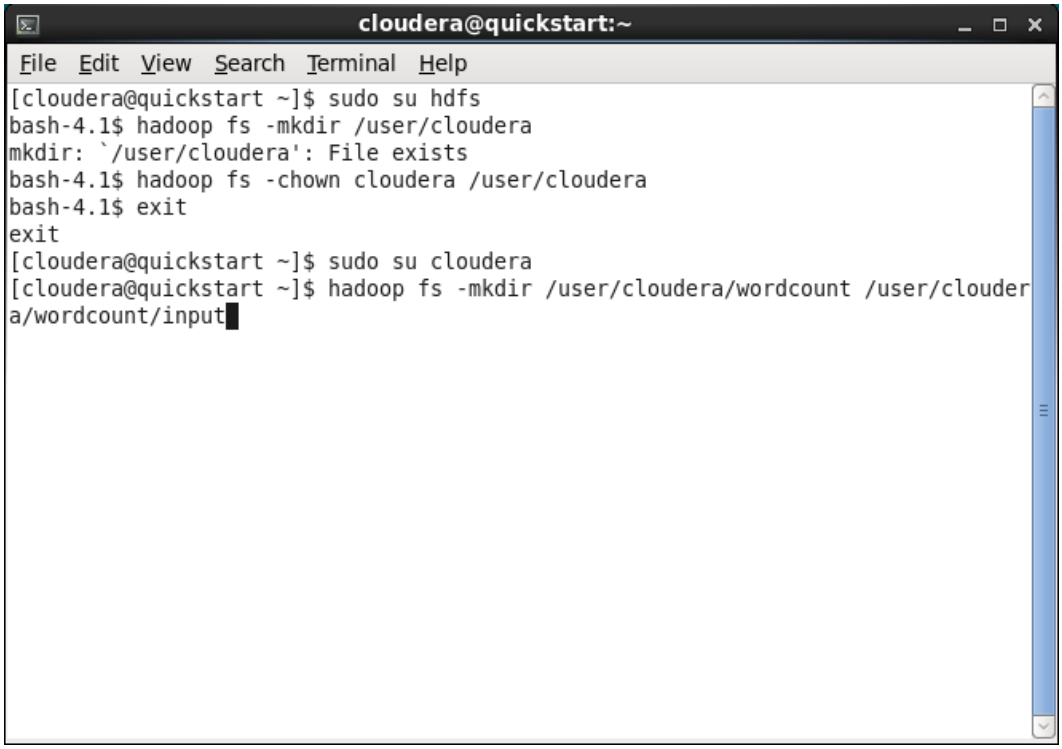

Рисунок 3.8–Створення папок вводу/виводу

Зараз вже можемо завантажувати вхідні дані до системи (рис. 3.9).

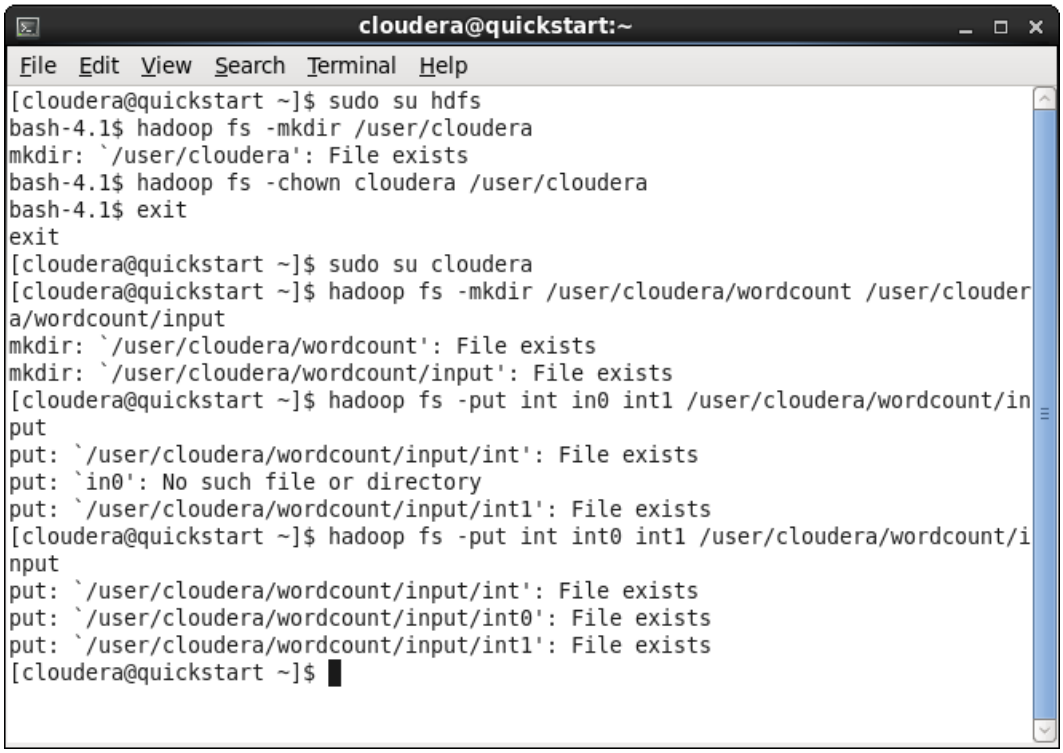

Рисунок 3.9–Завантаження файлів до фреймворку

Перед безпосереднім виконанням задачі залишився ще один шаг – скомпілювати Java-код та згенерувати виконуючий .jar-файл (рис. 3.10).

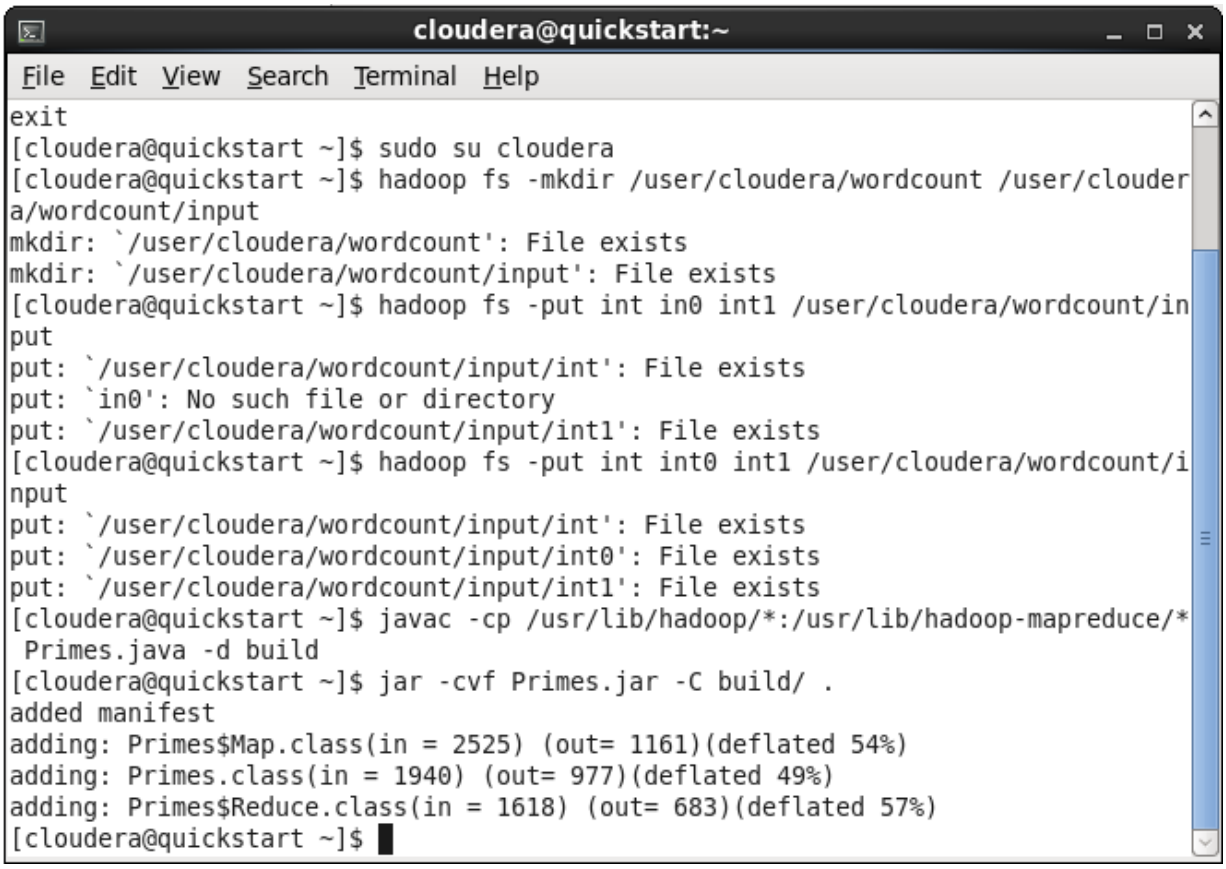

Рисунок 3.10–Компіляція Java-коду

Для того щоб дати задачу на виконаня для системи треба ввести в термінал команду зображену на (рис. 3.11).

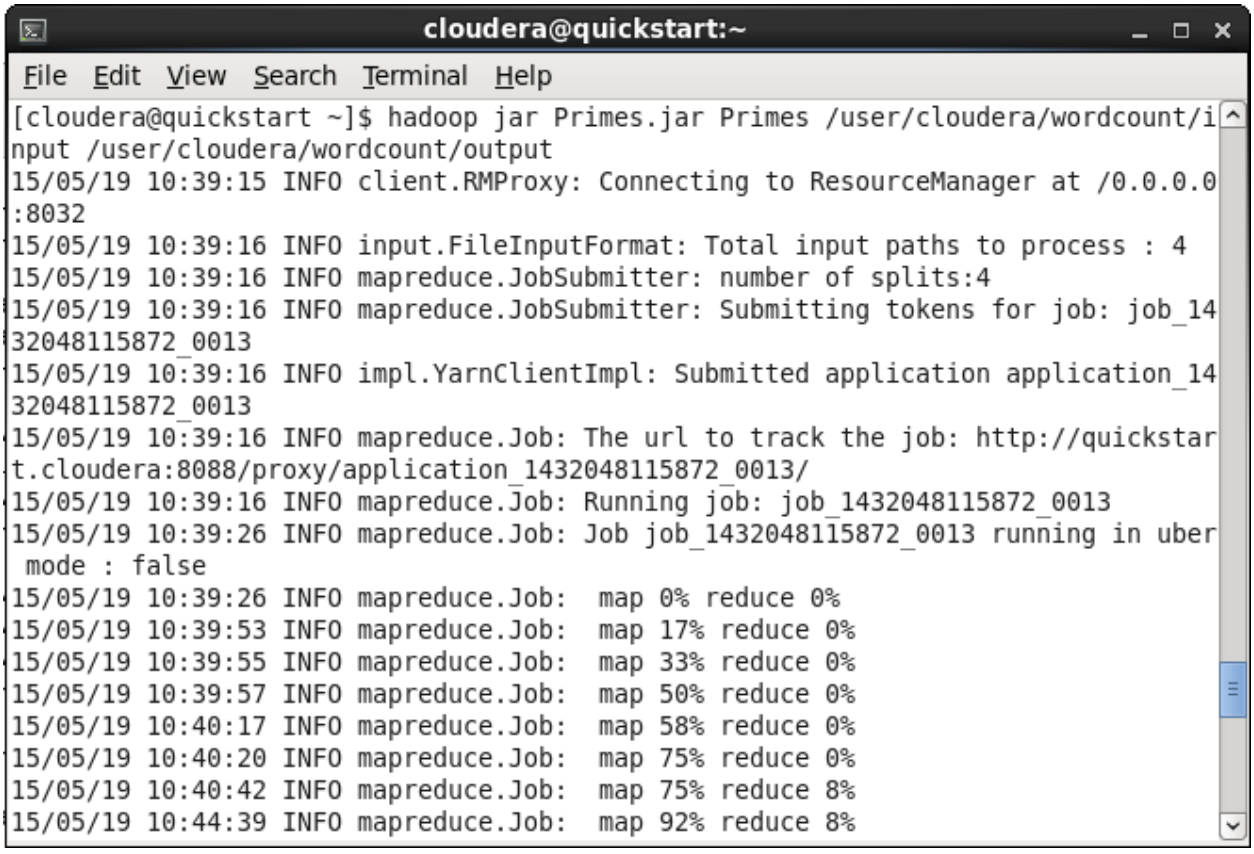

Рисунок 3.11–Запуск задачі

Як бачимо, робота була успішно подана на опрацювання, можна помітити який статус у задачі та на якому етапі зараз проводиться виконання.

Для більш детального розгляду всіх задач за весь час треба перейти по посиланню вказаному в терміналі. Відкриється вікно браузеру, що продемонстровано на (рис. 3.12).

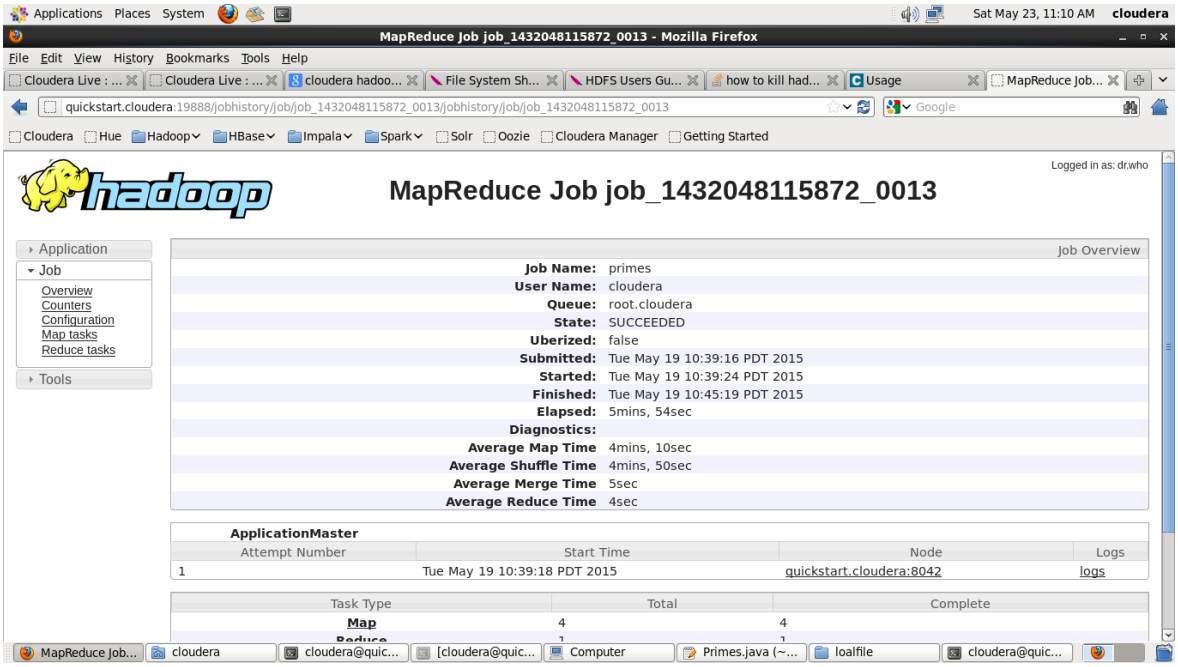

Рисунок 3.12–Веб-сторінка з інформацією про задачу

Все що залишилося це скачати файл з вихідними даними з дискового простору розподіленої системи, що зображено на (рис. 3.13).

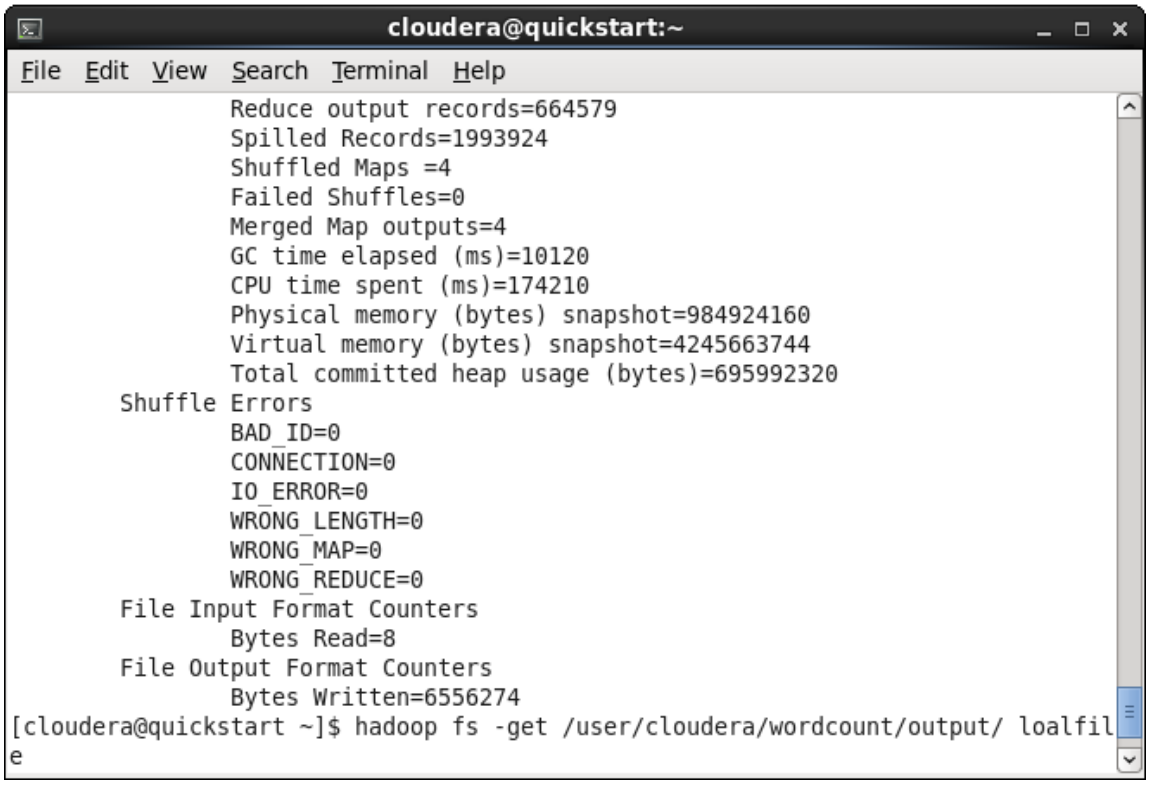

Рисунок 3.13–Завантаження виводу назад до простору ОС

Пересвідчуємося, що новий файл дійсно з’явився в локальниі файловій системі ОС Linux (рис. 3.14).

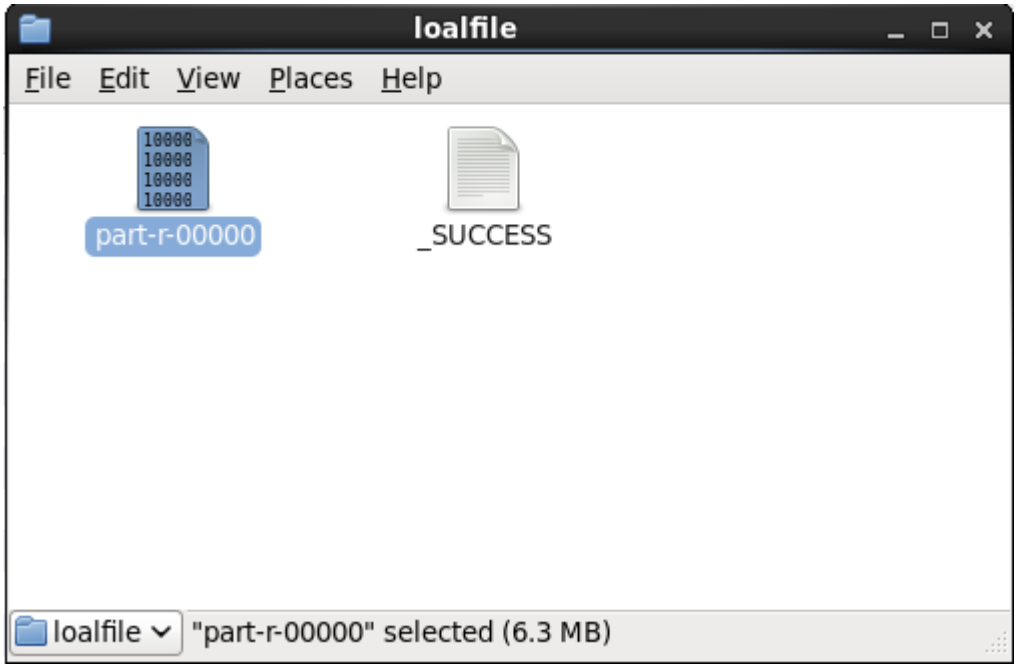

Рисунок 3.14–Завантажений вихідний файл

3.6 Аналіз отриманих результатів

Зробимо аналіз отриманих результатів. У результаті виконання даної дипломної роботи було отриманого програмний продукт, за допомогою якого можна згенерувати таблицю простих чисел. Даний продукт являє собою додаток, що може бути виконаний на системі розподілених обчисленьHadoop. Для його розміщення було використано послуги компанії Cloudera, що надала вільний та безкоштовний доступ до образу операційної системи Linux для віртуальної машиниз предвстановленним фреймворком.

Для генерації таблиці простих чисел були використаніймовірністні алгоритми з послідуючою перевіркою останнього числа, який, як було описано в статті, показав доволі гарні результати.

3.7 Висновки

В даному розділі було проаналізовано особливості архітектур програмного забезпечення та пакету розробки додатків до системи Hadoop і було з’ясовано, що в даному програмному продукті повинна використовуватись архітектура з трьома класами що наслідуються від стандартних з пакету розробки. Був вибранийметодздобуття всієї системи в цілому та реалізована покрокова інструкція по розробці програмних додатків для фреймворку. В ході дослідження було описано реалізацію даної архітектури і описано про реалізацію кожного рівня.

Крім того було написано керівництво користувача, користуючись яким, людина, яка перший раз працює з даною програмою, може легко в ній розібратись і внести необхідні зміни, після чого швидко і без надзвичайних зусиль зможе використати необхідні ресурси для вирішення проблеми. Також зроблено аналіз результатів, які були отримані в ході виконання роботи.

4.АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1 Опис задачі

Метою даної роботи є дослідження використання фреймворку Hadoop, що не э можливим без опису тестовихприкладів та системи оцінювання. У цьому розділі будуть описані типи задач які мі будемо розв’язувати, безпосередньо алгоритми розв’язання, виведені результати роботи фреймворку та представлена система порівнянь.

4.2 Описприкладів що будуть розв’язуватися

Отже, для показовості оберемо три класичні математичні задачі: генерація таблиці простих чисел, пошук чисел Серпінського використовуючи цю таблицю та задачу комівояжера.

4.2.1 Генерація таблиці простих чисел

Ідея доволі очевидна: треба знайти всі прості числа не більше ніж заданий ліміт. Технічна частина, як би там не було, не така проста. Для розв’язування будемо використовувати алгоритм запропонований у статті Качко М. та Яременко В. «PRIMALITYTESTPROBLEM» [25] – будемо перевіряти всі числа до максимуму ймовірністним методом, а потім перевіряти чи співпадають індекс та запис простого числа, таким чином піддаючи увесь список перевірці. Для розділення задачі на вузли будемо використовувати метод описаний Качко М. в статті «Primetablegeneration» [26]. Роботу способу можна продемонструвати на (рис. 4.1).

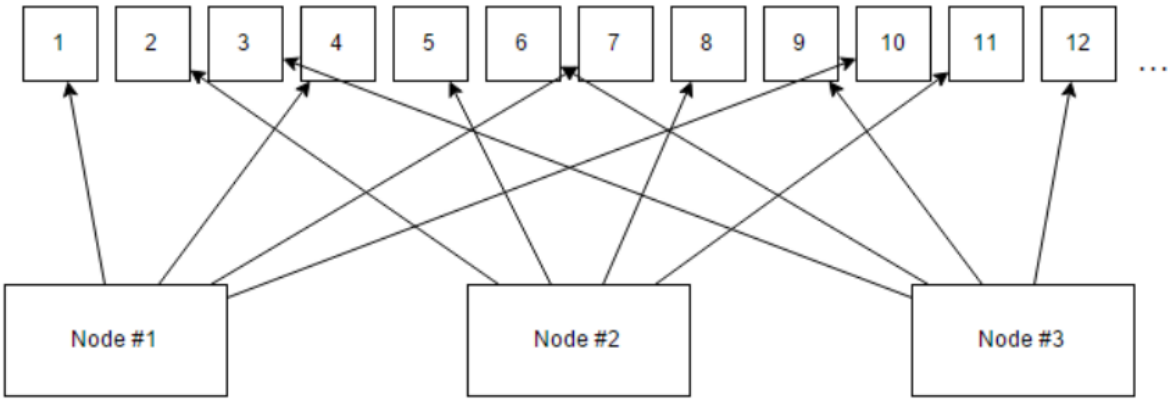

Рисунок 4.1 – Розподілення вхідних даних

Кожен вузол буде перевіряти тільки належну йому частину вхідних чисел. У такий спосіб на кожен вузол прийдеться приблизно однакова кількість інформації що треба опрацювати. Таким чином буде забезпечене рівноміре завантаження всіх вузлів, а також оптимальний час роботи програми.

4.2.2 Пошук чисел Серпінського

Задане число k. Для нього треба знайти всі такі n, що $k \times 2^n+1$ є складеним. Прямий розв'язок очевидний – перебирати всі n та перевіряти отримані числа на простоту, використовуючи попередньо згенеровану таблицю простих чисел. Оскільки задача зводиться до перебору натуральних чисел та тестування кожного з них, для цього прикладу скористуємося алгоритмом, аналогічним до попереднього. Дійсно, як і у ситуації з простими числами, там є набір натуральних чисел які треба перевіряти у певний спосіб. Тому, в цьому випадку, відправляємо кожному вузлу таблицю та його безпосередній порядковий номер. Далі метод слідує абсолютно тому ж самому алгоритму.

4.2.3 Задача комівояжера

Нехай є Nміст. Кожне місто з'єднано з кожним. Кожне з'єднання має свою «ціну». Треба знайти маршрут починаючи з першого міста що буде проходити через всі міста із найменшою загальною «сумою». Для цього прикладу нам знадобиться допоміжне програмне забезпечення що буде

генерувати початкові маршрути для кожного вузла. Для опису роботи програмного забезпечення візьмемо приклад: чотири міста. Всі можливі варіанти подорожі зображені на (рис. 4.2).

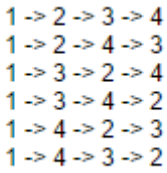

Рисунок 4.2 – Варіантимаршрутів

Післярозбиттявсіхможливихмаршрутівпрограмавиявляєпідгрупи, такщокожномузвузлівдістанетьсяокремийнабіршляхів. Дляприкладузтрьохвузліврозподіленнязображенена (рис. 4.3).

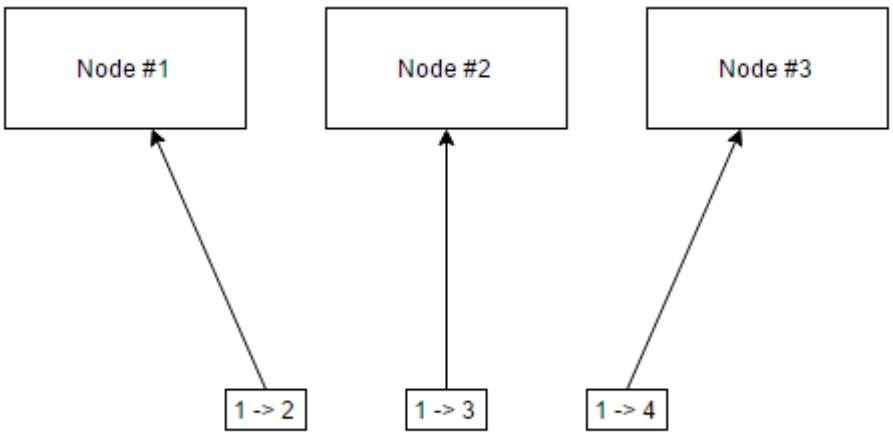

Рисунок 4.3 – Розподілення інформації

Після наданих даних кожен вузол вже самостійно згенерує залишки шляхів. Так, вузол №1 додасть маршрути 1 -> 2 -> 3 -> 4 та 1 -> 2 -> 4 -> 3, вузол №2 маршрути 1 -> 3 -> 2 -> 4 та 1 -> 3 -> 4 -> 2, вузол №3 маршрути 1 -> 4 -> 2 -> 3 та 1 -> 4 -> 3 -> 2. Після цього буде проведене обчислення ціни

кожногу шляху, обирання кращого та відправлення даних до головного вузла. Той вже буде обирати серед них найкращий.

4.3 Оцінка результатів

Для того щоб мати змогу об’єктивно оцінити результати, зробимо задачі обчислювально складними, з великою кількостю даних та приблизним часом роботи у декілька хвилин. Основним критерієм за яким буде оцінена робота системи – це час. Буде розроблене програмне забезпечення на мові програмування Java,що буде слідувати прямому алгоритму, від так буде служити точкою порівняння. Ця програма буде написана за концепцією однопоточного програмування.

Для того щоб зовнішні фактори не впливали на чистоту тесту, обидві програми будуть виконуватися на віртуальній операційній системі встановленій на одному і тому ж персональному комп’ютері, а вимір часу буде проводитися за допомогою вбудованого інструментарію фреймворку та віртуального середовища виконання програм JRE.

Результатом порівняння буде 3 таблиці для кожного з тестових прикладів де будуть записані часи розрахунків для двох концепцій та різної кількості даних. Необхідну кількість вхідних даних будемо обирати таким чином: однопоточна пряма програма на Javaповинна виконувати завдання максимум за годину, часові характеристики для всіх інших значень повинні починатися в проміжку від секунди.

Для більшої точності кожний тест виконувався десять разів, а результат рахувався як середнє арифметичне серед них.

4.4 Порівняльна таблиця

Таблиця 4.1 – Генерація простих чисел

Обмежуюча кількість	Прямолінійний спосіб, с	Розподілення обчислень, с
10	0.821	16
100	0.582	17
1 000	0.803	16
10 000	0.761	18
100 000	6.103	24
1 000 000	15.841	28
10 000 000	129.17	132
100 000 000	1554.088	1273

Таблиця 4.2 – Пошук чисел Серпінського

Обмежуюча кількість, N	Прямолінійний спосіб, с	Розподілення обчислень, с
10	0.000	13
100	0.002	12
1 000	0.07	13
10 000	0.005	13
100 000	0.020	12
1 000 000	0.052	14
10 000 000	0.088	15
100 000 000	0.532	20

Таблиця 4.3 – Задача комівояжера

Обмежуюча кількість, N	Прямолінійний спосіб	Розподілення обчислень
8	0.082	14
9	0.138	15
10	0.757	20
11	6.461	26
12	76.061	54
13	893.979	89

4.5 Графіки порівнянь

Далі наведені графіки порівнянь двох методів.

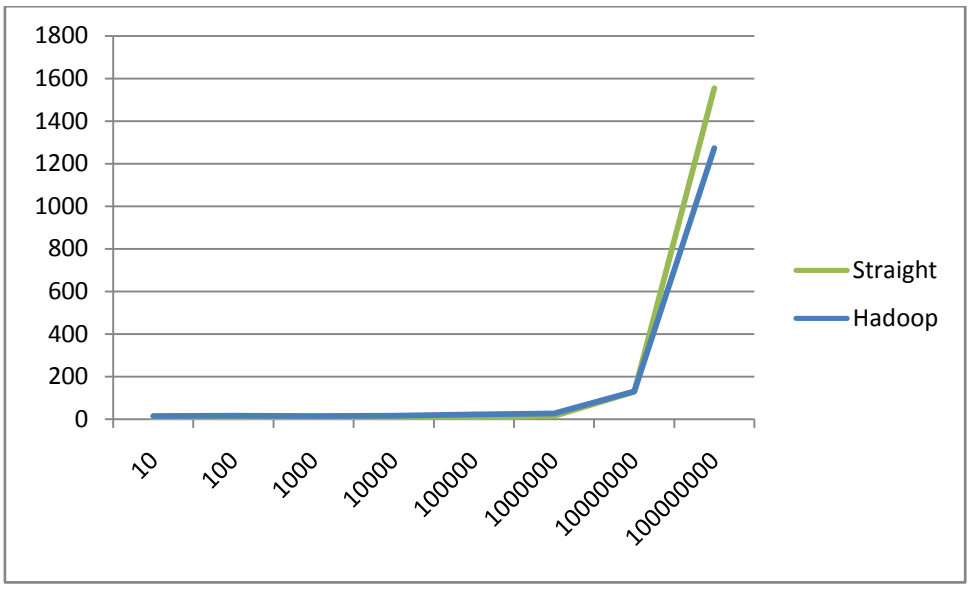

Рисунок 4.4 – Час виконання першої задачі

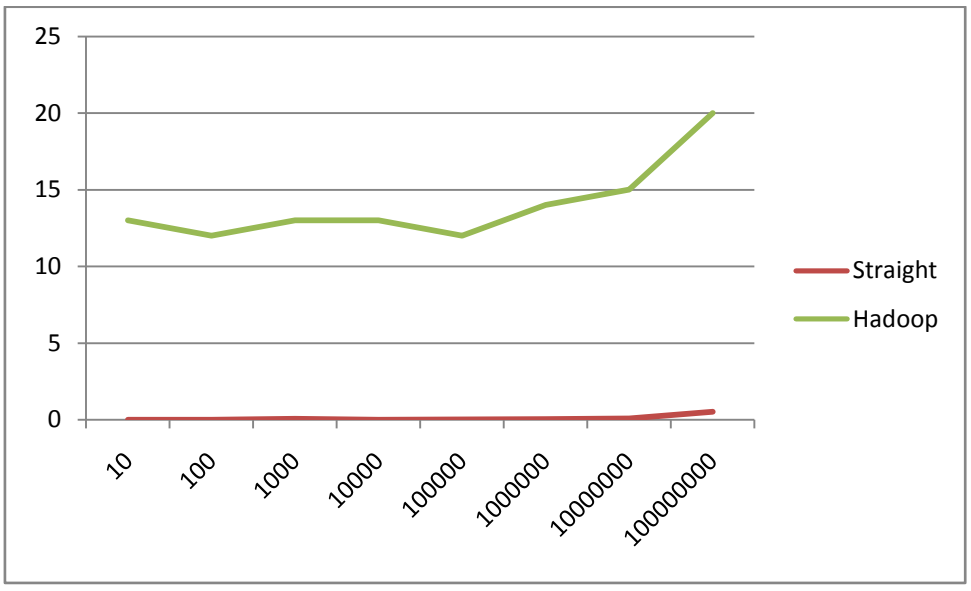

Рисунок 4.5 – Час виконаннядругої задачі

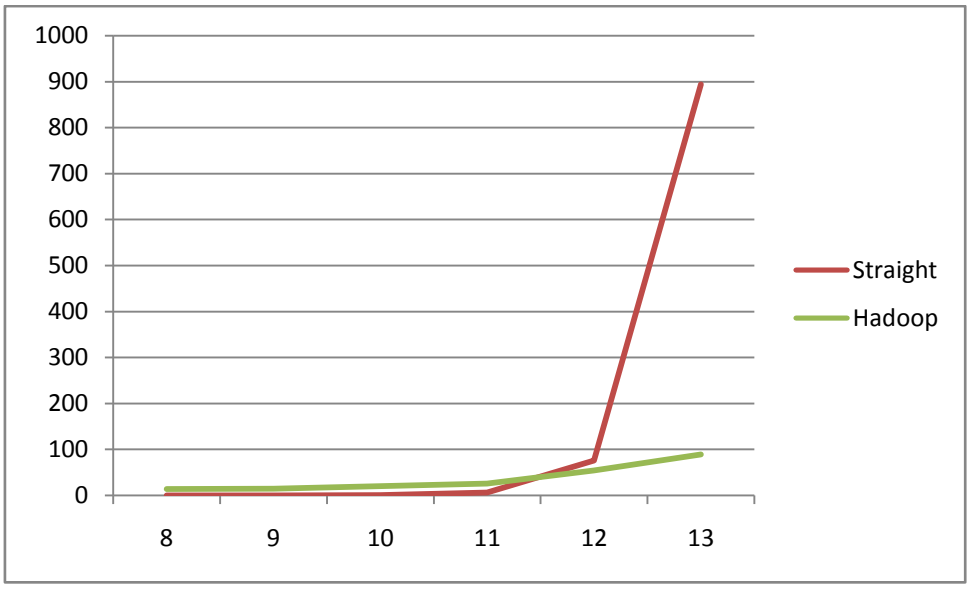

Рисунок 4.6 – Час виконаннятретьої задачі

4.6 Висновки

В даному розділі проводиться аналіз отриманих результатів, а саме порівняння двох концепцій програмних додатків – з та без підтримання фреймворку HADOOP.

Виявилося, що досліджувана платформа дуже добре поводить себе в середовищі великих обчислень та задач із об’ємними вхідними або вихідними даними.

5. ОХОРОНА ПРАЦІ

5.1 Вступ

Охорона праці - це система, яка створена для забезпечення безпеки життя і здоров'я працівників під час їх трудової діяльності. Вона включає в себе правові, організаційно-технічні, санітарно-гігієнічні, соціально-економічні, лікувально-профілактичні, реабілітаційні та інші заходи.

Проектування виробничих об'єктів, розробка нових технологій, засобів виробництва, засобів колективного та індивідуального захисту працюючих повинні провадитися з урахуванням вимог щодо охорони праці.

Виробничі будівлі, споруди, устаткування, транспортні засоби, що вводяться в дію після будівництва або реконструкції, технологічні процеси повинні відповідати нормативним актам про охорону праці.

5.2 Аналіз умов праці

У даній дипломній роботі розроблений програмний продукт для пошуку нових чисел Серпінського.

Створене програмне забезпечення буде застосовуватися в кабінеті відповідної установи, завданням якої є складання таблиці простих чисел за допомогою Hadoop. В даному кабінеті має знаходитись одна людина, тому згідно з [30] розмір площі для одного робочого місця оператора персонального комп’ютера має бути не менше 6 кв. м, а об’єм — не менше 20 куб. м. Отже, приміщення з розмірами: ширина = 3м, довжина = 3м, висота = 3м цілком відповідає зазначеним нормам. Для можливості роботи з даною програмою в приміщенні встановлено робочий стіл з комп’ютером та шафу з документами для зручності.

Задля дотримання визначено рівня мікроклімату встановлено систему опалення та кондиціонер. Для забезпечення потрібного рівного освітленості кімната має вікно та систему загального рівномірного освітлення, що встановлена на стелі. Для дотримання вимог пожежної безпеки встановлено порошковий вогнегасник та систему автоматичної пожежної сигналізації. Схема приміщення зображена на (рис. 4.1).

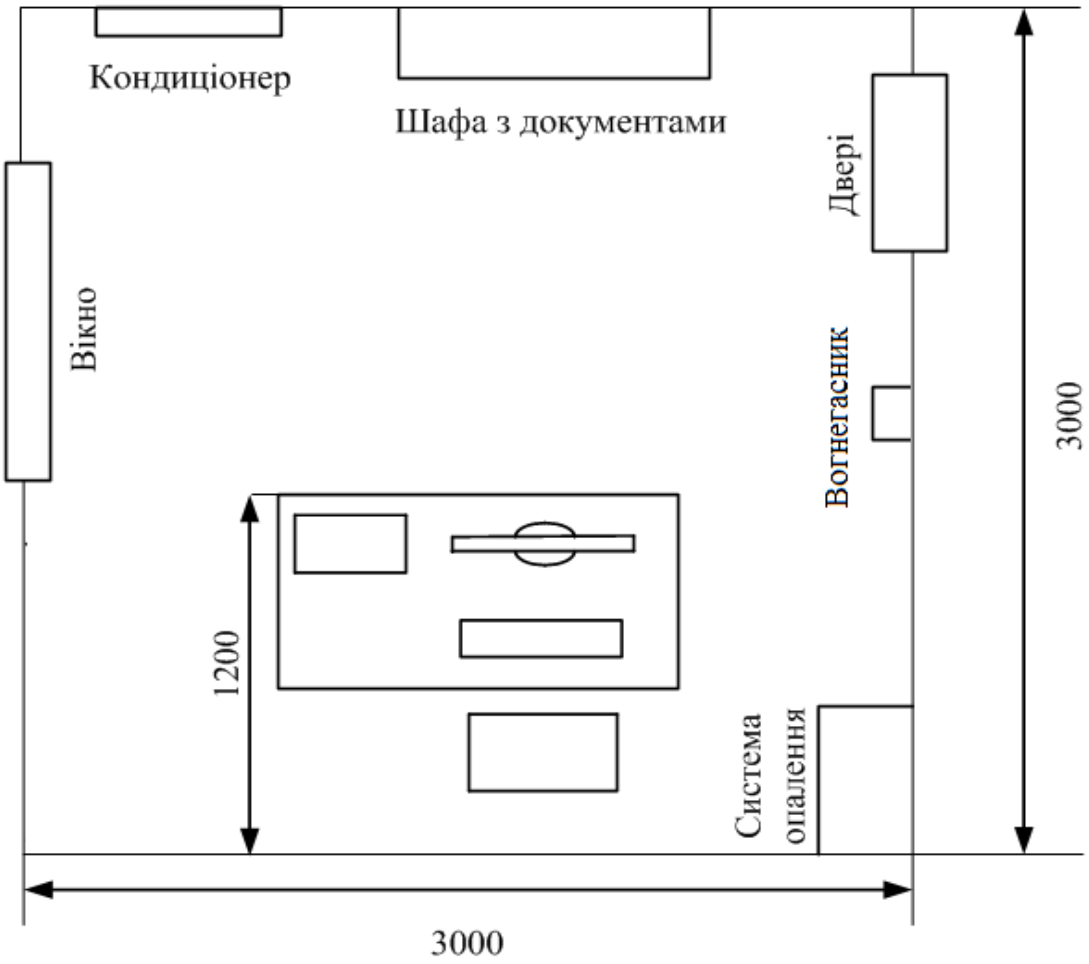

Рисунок 4.1 - Схема приміщення

5.3 Аналіз шкідливих та небезпечних чинників

При використанні під час роботи обчислювальної техніки слід дотримуватись правил безпеки та користування, оскільки вона має певні шкідливі та небезпечні чинники.

Небезпечні чинники – це чинники, у результаті яких людина може втратити працездатність або навіть втратити своє життя.

Шкідливі чинники можуть призвести до погіршення стану здоров'я людини, а в результаті довгого впливу до виникнення професійних захворювань.

До таких чинників відносяться:

- підвищений рівень шуму, вібрації, ультра- та інфразвука;
- недостатня освітленість робочої зони;
- підвищена чи знижена температура, вологість і рухомість повітря та інші;

Розглянемо заходи, які мають бути зроблені для підвищення комфорту праці

5.3.1 Шум та вібрація

Рівні звукового тиску в октавних смугах частот, рівні звуку та еквівалентні рівні звуку на робочих місцях мають відповідати [29].

Устаткування, що становить джерело шуму (АЦП, принтери тощо), слід розташовувати поза приміщеннями, де знаходяться робочі місця. Для забезпечення допустимих рівнів шуму на робочих місцях слід застосовувати засоби звукопоглинання, вибір яких має обґрунтовуватись спеціальними інженерно-акустичними розрахунками.

Значення характеристик вібрації на робочих місцях мають не перевищувати допустимі відповідно до [29].

У приміщенні, що розглядається, рівень шуму 50 Дб, що відповідає нормам [29].

5.3.2 Освітленість

Приміщення, в яких встановлені персональні комп’ютери, повинні мати природне та штучне освітлення відповідно до [28].

Природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи північний схід і забезпечувати коефіцієнт природною освітленості (КПО) не нижче ніж 1,5%.

Штучне освітлення в приміщеннях з робочими місцями має здійснюватись системою загального рівномірного освітлення. У разі переважної роботи з документами, допускається застосування системи комбінованого освітлення (крім системи загального освітлення додатково встановлюються світильники місцевого освітлення). Зазначення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300-500лк. Якщо ці значення освітленості неможливо забезпечити системою загального освітлення, допускається використовувати місцеве освітлення. При цьому світильники місцевого освітлення слід встановлювати таким чином, щоб не створювати відблисків на поверхні екрана, а освітленість екрана має не перевищувати 300лк. Як джерела світла в разі штучного освітлення мають застосовуватись переважно люмінесцентні лампи типу ЛБ. У разі влаштування відбитого освітлення у приміщеннях, де переважним чином працюють з документами, допускається застосування металогалогенних ламп потужністю 250Вт. Допускається застосування ламп розжарювання у світильниках місцевого освітлення. Система загального освітлення має становити суцільні або переривчасті лінії світильників, розташовані збоку від робочих місць (переважно ліворуч), паралельно лінії зору працюючих [31].

Отже в даному приміщенні для можливості забезпечення рівня освітленості 500лк має бути встановлено дві люмінесцентні лампи потужністю 2-3 Вт, причому вони мають бути встановлені ліворуч від робочих місць, паралельно зору робітника.

Для забезпечення нормованих значень освітленості у приміщеннях з ВДТ ЕОМ та ПЕОМ слід чистити шибки і світильники принаймні двічі на рік і вчасно замінювати лампи, що перегоріли.

5.3.3 Мікроклімат

Оскільки в дане приміщення призначене для роботи, що виконуються сидячи і не потребує фізичного напруження, то для нього відповідає категорія робіт Іа.

Дане приміщення має бути обладнане системами опалення, кондиціонування повітря, або припливно-витяжною вентиляцією. У приміщеннях на робочих місцях мають забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря у відповідності до [27]. Рівні позитивних і негативних іонів у повітрі мають відповідати [27]. Оскільки категорія робіт в дане відповідає категорії робіт Іа, то оптимальні значення для температури, відносної вологості й рухливості повітря такі [31]:

Таблиця 5.1 – Норми мікроклімату робочої зони об’єкту

Період року	Категорія робіт	Температура C^0	Відносна вологість %	Швидкість руху повітря, м/с
Холодна	легка-1 а	22 - 24	40 – 60	0,1
Тепла	легка-1 а	23 - 25	40 – 60	0,1

Оптимальна кількість іонів має бути 1500 – 3000 n+ та 3000 – 5000 n-.

У приміщенні, що розглядається, температура повітря знаходиться на рівні 24 градуси за шкалою Цельсія, вологість – 55%, а швидкість руху повітря

становить 0,1 м/с. Кількість іонів 1600 n+ та 3500 n-. Отже, параметри мікроклімату відповідають оптимальним показникам.

Для підтримки допустимих значень мікроклімату та концентрації позитивних та негативних іонів необхідно передбачати установки або прилади зволоження та/або штучної іонізації, кондиціювання повітря.

5.3.4 Пожежна безпека

Дане приміщення до категорії В. Це обумовлено тим, що в приміщенні знаходяться тверді горючі та важкозаймисті речовини та матеріали. Приміщення, у яких розташовуються ЕОМ, повинні мати не нижче II ступеня вогнестійкості. Для гасіння пожеж в офісних приміщеннях слід використовувати порошкові вогнегасники, так як вони є універсальними. заземлені конструкції, що знаходяться в приміщеннях, де розміщені робочі місця (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном), мають бути надійно захищені діелектричними щитками або сітками з метою недопущення потрапляння працівника під напругу [31].

Дане приміщення має бути оснащено системою автоматичної пожежної сигналізації, мати 1 вогнегасник ВП-5 із зарядом вогнегасної речовини 8-12 кг, відповідно до вимог чинного законодавства України. Проходи до засобів пожежогасіння мають бути вільними.

5.4 Висновки

Отже, в результаті проведеної роботи було зроблено аналіз умов праці, шкідливих та небезпечних чинників, з якими стикається робітник. Було визначено яким повинне бути приміщення для функціонування програмного продукту написаного в дипломній роботі, описано які заходи потрібно зробити для того, щоб дане приміщення відповідало необхідним нормам і було комфортним для робітника.

Була наведена схема, розміри приміщення та наведено значення температури, вологості й рухливості повітря, оптимальної кількості іоноів, необхідна кількість і потужність ламп та інші параметри значення яких впливає на умови праці робітника.

ВИСНОВКИ

Дипломна робота присвячена задачі розподілення обчислень у складних системах для вирішення комплексних задач.

У результаті виконання даної дипломної роботи було отриманого програмний продукт, за допомогою якого можна дослідити роботу системи розподілених обчислень HADOOP.

Для розробки програмного забезпечення було розроблено іноваційний алгоритм генерації таблиць та розподілення задач між вузлами, роботу яких було проаналізовано та виведені результати в четвертому розділі.Виявилося, що фреймворк в цілому справляється з поставленою задачею, але є такі випадки або такі задачі де використання більш простої системи буде доцільнішим. Також, є алгоритми де HADOOPв зв’язку із специфічним алгоритмом роботи платформи на рівень ефективніше виконує програму.

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

Другий розділ було присвячено дослідженню існуючих методів для розв’язання проблеми, побудовано математичну модель, описано за якими критеріями визначалась якість результату роботи системи, наведено алгоритм роботи програми для тестуваня даної системи.

У третьому розділі було обґрунтовано вибір платформи та мови реалізації програмного продукту, було зроблено аналіз архітектури системи. Крім того, були розроблені експерименти за допомогою яких можна було проводити аналіз результатів, отриманих в роботі.

У четвертому розділі проводився аналіз отриманих результатів, а саме порівняння двох концепцій програмних додатків – з та без підтримання фреймворку HADOOP.

П’ятий розділ був присвячений аналізу умов праці для виробничого приміщення, у якому операторами експлуатуються програмний продукт.

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. Богданов В.Л.Концепція програми наукових досліджень НАН України. / Богданов В.Л.. - К. : Президія НАН України, 2012. -7 с.
5. Паралельні бази даних. — Режим доступа : http://www.simulation.kiev.ua/dbis/lection27.html. — Дата доступа : 26.04.15.
6. Кобець О. І. Моделі і методи розподілу ресурсів в системах, побудованих на хмарних обчисленнях. / Кобець О. І.// Міжнародна наукова інтернет-конференціяСоціум. Наука. Культура., - К.: Національний технічний університет України "Київський політехнічний інститут", 2012- С. 28-33.
7. Розподілені Обчислення— Режим доступа : http://www.simulation.kiev.ua/dbis/lection27.html. — Дата доступа : 01.05.15.
8. WhatisthepurposeofEinstein@Home?— Режим доступа : http://einstein.phys.uwm.edu/PartialS3Results/node3.html — Дата доступа : 01.05.15.
9. Пользователей интернета пригласили подтвердить теорию Эйнштейна— Режим доступа : http://lenta.ru/news/2005/02/21/einstein/ — Дата доступа : 01.05.15.

10. Holger J. PletschDeepest All-Sky Surveys for Continuous Gravitational Waves. / Holger J. Pletsch. –Hannover. : Leibniz Universität Hannover, July 25, 2010. – 5 p.
11. Офіційний сайт PrimeGrid. — Режим доступа : http://www.primegrid.com. — Дата доступа : 26.04.15.
12. Prime Number. — Режим доступа : http://mathworld.wolfram.com/PrimeNumber.html. — Дата доступа : 26.04.15.
13. Séroul R. "The Sieve of Eratosthenes." /Séroul, R. // Programming for Mathematicians. - Berlin: Springer-Verlag, 2000, pp. 169-175.
14. Arnault F. "Rabin-Miller Primality Test: Composite Numbers Which Pass It." / Arnault F. // Math. Comput. - New York,1995 355-361.
15. Richard Crandall and Carl Pomerance. Prime Numbers: A Computational Perspective. / Richard Crandall and Carl Pomerance. – Springer : 2005. p.159–190.
16. Richard Crandall and Carl Pomerance. Prime Numbers: A Computational Perspective. / Richard Crandall and Carl Pomerance. – Springer : 2005. p.334–340.
17. ThePrimeGlossary — Режим доступа : http://primes.utm.edu/glossary/home.php. — Дата доступа : 02.05.15.
18. Arora Sanjeev and Barak Boaz. Computational Complexity – A Modern Approach. / Arora Sanjeev and Barak Boaz. – Cambridge: 2009.
19. Andrews, Gregory R. Foundations of Multithreaded, Parallel, and Distributed Programming. / Andrews, Gregory R. – Addison–Wesley: 2000. p. 25-26
20. Andrews, Gregory R. Foundations of Multithreaded, Parallel, and Distributed Programming. / Andrews, Gregory R. – Addison–Wesley: 2000. p. 87-90
21. Andrews, Gregory R. Foundations of Multithreaded, Parallel, and Distributed Programming. / Andrews, Gregory R. – Addison–Wesley: 2000. p. 140-142

22. Lind P. and Alm M.A. database-centric virtual chemistry system. / Lind P. and Alm M.A. – Huddinge:2006. p. 46
23. Develop C# Hadoop streaming programs for HDInsight— Режим доступа : https://azure.microsoft.com/en-us/documentation/articles/hdinsight-hadoop-develop-deploy-streaming-jobs/ — Дата доступа : 06.05.15.
24. Scott Guthrie: Silverlight and the Cross-Platform CLR— Режим доступа : https://channel9.msdn.com/shows/Going+Deep/Scott-Guthrie-Silverlight-and-the-Cross-Platform-CLR — Дата доступа : 06.05.15.
25. YaremenkoV., KachkoN. Primalitytestproblem/ YaremenkoV., KachkoN. : Матеріали XIV всеукраїнської науково – практичної студентської конференції, 07квітня 2015, Київ, Україна : матеріали. – К. : НТУУ «КПІ», 2015. – С. 136.
26. KachkoN.Primetablegeneration / KachkoN.: міжнародна науково-технічна конференція «САІТ-2015», 23 червня 2015, Київ, Україна : матеріали. – К. : НТУУ «КПІ», 2015.
27. Санітарні норми мікроклімату виробничих приміщень : ДСН 3.3.6.042-99. – [Чинний від 2000-01-01]. – К. : МОЗ України, 2000. – 42 с. – (Національні стандарти України).
28. Природнє і штучне освітлення : ДБН В.2.5-28:2006 – [Чинний від 2015-01-01]. – К. : Міністерство будівництва, архітектури та житлово-комунального господарства України, 2015. – 171 с. – (Національні стандарти України).
29. Санітарні норми виробничого шуму, ультразвуку та інфразвуку : ДСН 3.3.6.037-99. – [Чинний від 2000-01-01]. – К. : МОЗ України, 2000. – 37 с. – (Національні стандарти України).
30. Правила охорони праці під час експлуатації електронно-обчислювальних машин : НПАОП 0.00.-1.31-10. – [Чинний від 2010-03-26]. – К. : Держнаглядохоронпраці України, 2010. – 7 с. – (Національні стандарти України).

31. Охорона праці в офісі. Вимоги до робочого місця офісного працівника – Режим доступу:http://gc.ua/business-news/oxorona-praci-v-ofisi-vimogi-do-robochogo-miscya-ofisnogo-pracivnika/— Дата доступа : 06.05.15.

ДОДАТОК А

ТЕКСТ ПРОГРАМИ

Primes.java - Hadoopversion

```
import java.io.IOException;
import java.util.regex.Pattern;
import java.math.*;
import java.math.BigInteger;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import org.apache.log4j.Logger;

public class Primes extends Configured implements Tool {

  private static final Logger LOG = Logger.getLogger(Primes.class);

  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Primes(), args);
    System.exit(res);
  }

  public int run(String[] args) throws Exception {
    Job job = Job.getInstance(getConf(), "primes");
    job.setJarByClass(this.getClass());
    // Use TextInputFormat, the default unless job.setInputFormatClass is used
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
```

```
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    return job.waitForCompletion(true) ? 0 : 1;
   }

   public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    private long numRecords = 0;
    private static final Pattern WORD_BOUNDARY = Pattern.compile("\\s*\\b\\s*");

    public void map(LongWritable offset, Text lineText, Context context)
        throws IOException, InterruptedException {
      String line = lineText.toString();
      Text currentWord = new Text();
      for (String word : WORD_BOUNDARY.split(line)) {
       if (word.isEmpty()) {
           continue;
       }
           BigInteger i = new BigInteger(word.toString());
           BigInteger boundary = new BigInteger("100000000");
           for (; i.compareTo(boundary) < 0;
i=((i.add(BigInteger.ONE)).add(BigInteger.ONE)).add(BigInteger.ONE))
           {
                  if (i.isProbablePrime(500))
                 context.write(new Text(i.toString()), one);
             }
       }
    }
   }

   public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    public void reduce(Text word, Iterable<IntWritable> counts, Context context)
        throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable count : counts) {
       sum += count.get();
      }
      context.write(word, new IntWritable(sum));
    }
   }
}
```

Main.java - straightforward version

```
import Interface.General_Ops;

import java.util.ArrayList;
import java.math.BigInteger;

public class Main {
	static ArrayList<BigInteger> Numbers;
	static BigInteger current, max, divisor, hundred;
	static long tStart,tEnd;
	static double elapsedSeconds;
	public static void main(String [] args)
			throws Exception{
		Numbers=new ArrayList<BigInteger>();
		max=new BigInteger("10000000");
		current=new BigInteger("2");
		hundred=new BigInteger("100");
		divisor=max.divide(new BigInteger("100"));
		//current=BigInteger.ONE;
		tStart = System.currentTimeMillis();
		while (current.compareTo(max)!=1)
		{
			if (General_Ops.IsPrime(Numbers, current))
				Numbers.add(current);
			current=current.add(BigInteger.ONE);
			if (current.mod(divisor).compareTo(BigInteger.ZERO)==0){
				//Runtime.getRuntime().exec("cls");

	System.out.println(current.multiply(hundred).divide(max).toString());
			}
		}
		tEnd = System.currentTimeMillis();
		elapsedSeconds = (tEnd-tStart) / 1000.0;
		//General_Ops.Out(Numbers);
		General_Ops.OutLast(Numbers);
		General_Ops.SaveNumbers(Numbers,"list.txt");
		System.out.println("Elspsed time: "+elapsedSeconds+"ms");
	}

}
```