РЕФЕРАТ

Дипломна робота: 60 ст. , 9 рис., 40 табл., 16 джерел та ? додатки.

Метою данної роботи є побудувати симуляційну систему для емуляції процесу паралельного множення матриць шляхом розбиття їх на блоки. Другою частиною роботи є аналіз та розробка математичної моделі процесу та переверки моделі за допомоги побудованої симуляційної системи. Також запропоновані альтернативні методи вирішення проблеми.

Результати роботи:

- проаналізована модель для статичного планувальника типу minmin
- розглянуті планувальники minmax, maxmin та maxmax як подібні до minmin.
- розроблена симуляційна система для емуляції процесу множення матриць поблочно.
- розглянуті інші підходи до вирішення проблеми: числові методи оптимізації, машинне навчання тощо.

Результати данної роботи можна використати при розробці системи розподілених обчислень з метою мінімізації часу виконання парелельних задач в умовах можливості регулювання складностей задач. Також на базі цієї роботи можна проаналізувати більш складні планувальники.

ПЛАНУВАЛЬНИКИ, МНОЖЕННЯ МАТРИЦЬ, ТЕОРІЯ ІГОР, ХМАРНІ ОБЧИСЛЕННЯ.

ABSTRACT

The thesis: 60 p. , 9 fig., 40 tabl., 16 sources and ? appendices.

The theme of this thesis is “Game theoretic analysis of schedulers in heterogeneous multiprocessor environment”.

The purpose of this work is to construct a simulation system to emulate the process of parallel multiplication of matrices by splitting them into blocks. The second part of the work is the analysis and development of the mathematical model of the process and revaluation of the model with the help of the built simulation system. Alternative methods of solving the problem are also proposed.

Thesis results:

- The model for a static scheduler of type minmin is analyzed
- Considered the planners minmax, maxmin and maxmax as similar to minmin.
- A simulation system was developed to emulate the process of multiplying matrices by block.
- other approaches to the problem are considered: numerical methods of optimization, machine learning, etc.

The results of this work can be used in the development of a distributed computing system in order to minimize the time of parelle tasks in the conditions of the ability to control the complexity of tasks. Also, based on this work can be analyzed more complex schedulers.

SCHEDULERS, MATRIX MULTIPLICATION, GAME THEORY, CLOUD COMPUTING.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

СПЗ - симуляційне програмне забезбечення

ОВ - обчислювальний вузол

ВСТУП

Хмарні обчислення на даний момент вважається чимось достатньо простим да тоступним, деякі компанії навіть дозволяють безкоштовно використовувати певну кількість ресурсів, також університетам надаються ресурси для складних обчислень, які просто не можливо провести за допомоги звичайних комп'ютерів.

Планувальник у хмарних обчисленнях це те, що управляє самим процесом розподілення задач на обчислювальних вузлах, які виділені для розподіленої системи чи хмари. Саме від стратегії розподілення задач залежить як швидко користувачі отримають результати, наскільки справедливо буде розподілений час обчислень між багатьма користувачами та інші параметри.

Чимало досліджень виконано з метою проектування найефективнішого планувальника, проте такого ще не існує. Частіше за все для кожного з планувальників можна знайти перелік параметрів які він оптимізує. І все ще немає такого планувальника, який буде кращим за інший по всім параметрам та бути справедливим у середовищі з багатьма користувачами.

Основною метою цієї роботи є не розробка найкращого планувальника, а аналіз звичайних планувальників та характеристик задач, які будут краще всього розкривати потенціал вибраного планувальника. Також у роботі проведений ігровий аналіз процесу обчислення добутку двох матриць у розподіленому середовищі як гри двох користувачів із різними стратегіями розрізання матриці на блоки для паралельного обчислення добутку. Особливістю такої проблеми є те, що сам процес множення матриць при їх великих розмірах може займати дуже багато часу і тому в першчу чергу потрібно розробити симуляційне програмне забезпечення для більш швидкого аналізу моделей та алгоритмів.

РОЗДІЛ 1 АНАЛІЗ ЗАДАЧІ ПЛАНУВАННЯ У ХМАРНОМУ СЕРЕДОВИЩІ

1.1 Планувальники у розподілених обчисленнях

Планувальники в загальному поділяються на два типи: статичні та динамічні. Статичні мають певне правило розподілу задач на обчислювальні вузли та правило не зміюється під час роботи планувальника. Динамічні планувальники в свою чергу можуть адаптуватися під час роботи та змінювати стратегії планування. Хоч динамічні планувальники і здаються більш універсальними, проте вони дуже складні для аналізу та часто розробляються з метою оптимізації певного параметра.

Основною метою планування є розподіл виконання задач, які надсилають користувачі, між обчислювальними вузлами. Планувальник формує чергу виконання задач, та визначає яку задачу на якому вузлі стартувати. Частіше за все користувачів хвилює лише мінімізація часу завершення деякої відправленої множини задач у хмару.

Структура хмари у загальному випадку описується за допомоги таких сутностей як:

- брокер
- планувальник
- дата центр
- хост
- віртуальна машина

Брокер являєься посередником між користувачем та хмарою і саме він формує запити до хмари на виконання задач та отримує результати їх виконання. Планувальник отримує задачі від брокерів різних користувачів та має свою певну стратегію розподілу черги задач, саме він керує дата центрами. Дата центри це несамостійні сутності і повністю керовані планувальниками. Їх основна мета - надавати обчислювальні вузли, запускати задачі по команді від планувальника та віддавати результати. Також у випадку динамічної хмари по певним запитам вони можуть збільшувати чи зменшувати кількість обчислювальних вузлів у мережі. Віртуальна машина - це окремий обчислю-

вальний вузол, який вміє лише отримувати вхідні дані, виконувати задачу та відправляти результат у дата центр.

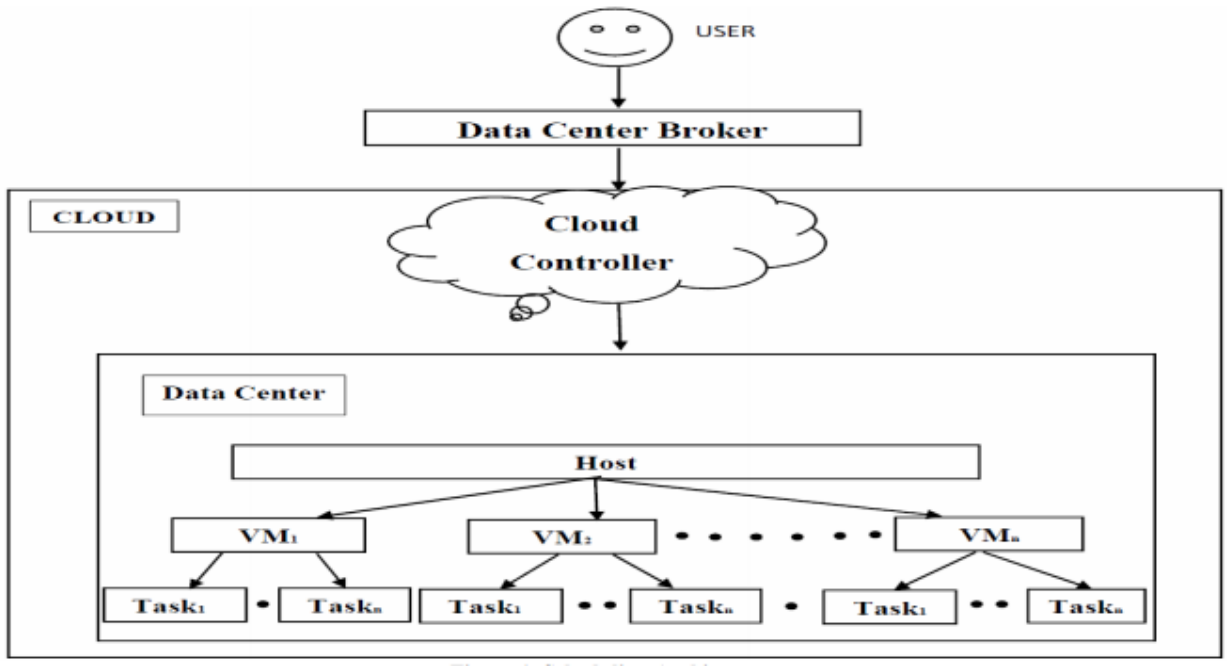


Рисунок 1.1 – Ілюстрація структури хмарної обчислювальної мережі

Існує дві політики планування: space shared та time shared. У політиці space shared задачі у черзі розділяють між собою логічні ядра процесора обчислювального вулза і у випадку якщо усі ядра зайняті, то задач чекає звільнення ядра. Саме ця політика і вікористовується для тестування алгоритмів планування оскільки у випадку коли обчислювальні вузли мають лише одне ядро процесора, задачі виконуються послідовно від початку до кінця. Найпростіший алгоритм, який можна зустріти разом з політикою space shared - first come first served (FCFS).

Етапи роботи space shared політики:

Крок 1 Формується черга задач

Крок 2 Запланувати виконання наступної задачі із черги

Крок 3 При завершенні задачі відправити результат

Крок 4 Якщо черга непорожня, то повернутися на крок 2

Крок 5 Кінець

*** Нові отримані задачі просто додаються у чергу і вони чекають свого виконання

Time shared політика в свою чергу означає що задачі у чергі розділяють між собою процесорний час. Тобто не обчислювальні ресурси, а проміжки часу які будуть витрачені на виконання кожної із задач. Усі задачі в черзі стартують в один і той самий час. Задача планувальника в цьому випадку - вирішувати коли потрібно призупинити виконання одної задачі та стартувати чи продовжити виконання іншої задачі. На перший погляд ця політика значно краща оскільки дозволяє виконувати набір задач поступово, проте час переключення від одної задачі до другої також потрібно враховувати. А також щоб стартувати усі задачі потрібно мати вхідні дані для усіх задач, що також не завжди зручно. Разом з цією політикою часто можна зустріти посилання на алгоритм планування Round Robin.

Етапи роботи tune shared політики:

Крок 1 Формується черга задач.

Крок 2 Усі задачі запускаються одночасно у режимі переключення між задачами за правилом, яке визначає планувальник

Крок 3 Кінець

*** Нові отримані задачі просто додаються у чергу і зразу запускаються та працюють у режимі спільного використання процесорного часу

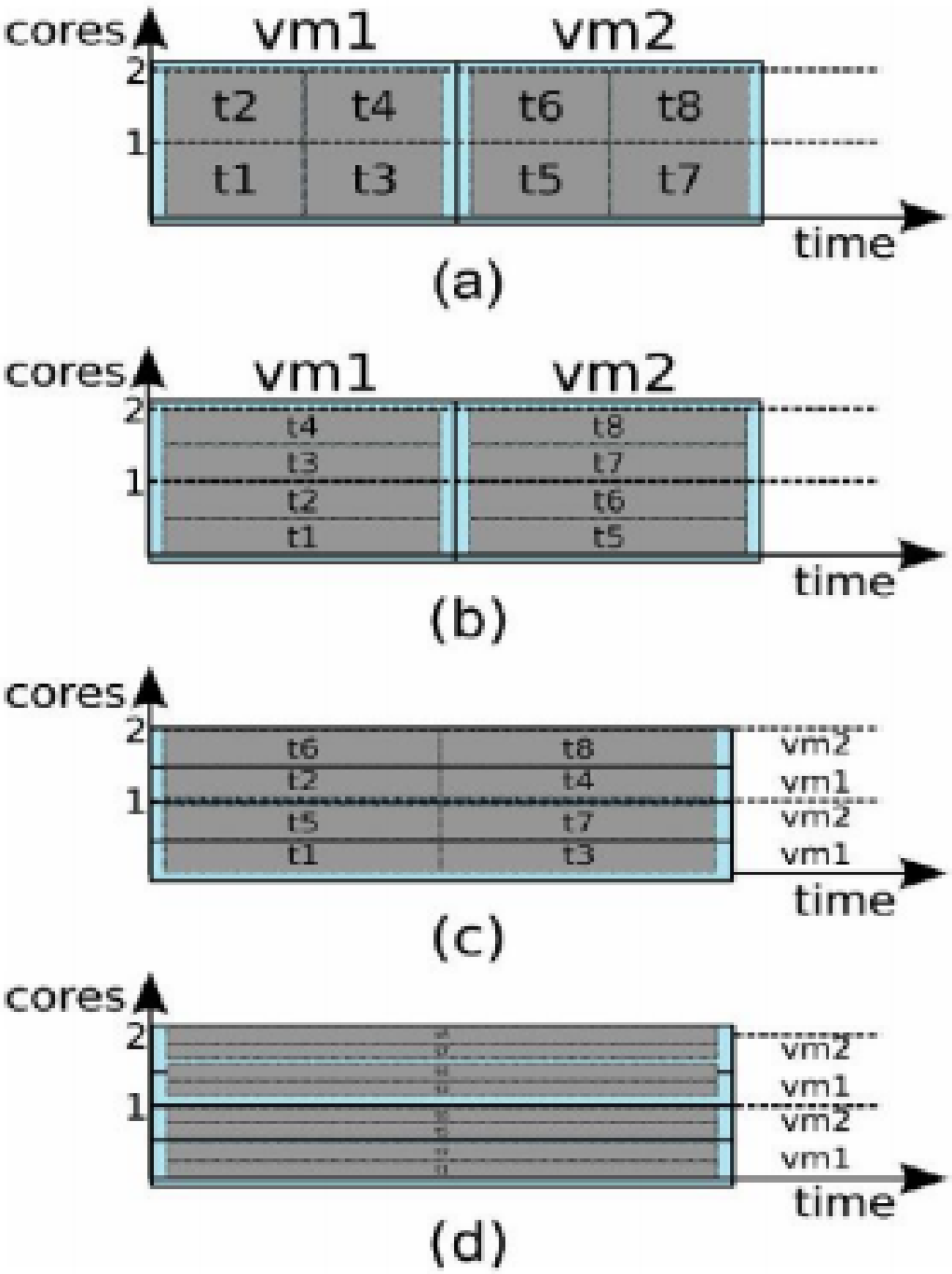


Figure 2. Effects of scheduling polices on task execution: (a) Space-shared for VMs and Tasks, (b) Space-share for VMs and Time-shared for tasks, (c) Time-shared for VMs, Space-shared for tasks, and (d) Time-shared for both VMs and Tasks [7]

Рисунок 1.2 – Ілюстрація планування задач для time та space shared

1.2 CloudSim як засіб для симуляції хмарних обчислень

Останнім часом технологія хмарних обчислень виникла як провідна технологія для забезпечення надійних, безпечних, відмовостійких, стійких та масштабованих обчислювальних послуг, які представлені як програмне забезпечення, інфраструктура або платформа як послуги (SaaS, IaaS, PaaS). Більше того, ці послуги можуть бути запропоновані в приватних центрах обробки даних (приватні хмари), можуть бути комерційно запропоновані для клієнтів (загальнодоступні хмари), або можливо, що як державні, так і приватні хмари об'єднуються в гібридні хмари.

Ця вже широка екосистема хмарних архітектур, разом із зростаючим попитом на енергоефективні IT-технології, вимагає своєчасних, повторюваних та контрольованих методологій для оцінки алгоритмів, програм і політики до фактичного розвитку хмарних продуктів. Оскільки використання реальних тест-сейфів обмежує експерименти до масштабу тест-сейфів і робить відтворення результатів надзвичайно важким завданням, альтернативні підходи для тестування та експериментів сприяють розробці нових технологій Cloud.

Підходящою альтернативою є використання інструментів моделювання, що дають можливість оцінити хмарну систему перед розробкою програмного забезпечення в середовищі, де можна відтворити тести. Зокрема, у випадку обласного обчислення, коли доступ до інфраструктури здійснює платежі в реальній валюті, підходи на основі моделювання дають значні переваги, оскільки це дозволяє Cloud клієнтам протестувати свої послуги в повторимому та контрольованому середовищі безоплатно, а також настроювати продуктивність вузькі місця до розгортання на реальних хмарах. На стороні постачальника, симуляційні середовища дозволяють оцінити різні види сценаріїв лізингу ресурсів при різному розподілі навантаження та ціноутворення. Такі дослідження могли б допомогти постачальникам оптимізувати вартість доступу до ресурсів з упором на підвищення прибутку. За відсутності подібних імітаційних платформ, клієнти Cloud і постачальники повинні спиратися або на теоретичні та неточні оцінки, так і на підходи щодо спроб і помилок, які

призводять до неефективної ефективності обслуговування та отримання доходу.

Основна мета цього проекту полягає у забезпеченні узагальненої та розширюваної системи моделювання, яка дозволяє безперешкодно моделювати, моделювати та експериментувати з розвиваються інфраструктурою Cloud Computing та додатковими службами. Використовуючи CloudSim, дослідники та розробники на базі промисловості можуть зосередити увагу на конкретних проблемах дизайну систем, які вони хочуть досліджувати, не занепокоєні деталями низького рівня, пов'язаними з інфраструктурою та службами Cloud.

1.2.1 Сильні сторони

CloudSim фреймворк [1] достатньо широко охоплює хмарні системи та їх внутрішню структуру. Саме за його допомоги можна перед проектуванням хмарної інфраструктури спочатку провести симуляції та перевірити адекватність спроектованої архітектури мережі.

Пакет працює на базі подій і усі процеси, що моделюються за допомогою CloudSim, реєструються як події. Також як і запит на створення обчислювального вузла, запит на виконання задач та інші. Це дозволяє додавати нові елементи у симуляційну систему без змін в існуючих файлах, потрібно лише створити свою власну подію та додати її обробку до сутностей, в яких ця подія відбувається. Додаваня обробки події часто виконується через наслідування від основного об'єкта та імплементації метода "processEvent".

Така система дуже гнучка та дозволяє швидко написати свою власну симуляцію.

1.2.2 Слабкі сторони

СПЗ CloudSim написане на мові програмування Java та непридатне для моделювання ситуацій з великою кількістю задач. Це одна із проблем, яка і привела до написання власної упрощеної версії СПЗ на мові C++, яка скоротила час симуляцій майже в 50 разів у порівнянні з модифікованою версією

СПЗ, яка була написана на базі CloudSim та була майже у 150 разів швидшою за звичайну версію.

Також сама по собі мова Java неадекватно працює у випадку коли потрібно швидко створики велику кількість малих об'єктів, провести з ними певні операції та видалити.

Наприклад, для симуляції множення двох матриць $N * N$ та $N * N$, при розмірі розрізання $n = 1$ для одного користувача буде створено $N * N$ задач множення підматриць розмірів $1 * N$ та $N * 1$. Звісно цей приклад не має сенсу оскільки розрізання $n = 1$ скоріше за все не ефективне, проте сама неможливість його змоделювати для $N = 5000$ вважається великим недоліком, оскільки поставивши $N = 50000$ та $n = 10$ отримаємо таку ж саму кількість задач, і лише на їх створення на мові Java витрачається більше 5 секунд. У той час как симуляція, написана на мові C++ дозволяє за ці 5 секунд провести 2 симуляції з такими ж параметрами.

Також слід зазначити, що першою спробою написання симуляції для цієї роботи було саме використання CloudSim. Проте під час розробки програми доводилось дуже довго вивчати документацію і виправляти деякі внутрішні недоліки. Наприклад, у системі була знайдена проблема, що велика кількість малих задач оброблялась повністю і обривалась у випадковому місці. Це пов'язано з тим, що система не розроблялась з метою проведення важких симуляцій.

Також під час проведення симуляцій на виправленій версії системи було помічено дуже повільну швидкість симуляцій для великої кількості задач. Було знайдено 2 основні точки, які сильно сповільнювали програму та виправлені. Одна із правок дала пришвидшення симуляцій приблизно у 50 разів, а друга ще у приблизно 100 разів. Проте навіть з цими правками симуляції проходять значно повільніше за написаний нами самуляційний пакет.

1.2.3 CloudSim Plus

CloudSim Plus [2] це фреймворк, що є удосконаленою версією CloudSim та все ще розвивається, на відміну від свого батька, який не має оновлень з 2016 року. Оскільки пакет розробляється великою кількістю людей у їх

вільний час, то довіра до нього сильно падає. Немає ніякого контролю якості системи, особливо ігнорується швидкодія. Також цей пакет може містити помилки у коді, які виправити сторонньому користувачу буде дуже важко, оскільки для цього протрібно знати та розуміти архітектуру саме цього пакета.

Проте саме цей пакет дозволяє проводити симуляції паралельно і на комп'ютерах з декількома ядрами можна отримати пришвидшення набору симуляцій за рахунок паралельного їх запуску. Саме це і було зроблено у першій версії симуляційного пакету. Але швидкодії всеодно було недостатньо, оскільки бажаним результатом було отримати симуляційну систему, яка швидко зможе проводити симуляції усіх можливих комбінацій стратегій двох гравців. І навіть для розміру матриць 1000 симуляція усіх комбінацій проходила більше години

1.3 Аналіз існуючих робіт

У роботі [3] проведений короткий аналіз стратегій виділення ресурсів типів Min-Min та Max-Min. Також у цій роботі занадто проста схема задач, складність обчислень прямо пропорційна розмірам файлів, що буває дуже рідко для трудоемних задач. Ця робота проводить симуляції, в яких кількість задач не більше 10 та всього 2 обчислювальних вузла. Проте вони запропонували алгоритм вибору, який покращує звичайну реалізацію стратегії виділення ресурсів. Також у цій роботі був використаний пакет CloudSim як основний засіб для аналізу та перевірки теорії.

Також цікавою роботою є використання теорії керування з метою мінімізації часу на виконання задачі заданого набору задач [4]. У цій роботі використовується time shared політика планування та динамічне переключення між виконанням задач. Для цього використовується теорія оптимального керування, яка керує кількістю часу, яка надається для задачі у наступній ітерації. Також доведена еквівалентність задачі планування і задачі оптимального управління часом.

Проблема ефективного множення великих матриць також розглянута у [5]. У цій роботі розглядають більш стандартизовану операцію над матриця-

ми: $C := A * B + C$. Ця операція є основною у бібліотеках стандарту BLAS (Basic Linear Algebra Subprograms), яка в них називається GEMM (General Matrix Multiply) та описується як $C = \alpha A * B + \beta C$. Робота базується в першу чергу на [6] та [7], я яких проводиться аналіз складності вводу та виводу у випадку множення матриць і також можливі оптимізації, які можна застосувати у розподіленому середовищі з метою зменшення затрат на операції передачі даних. Ця проблема також випливає і у нашому дослідженні, оскільки показано, що надлишковість передачі даних при розбитті матриць на блоки дуже швидко зростає, проте найбільш оптимальними розбиттями є саме розбиття на малі блоки так як вони мінімізують простоювання вузлів при завершенні виконання блоку задач. Дослідження у роботі виконані саме для випадку блочного розбиття матриць і не підходять для паралельної версії алгоритма Штрассена.

Одна з свіжих робот на тематику ефективного множення матриць це [8]. В роботі розглядається операція $y := Ax$ для матриці $A \in \mathbb{R}^{m*n}$ та вектора $x \in \mathbb{R}^n$ як частковий випадок множення двох матриць. Побудована модель обчислень також включає дві схеми: Uncoded Load Balanced (ULB) та Coded Equal Allocation (CEA). Особливістю роботи є оцінка оптимальної конфігуляції обчислювального середовища на базі Amazon AWS завдяки побудові моделі. Знаходиться компроміс між вибором набору кластерів із заплопонованих тарифних планів та швидкістю виконання операцій. Для пошуку оптимальної конфігурації запропонований еврестичний пошук. На основі цієї роботи була запропонована покращена схема кодування та проведені порівняння з існуючими [9].

РОЗДІЛ 2 ПОБУДОВА МАТЕМАТИЧНОЇ МОДЕЛІ МНОЖЕННЯ МАТРИЦЬ

2.1 Потокова модель задачі обчислень множення матриць

Нехай система складається з m обчислювальних вузлів та кожен з них характеризується швидкістю роботи $p_i, i = 1, \dots, m$ – тобто кількістю операцій з плаваючою точкою за секунду, які він може здійснити. Процесори з'єднані лініями зв'язку з планувальником, який передає задачі та приймає від них результат. Будемо вважати, що лінії зв'язку ідентичні та мають швидкість передачі даних q та затримку l. Будемо вважати, що виконуються наступні припущення

- Всі процесори починають роботу одночасно
- Планувальник здійснює призначення миттєво

Нехай задані дві квадратні матриці розмірності $N \times N$, результат множення яких необхідно обчислити. При використанні блочного алгоритму користувач задає розмір блоку n , в результаті чого формуються $k = \frac{N^2}{n^2}$ задач, кожна з яких буде мати складність $\mathcal{O}(n^2)$. Припустимо, що планувальник забезпечує пересилку повідомлень на вузли за певним фіксованим алгоритмом, який завершує обчислення за час $T(N, n)$. Тоді задача користувача полягає у пошуку мінімуму функції:

$$T(N, n) \longrightarrow \min \tag{2.1}$$

Функція $T(N, n)$ може мати багато локальних мінімумів в залежності від конфігурації системи. Ілюстрації графіків функції, отриманої шляхом симуляції, наведені у експериментальній частині та корелють з отриманими результатами у [10].

Одним з розповсюджених підходів до аналізу таких задач полягає у дослідженні потокової моделі даного процесу [11].

Припустимо, що користувач вибрав вектор $x \in \mathbb{R}$ з компонентами x_i, де $x_i > 0$, $x_i \leq k$, $i = 1, \dots, m$, $\sum_{i=1}^{k} x_i = k$. Всі таки вектори утворюють множину $X(n)$. Кожен компонент вектора x описує відсоток задач, призначених

для виконання на i-тому процесорі. Будемо брати до уваги тільки операції, множення. Таке спрощення дозволяє у явному вигляді виписати функції часу. Загальний час закінчення залежить від x, та дорівнює:

$$T(x, X(n)) = \max_{i=1,\dots,m} \frac{x_i N n^2}{p_i} \tag{2.2}$$

Потокова модель передбачає можливість розділення задачі на підзадачі розміру $\epsilon = Nn^2$, компонування з них підходящих підзадач та визначення загального часу при $\epsilon \longrightarrow 0$.

Твердження 1. Мінімальний час обчислень для потокової моделі з одним користувачем дорівнює:

$$T = \frac{N^3}{\sum_{i=1}^{m} p_i} \tag{2.3}$$

Згізно з цим користувач має розділити задачі так, щоб вузол i отримав задач з сумарною складністю $p_i * T$ часу.

Функція Мінковського для множини X та вектора $p \in \mathbb{R}^m$ визначається наступним чином:

$$\mu_X(p) = inf\lambda > 0 : p \in \lambda X \tag{2.4}$$

Відомо, що ця функція опукла для опуклої X. Визначимо множину потужностей системи $R = r \in \mathbb{R}^m : r_i \in [0, p_i]$ та масштабуємо її наступним чином:

$$R(n) = \frac{R}{Nn^2} \tag{2.5}$$

Тоді $T(x, X(n)) = \mu_{R(n)}(x)$.

Доведення.

Розглянемо праву частину: $\mu_{R(n)}(x) = inf\lambda > 0 : x \in \lambda R(n)$ Умова належності вектора x множині R записується як $\max\limits_{i=1,\dots,m} \frac{x_i N n^2}{p_i}$,а значить $\mu_{X(n)}(p) = inf\{\lambda > 0 : \max\limits_{i=1,\dots,m} \frac{x_i}{p_i} = \frac{\lambda}{Nn^2}\}$. Або $\lambda = \max\limits_{i=1,\dots,m} \frac{x_i N n^2}{p_i}$. З властивостей функції $mu_{X(n)}(x)$ випливає, що мінімальний час $T_{min} = \min\limits_{x \in X(n)}$ існує і єдиний. Для врахування пересилок та затримки з'єднання потрібно

зазначити, що алгоритм надсилає $2x_iNn$ елементів (x_i пар матриць розмірів $n * N$) на відповідний вузол i та приймає $x_i * n^2$ елементів (x_i результатів множення матриць $n * N$ та $N * n$).

Отже, сумарний час закінчення з урахуванням пересилок та затримок дорівнює:

$$T_s(x, X(n)) = \max_{i=1,\dots,m} \left\{ \frac{x_i N n^2}{p_i} + \frac{x_i(n^2 + 2Nn)}{q} + x_i l \right\} \tag{2.6}$$

Твердження 2. Існує мінімум часу по $x - \min_{x \in X(n)} T_s(x, X(n))$

2.2 Аналіз штрафів від величини розбиття

У цьому підрозділі ми проведемо аналіз того, як величина розбиття впливає на час виконання задач у потоковій моделі.

Як ми бачимо з 2.6, наявні 2 види штрафів від дрібності розбиття. Перший вид з'являється через надлишковість передачі даних при розбитті матриці на підматриці, а другий напряму від кількості задач, що утворились в результаті розбиття.

Пронормуємо x_i:

$$d_i = \frac{x_i}{\sum\limits_{i=1,\dots,m} x_i} = \frac{x_i}{k} \tag{2.7}$$

Таким чином d_i це доля обчислень вузла i, $d_i \in [0, 1]$, $\sum\limits_{i=1,\dots,m} d_i = 1$. І навпаки, $x_i = d_i * k = d_i * \frac{N^2}{n^2}$

Перепишемо 2.6 за допомоги d_i:

$$T_s(x, X(n)) = \max_{i=1,\dots,m} \left\{ \frac{d_i k N n^2}{p_i} + \frac{d_i k(n^2 + 2Nn)}{q} + d_i k l \right\} \tag{2.8}$$

Проте нас більше всього цікавить рівняння із заміною $k = \frac{N^2}{n^2}$.

$$T_s(x, X(n)) = \max_{i=1,\dots,m} \left\{ \frac{d_i \frac{N^2}{n^2} N n^2}{p_i} + \frac{d_i \frac{N^2}{n^2}(n^2 + 2Nn)}{q} + d_i \frac{N^2}{n^2} l \right\} \quad (2.9)$$

Після скорочення отримаємо:

$$T_s(x, X(n)) = \max_{i=1,\dots,m} \left\{ d_i \frac{N^3}{p_i} + d_i \frac{N^2(1 + 2 * \frac{N}{n})}{q} + d_i \frac{N^2}{n^2} l \right\} \quad (2.10)$$

Позначимо за T_i час роботи вузла i, тоді:

$$\begin{aligned} T_i &= d_i \frac{N^3}{p_i} + d_i \frac{N^2(1 + 2 * \frac{N}{n})}{q} + d_i \frac{N^2}{n^2} l \\ T_s(x, X(n)) &= \max_{i=1,\dots,m} T_i \end{aligned} \quad (2.11)$$

Виділимо окремі складові T_i:

$$\begin{aligned} A_i &= \frac{N^3}{p_i} \\ B_i &= \frac{N^2(1 + 2 * \frac{N}{n})}{q} \\ C_i &= \frac{N^2}{n^2} l \\ T_i &= d_i(A_i + B_i + C_i) \end{aligned} \quad (2.12)$$

Таким чином ми розділили час виконання на 3 окремі складові, кожна з яких має свій сенс. Перша складова A_i відповідає за сумарну складність алгоритма множення двох матриць без розбиття. Друга складова B_i відповідає за надлишковість передачі даних і третя C_i за затримки.

Причому помітимо, що A_i ніяк не залежить від n. Тобто перша складова це завжди повна складність задачі множення цілих матриць і незалежно від розбиття складність блочного множення сумарно залишається незмінною.

Нехай фіксована конфігурація середовища, тобто $p_i, i = 1, \dots, m$ задані та незмінні. У такому випадку при зміні розбиття долі обчислень вважаємо

незмінними, оскільки вони в першу чергу залежать від потужностей обчислювальних вузлів.

І для двох різних розбиттів $n_1, n_2 : n_1 < n_2$ ми маємо складові A_i однаковими, оскільки вони не залежать від розбиття.

$$\begin{aligned} & n_1, n_2 : n_1 < n_2 \\ & A_i^{n_1} = A_i^{n_2} = \frac{N^3}{p_i} \\ & \frac{B_i^{n_1}}{B_i^{n_2}} = \frac{1 + 2 * \frac{N}{n_1}}{1 + 2 * \frac{N}{n_2}} = \frac{n_2(n_1 + 2N)}{n_1(n_2 + 2N)} \\ & \frac{C_i^{n_1}}{C_i^{n_2}} = \left(\frac{n_2}{n_1}\right)^2 \end{aligned} \tag{2.13}$$

Візьмемо за $n_2 = N$, оскільки як можна здогадатися, якщо ми матрицю не розрізаємо, тоді надлишковості немає. Тому будемо порівнювати випадок розрізання і множення цілої матриці на віддаленому вузлі. У такому випадку звісно з'являється проблема неефективного використання обчислювальної мережі, проте на даний момент нас цікавить дослідження надлишковості при множенні матриці блочно.

$$\begin{aligned} & n_1, n_2 : n_1 < n_2, n_2 = N \\ & r_{n_1} = \frac{N}{n_1} \\ & \frac{B_i^{n_1}}{B_i^N} = \frac{N(n_1 + 2N)}{n_1(N + 2N)} = \frac{n_1 + 2N}{3n_1} = \frac{1 + 2r_{n_1}}{3} \\ & \frac{C_i^{n_1}}{C_i^{n_2}} = \left(r_{n_1}\right)^2 \end{aligned} \tag{2.14}$$

Для наглядності побудуємо графік відношення $\frac{B_i^{n_1}}{B_i^N}$ для деяких віксованих N. Будемо брати $n_1 \in [50, N]$ щоб не мати проблем з маштабом оскільки при малих n_1 це відношення дуже велике.

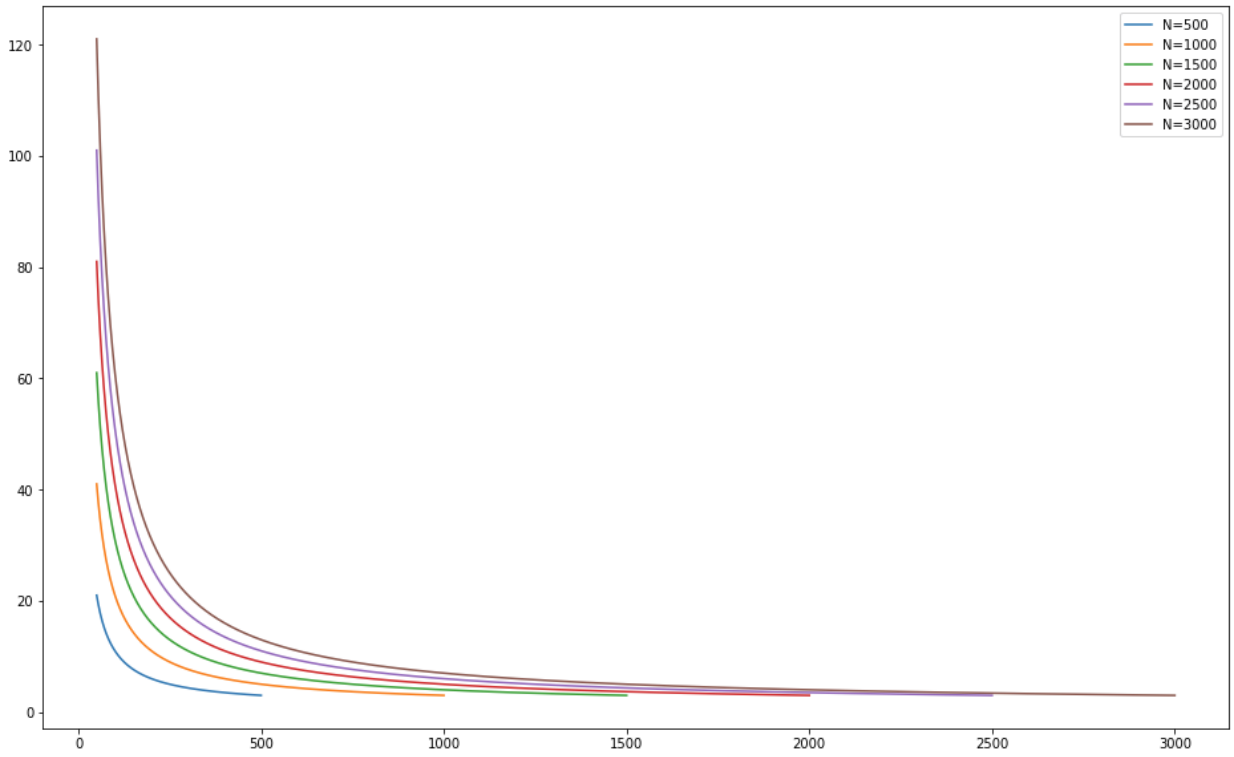


Рисунок 2.1 – Графік залежності відношення $\frac{B_i^{n_1}}{B_i^N}$ від $n_1 \in [50, N]$ при фіксованих N

2.3 Дискретна версія моделі обчислень множення матриць

Нехай розмір блоку n є дільником N, тоді задача множення матриць $N * N$ та $N * N$ поділяється на $k = \frac{N^2}{n^2}$ паралельних задач множення матриць $n * N$ та $N * n$. У порівнянні з потоковою моделю, дискретний варіант має скінченну множину комбінацій розподілу задач планувальником на обчислювальні вузли.

Визначимо множину $Y(n)$ по аналогії з $X(n)$:

$$Y(n) = \{y \in R^m : y_i \in 0, \dots, k, \sum_{j=1}^{m} y_j = k, i = 1, \dots, m\} \tag{2.15}$$

Оскільки $Y(n)$ це дискретний варіант $X(n)$, то очевидне включення $Y(n) \subset X(n)$. Задача планувальника є саме вибір конкретного вектора

$y \in Y(n)$. У роботі досліджуються планувальники типу $extr_1 extr_2$ - min min, min max, max min та max max. Принцип роботи їх дуже простий.

1 Формується черга задач
2 Із черги задач вилучається задача згідно з конфігурацією $extr_1$. min конфігурація вибирає задачу із черги з найменшою складністю, а max з найбільшою
3 Вилучена з черги задача надсилається на вільний процесор згідно з правилом $extr2$. min конфігурація має на меті виконати задачу за найкоротший час і вибирає вільний процесор з найбільшою потужністю, а max навпаки - з найменшою
4 Якщо черга задач не пуста, то повернутися на крок 2

Твердження Для будь-якого n виконуються нерівності:

$$T_d(n) = \min_{y \in Y(n)} T(y, X(n))) \geq \min_{x \in X(n)} T(x, X(n))) \tag{2.16}$$

Також варто пояснити саму суть пошуку найкращого розбиття. Основна проблема у тому, що для великих розбиттів планування може бути недостаньо рівномірно розподелене. Наприклад, розглянемо систему з трьома однаковими обчислювальними вузлами. Розбиття матриць на 2 підматриці буде недостатньо ефективно оскільки будуть сформовані 4 задачі. 3 з 4 задач виконаються повністю паралельно, але четверта буде виконуватись на одному вузлі, в той час як інші 2 будут простоювати. Саме для цього ми і намагаємося розрізати матриці на рівні малі шматки, щоб такої проблеми не виникало. Проте через наявність штрафів за пересилку за затримок дуже малі розбиття також не підходять оскільки штрафи стають занадто великими.

Також з граничних випадків можна розгянути такий, при якому $q = \inf$ та $l = 0$. Таку ситуацію можна зустріти при обчисленні добутку матриць на одному комп'ютері з кількома ядрами. Саме для цього випадку очевидно, що найкращою стратегією буде розрізати на найменші шматки. Таким чином буде мінімізований час простою при виконанні останніх задач з черги.

2.4 Некооперативна ігрова модель планування множення матриць для двох користувачів

Сформулюємо концепції гри між двома гравцями, бажання яких є виконання задачі множення матриць у розподіленому середовищі з паралелізацією методом розбиття матриць на блоки.

Некооперативна гра описує процес прийняття рішення про розбиття двома гравцями в умовах конфлікту інтересів. Некооперативність полягає у тому, що немає зовнішніх причин до їх співпраці, проте в самій грі з певною структурою може виникати співпраця гравців.

Нехай у загальному випадку ця гра проводиться між гравцями $u_i, i = 1, \dots, L$. Усі гравці мають рівний доступ до розподіленого середовища з потужностями обчислювальних вузлів $p_i, i = 1, \dots, m$ через спільний інтерфейс планувальника. Також передача даних для множення підблоків проходить по каналам з пропускними здатностями q та затримкою l. Кожен з гравців може зареєструвати певний набір задач і після цього чекати на результат. Часом для гравця u_i будемо вважати час повернення результата усіх відісланих ним задач, тобто момент, коли він отримає останній блок матриці результата.

Для гравців заданий розмір N і їх задача порахувати у розподіленому середовищі добуток матриць $N * N$ та $N * N$. Стратегіями гравців називаємо $n_i \in [1, N]$ які визначають розмір блоку розбиття.

Кожен гравець хоче виконати паралельне множення матриць якможна швидше. Конфлікт полягає у тому, що зміна стратегії розбиття одним гравцем може покращити його час, проте значно погіршити час другого гравця.

Виділяють гравців раціональних та нераціональних. Дії раціонального гравця спрямовані на максимізацію його виграшу. Нераціональні можуть вносити хаос випадковими ходами. Вважатимемо, що гравці у цій грі раціональні.

Для спрощення будемо вважати, що виконуються такі припущення:

- Страгерії користувачів $n_i, i = 1, \dots, k$ впорядковані за зростанням
- Стратегії користувачів n_i є дільниками N
- У випадку, якщо користувачі вибрали однакове розбиття, то їх час завершення однаковий та дорівнює подвоєному індивідуальному часу

- Існує єдиний мінімум $T_d(n_j), j = 1, \dots, k$. Позначимо індекс, при якому досягається мінімум за j^*

2.5 Імплементація імітаційної моделі

Імітаційна модель дозволяє симулювати процес множення матриць у розподіленому середовищі враховуючи як час на власне обчислення так і передачу даних.

Розглянемо задачу множення двох $N*N$ матриць M1 та M2. Позначимо за n кількість рядків матриці M1 та відповідно стовбців матриці M2. Таким чином отримуємо 2 підматриці $n * N$ та $N * n$ які і формують одну задачу, що буде відіслана планувальнику. Таких задач буде $\lfloor \frac{N}{n} \rfloor \times \lfloor \frac{N}{n} \rfloor$. Проте в обох матрицях M1 та M2 у випадку якщо n не дільник N буде залишок рядків M1 та стовбців M2 відповідно. Позначимо розмір залишка за . Тому до основних задач ще потрібно додати задачі множення матриць $m * N$ та $N * n$, матриць $n * N$ та $N * m$ і одну задачу множення $m * N$ та $N * m$.

Тобто задача множення матриць $N * N$ та $N * N$ розбивається на такі підзадачі:

- $\lfloor \frac{N}{n} \rfloor \times \lfloor \frac{N}{n} \rfloor$ множень матриць $n * N$ та $N * n$
- $\lfloor \frac{N}{n} \rfloor$ множень матриць $m * N$ та $N * n$
- $\lfloor \frac{N}{n} \rfloor$ множень матриць $n * N$ та $N * m$
- 1 множення матриць $m * N$ та $N * m$

Множення матриць $N1 * N2$ та $N2 * N3$ потребує $N1 * N3 * N2$ операцій множення та $N1 * N3 * (N2 - 1)$ операцій додавання. Позначимо за AM коефіцієнт складності операції множення по відношенню до операції додавання. Тоді складність множення матриць $N1 * N2$ та $N2 * N3$ можна виразати у одиницях операцій додавання як:

$$Complexity(N1, N2, N3) = N1 * N3 * (N2 * AM + N2 - 1) \qquad (2.17)$$

Позначивши потужність ОВ за P (кількість операцій додавання / секунду) та складність задачі C (кількість операцій додавання) отримаємо час, який ОВ витратить на обчислення задачі C:

$$T_{processing} = \frac{C}{P} \tag{2.18}$$

Також для множення матриць $N1 * N2$ та $N2 * N3$ потрібно переслати $N1 * N2 + N2 * N3$ елементів до обчислювалюного вузла, та отримати результат у розмірі $N1 * N3$ елементів. Час на передачу даних обчислюється таким чином:

$$T_{transfer} = latency + \frac{N1 * N2 + N2 * N3 + N1 * N3}{bandwidth} \tag{2.19}$$

де $latency$ визачає затримку між відправленням пакета від користувача до планувальника та $bandwidth$ пропускну здатність між користувачем та планувальником.

Загальний час на обробку задачі отримується з урахуванням 2.18 та 2.19:

$$T = T_{processing} + T_{transfer} \tag{2.20}$$

Для двох гравців вибираються n_1 та n_2, формуються задачі, додаються в загальний список та випадково перемішуються і подаються на планувальник. Час для кожного з гравців визначається як час повернення від планувальника до гравця останьої його задачі.

РОЗДІЛ 3 АНАЛІЗ РЕЗУЛЬТАТІВ СИМУЛЯЦІЙ

3.1 Структура симуляційного програмного забезпечення

Програма написана на мові C++ та умовно поділяється на дві логічні частини: модуль обробки параметрів та модуль симуляції.

Модуль обробки параметрів дозволяє задавати розмір матриць, режим одноно гравця чи двох, межі перебору стратегій розрізання, параметри планувальника, характеристики обчислювальних модулів, параметрів мережі – bandwidth та latency без перековпіляції програми через параметри командного рядка.

Модуль симуляції генерує задачі для кожного з користувачів, зливає їх в один список та власне подає їх на симулятор, який повертає оброблені задачі з проставленими змінними часу початку та завершення роботи над задачею і номером обчислювального вузла, який обробляю цю задачу.

Етапи роботи симулятора:

- Перемішування списка очікуючих задач з метою емуляції отримання задач у випадковому порядку.
- Сортування списка очікуючих задач по складності у відповідності до пріорітетності задач та списка обчислювальних вузлів по потужності у відповідності до пріорітетності обчилювальних вузлів. Наприклад для minmax планувальника список очікуючих задач буде відсортованим від простої до складної, а список обчислювальних вузлів від потужного до повільного.
- Ініціалізація початкових задач для обчислювальних вузлів вилучаючи перші елементи з відсортованих списків очікуючих задач та вузлів, проставлення початкового часу, який на даний момент рівний 0, та обчислення часу очікуваного завершення виконання задачі. Структури з посиланням на задачу, обчислювальний вузол та даними про початок та завершення виконання задачі поміщаються у чергу з пріорітетом по найменшому часу завершення.
- Взяти з пріорітетної черги задачу, яка буде найближчою по часу наступною виконаною задачею. Приняти час симуляції за час завершення

взятої з черги задачі, додати задачу до списку виконаних задач разом із даними про час її завершення.

- Якщо список очікуючих задач не пустий, то вилучити перший елемент, поставити час початку як час симуляції, обчислити час завершення та додати у чергу з прірітетом, поставивши обчислювальний вузол як перший вільний із відсортованого списка вузлів.
- Якщо пріорітетна черга не пуста, то повернутися на крок 4.
- Знайти у списку виконаних задач найпізніші повернені задачі для кожного з користувачів та повернути їх час завершення.

3.2 Ілюстрація результатів симуляції

3.2.1 Симуляція для одного користувача

Симуляція для одного користувача в загальному випадку навіть не вважається грою, а більш схоже на звичайну оптимізаційну проблему. Проте графіки симуляцій для одного користувача можуть показати характер обробки задач при блочному розрізанню матриць.

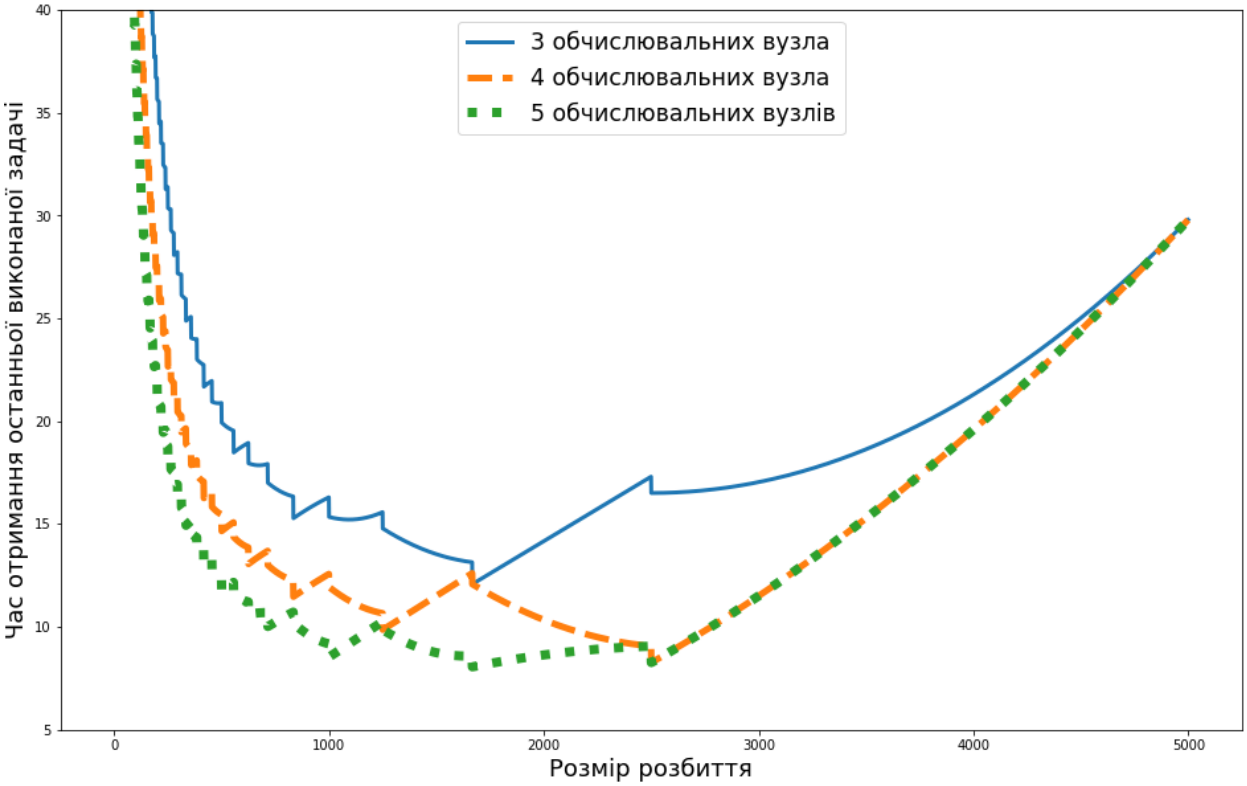


Рисунок 3.1 – Графік залежності часу виконання всіх задач користувача від розміру розрізання для різних кількостей обчислювальних вузлів

На Рис. 3.1 зображено залежність часу симуляції від розбиття при фіксованих N, latency, bandwidth для 3, 4 та 5 обчислювальних вузлів. Чим більше ОВ, тим швидше множення матриць, проте для деяких розрізань можна побачити майже однаковий час при різній кількості обчислювальних вузлів. Особливо це помітно для 4 та 5, починаючи з розміру розрізання 2500 час для них однаковий хоч для обчислень і задіяно більше ОВ.

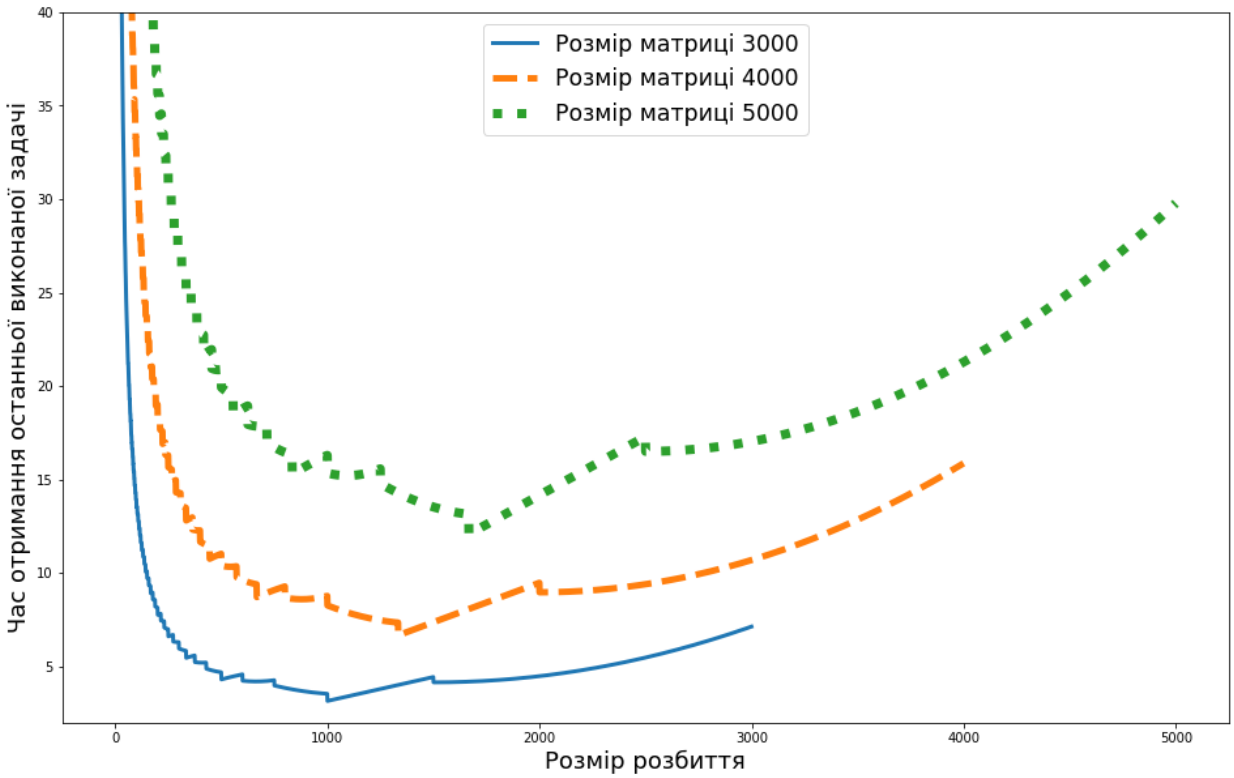


Рисунок 3.2 – Графік залежності часу виконання всіх задач користувача від розміру розрізання для різних розмірів матриць

На Рис. 3.2 показано для розмірів матриць 3000, 4000 та 5000 графіки залежності часу виконання усіх задач користувача від розбиття для 5 ОВ. З нього відносно можна помітити, що графіки мають приблизно однакову форму і можливо між ними має місце звичайна пропрорційну залежність від розміру матриці N.

3.2.2 Аналіз штрафів за розбиття

Також варто перевірити аналіз штрафів з теоретичного розділу за допомогою симуляційної системи. Це дуже просто зробити, оскільки програма дозволяє задавати деякі з параметрів як inf. З досліджувані параметри це: ping, bandwidth та mips. Якщо поставити $bandwidth = \text{inf}$ та $mips = \text{inf}$, тоді складові часу, які відповідають за передачу даних та саме общичслення, обнуляться і ми будемо мати чисто лише час, який був спричинений затримками пакетів.

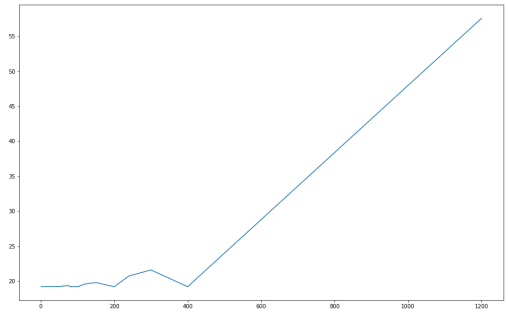
Проводити експерименти будемо для конфігурацій із заниженими потужностями обчислювальних 3х вузлів $6e7$ операцій/секунду, малим значенням пропускної здатності $8e7$ біт/секунду та затримкою пакетів в 1 мілісекунду.

Таблиця 3.1 наглядно показує для випадку множення матриць розмірів 1200 штрафи. Також можемо побачити велику різницю мів розрізаннями 1200 , 600 та 400. 1200 значить не розрізати матрицю та фактично виконати множення на одному ядрі, також час із першої колонки відповідає за чистий час обчислень без додавання часу пересилки та затримок. Час для розрізання 600 також ще поганий, оскільки було сформовано 4 задачі у той час як доступні всього 3 обчислювальних вузла. Тобто остання задача виконувалась лише на одному вузлі поки інші простоювали. Далі починаючи з розрізання 400 час відносно стабілізується. Це можна пояснити тим, що для наступних розрізань хоч і результуюча кількість задач може не бути дільником кількості обчислювальних вузлів, проте задачі вже настільки малі, що час простою в кінці значно менший ніж для випадків 1200 та 600. Також ці результати підтверджують теорію про те, що складність множення матриць ніяк не змінюється від щільності розбиття.

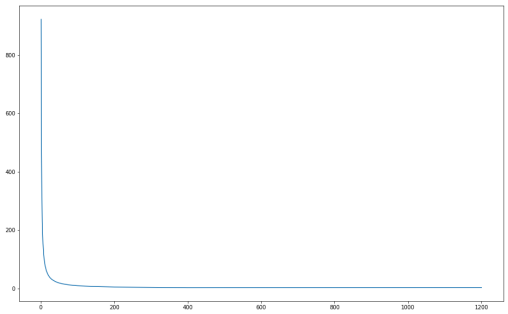
	processing time	transfer time	latency time	total
1	19.19200	921.98400	480.00000	1421.17600
2	19.19200	461.18400	120.00000	600.37600
3	19.19224	307.58784	53.33400	380.11408
4	19.19200	230.78400	30.00000	279.97600
5	19.19200	184.70400	19.20000	223.09600
6	19.19296	153.99170	13.33400	186.51866
8	19.19200	115.58400	7.50000	142.27600
10	19.19200	92.54400	4.80000	116.53600
12	19.19584	77.19944	3.33400	99.72928
15	19.19800	61.84332	2.13400	83.17532
16	19.19200	57.98400	1.87500	79.05100
20	19.19200	46.46400	1.20000	66.85600
24	19.20735	38.81503	0.83400	58.85638
25	19.19200	37.24800	0.76800	57.20800
30	19.21599	31.14288	0.53400	50.89287
40	19.19200	23.42400	0.30000	42.91600
48	19.25341	19.64667	0.20900	39.10908
50	19.19200	18.81600	0.19200	38.20000
60	19.28796	15.82272	0.13400	35.24468
75	19.34194	12.77100	0.08600	32.19894
80	19.19200	11.90400	0.07500	31.17100
100	19.19200	9.60000	0.04800	28.84000
120	19.57584	8.22528	0.03400	27.83512
150	19.79175	6.73200	0.02200	26.54575
200	19.19200	4.99200	0.01200	24.19600
240	20.72736	4.56192	0.00900	25.29828
300	21.59100	3.88800	0.00600	25.48500
400	19.19200	2.68800	0.00300	21.88300
600	28.78800	2.88000	0.00200	31.67000
1200	57.57600	3.45600	0.00100	61.03300

Таблиця 3.1 – Таблиця штрафів від величини розрізання

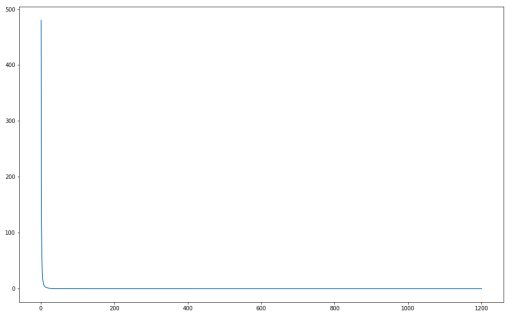
Розглянемо окремо графіки трьох складових загального часу: виконання задач, передачі даних та затримок.



(а) Графік часу обчислень від розміру розрізання



(б) Графік часу передачі даних від розміру розрізання



(в) Графік часу затримок від розміру розрізання

Рисунок 3.3 – Three subfigures

Також для кращого розуміння випадків розбиттів 1200 , 600 та 400 варто проілюструвати процес виконання задач на блоках.

3.2.3 Симуляція для двої користувачів

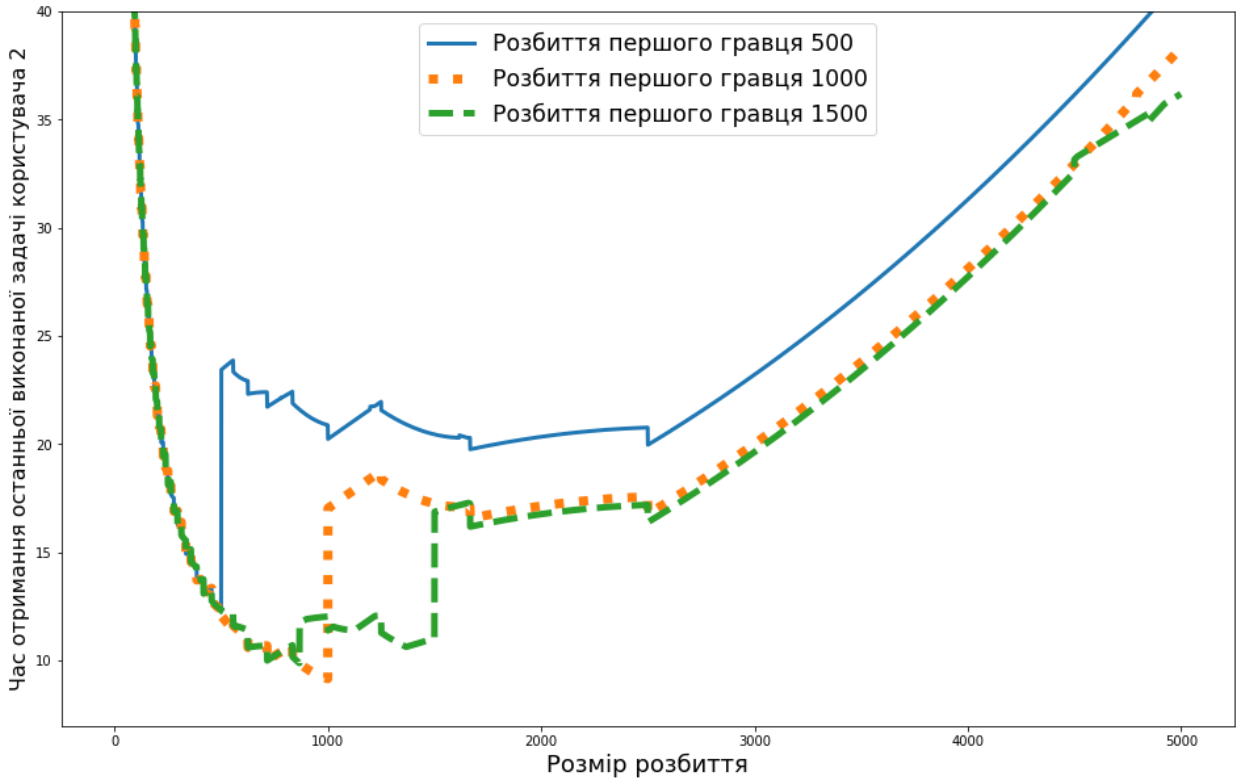


Рисунок 3.4 – Графік залежності часу виконання всіх задач другого користувача від розміру його стратегії розрізання при фіксованих стратегіях першого користувача

На Рис. 3.4 зображено залежність часу виконання усіх задач другого користувача від розміру розбиття при фіксованому розбитті користувача 1 для 5 ОВ. На графіку чітко спостерігається стрибки при переході розбиття користувача 2 за фіксоване значення розбиття користувача 1. Це особливість minmin та minmax оскільки вони в першу чергу виконують найлегші задачі, тому користувач, що вибрав менше розбиття, має менший час виконання усіх його задач.

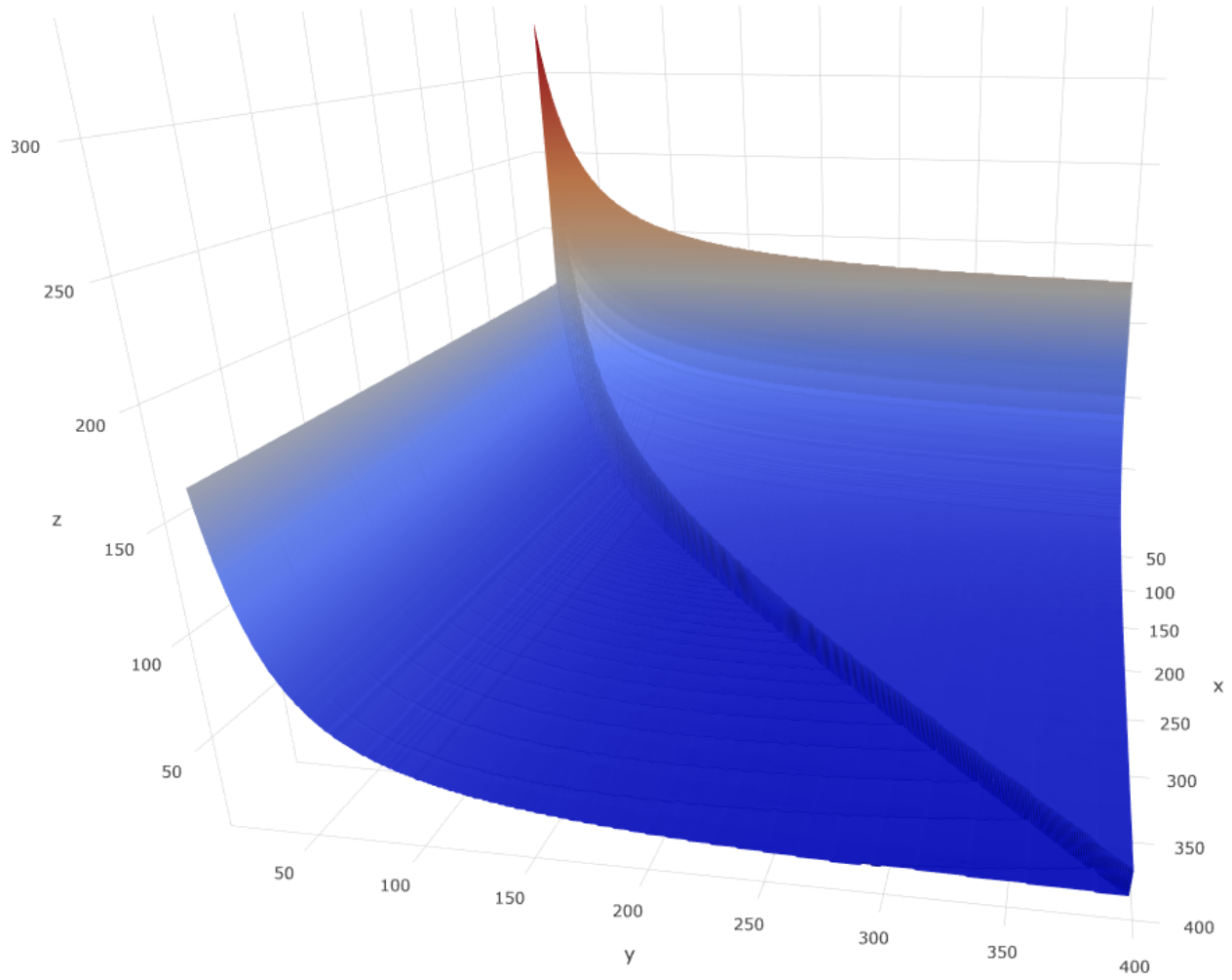
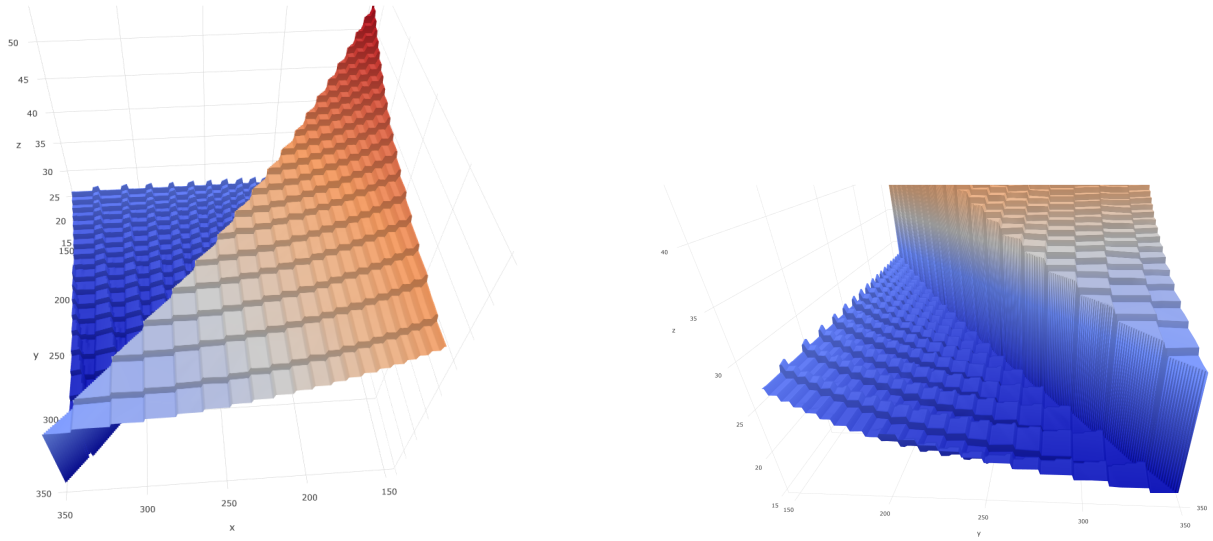


Рисунок 3.5 – Графік залежності часу виконання всіх задач другого користувача для всіх комбінацій стратегій обох користувачів з відрізка $[20, 400]$

На перший погляд поверхня, яка отримана шляхом симуляції усіх можливих пар стратегій обох користувачів з відріка $[20, 400]$, може здаватися гладкою та випуклою Рис. 3.5. Проте, пам'ятаючи природу графіків при фіксованій стратегії першого користувача на Рис. 3.4, слід подивитися на Рис. 3.5 більш ретельно, наприклад побудувати поверхню усіх комбінацій стратегій з відрізка $[150, 350]$.



(а) Верхня частина графіка (б) Нижня частина графіка

Рисунок 3.6 – Графік залежності часу виконання всіх задач другого користувача для всіх комбінацій стратегій обох користувачів з відрізка $[150, 350]$, 3.6а - фокус на верхню частину поверхні, 3.6б - фокус на нижню частину поверхні

При кращій деталізації можна чітко побачити, що на Рис. 3.6а спостерігається форма сходинок по всій верхній частині графіка і вона не така проблемна, як нижня частина, показана на Рис. 3.6б. Оскільки нижня частина більш цікава через те, що час завершення усіх задач користувача там менший, то і глобальний мінімум варто шукати саме на нижній частині. Нижня частина має особливої форми канави і саме вони є основною проблемою.

Slice First	Slice Second	Time First	Time Second
293	293	56.196913114	56.133074705
293	294	29.273646274	56.249161287
293	295	31.193692060	55.135836942
293	296	31.078784491	55.140002601
294	293	56.249161287	29.273646274
294	294	56.134305135	56.143839692
294	295	31.201760723	55.131381284
294	296	31.086853154	55.135546943
295	293	55.135836942	31.193692060
295	294	55.131381284	31.201760723
295	295	54.006477079	53.951126737
295	296	30.047061935	54.102473211
296	293	55.140002601	31.078784491
296	294	55.135546943	31.086853154
296	295	54.102473211	30.047061935
296	296	54.057876812	54.001345879

Таблиця 3.2 – Таблиця значень часів повернення усіх задач користувачів для різних стратегій розрізань

З Таблиці 3.2 можна побачити як в між деякими сусідніми значеннями спочатку час трохи збільшується, а потім різко зменшується. Таким чином структура функції і проблеми її оптимізації очевидні.

РОЗДІЛ 4 Керування стартапом проекту

4.1 Опис ідеї проекту

Назва проекту - “Efficient task distribution for cloud computing”. Проект являє собою систему, яка дозволяє правильно розбивати математичны задачі на підзадачі з метою мінімізації часу їх виконання у розподіленому середовищі.

Проект вирішує проблему виконання складних задач, які можна розділити на велику кількість простіших задач, у розподіленому середовищі з мінімізацією часу виконання задач для багатьох користувачів. В першу чергу така проблема може виникнути у дослідницьких центрах, де часто виконуються прості операції проте з неймовірно виликими обсягами даних. Часто такі задачі дуже легко розбивати на більш прості підзадачі, проте не завжди просто оцінити найефективнішу стратегію розбиття. На даний момент такі задачі вирішуються звичайним паралелізмом і такий метод відносно задовольняє потреби, проте основною метою проекта є пошук найефективніших шляхів паралельного виконання дрібних задач та використання симуляційних систем для аналізу ефективності розбиття задачі на підзадачі.

Зміст ідеї	**Напрямки застосування**	**Вигоди для користувача**
Розробка симуляційної системи, яка допоможе симюлювати обчислення у Cloud системі за значно коротший час без застостування додаткових обчюслювальних машин.	1. У сфері досліджень планувальників, перевірка теорій, аналіз особливостей планувальників	Можливість провести симуляцію роботи cloud системи із застосуванням вибраного планувалника без значних затрат часу чи грошей
	2. Проведення досліждень задач та знаходження найефективніших розбиттів.	Симуляція роботи cloud системи при великих обсягах задач за значно коротший час

Таблиця 4.1 – Опис ідеї стартап-проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Cloud Sim	Grid Sim	Amazon AWS			
1	Швидка симуляційна система	продукт на мові C++	бібліотека на Java	бібліотека на Java	повномасштабна система обчислень	потребує ретельного аналізу cloud систем	доступна платформа для наукових досліджень планувальників	дозволяє базово оцінити ефективність cloud системи
2	Ефективна аналітика паралельних алгоритмів	додаткові функції для пошуку оптимальних конфігурацій алгоритма	-	-	розробка власного пакету аналітики	потребує значних витрат у дослідження	-	дозволяє значно прискорити обчислення, що у майбутньому дозволить економити на обчисленнях у cloud системах

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Побудова симуляційної системи для тестування cloud систем	На мові C++ реалізація системи, що аналогів по швидкодії для якої немає	Технологію потрібно доробити, сама ідея протестована і показує непогані результати по швидкодії, потрібно лише розширити її можливості	Технології доступні усім користувачам
2	Аналіз швидкодії паралельних алгоритмів у cloud системах	Залучення команди математиків до аналізу найпопулярніших планувальників та побудови математичних моделей основних задач з лінійної алгебри	Не наявні, потрібно запускати процес з нуля	Доступні

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

4.2 Аналіз ринкових можливостей запуску стартап-проекту

№	Показники стану ринку(найменування)	Характеристика
1	Кількість головних гравців, од.	3
2	Загальний обсяг гравців, $	4.5 млрд.
3	Динаміка ринку	Попит зростає, пропозиція майже не збільшується
4	Обмеження для входу на ринок	Наявність якісного програмного продукту
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі	100%

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія(цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп	Вимоги користувачів до товару
1	Необхідність аналізу різних планувальників з метою побудови та перевірки математичних моделей	Дослідницькі центри та університети	У різних центрах чи університетах проводяться різні дослідження і	Простота у використанні та достатня швидкодія для перевірки гіпотез
2	Симуляція ресурсоємних задач з розбиттям їх на підзадачі	Університети, дослідницькі центри та компанії, які хочуть збільшити ефективність обчислень	Кожна окрема задача потребує особливий аналіз та алгоритм розбиття	Симуляційна система повинна бути достатньо універсальною оскільки задачі можуть бути не лише чисто математичні

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Платформи для хмарних обчислень розробляють функцію симуляції обчислень за значно меншою ціною	Поява функції симуляції на платформах хмарних обчислень за малою ціною може перетягнути значну частину клієнтів на сторону платформ	Перегляд цін на підписки, покращення співпраці з обчислювальними центрами та надання додаткових послуг по аналізу моделей задач
2	Поява продукту з аналогічною швидкодією та відкритим кодом	Приводить до повного знецінення товару оскільки відкритий код означає, що продукт доступний усім безкоштовно	Розробка додаткових функцій та графічного інтерфейсу для полегшення процесу роботи з програмним продуктом

Таблиця 4.6 – Фактори загроз

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Співробітництво з платформами хмарних обчислень	Можливість просування продукта напряму на платформах завдяки угоді з компаніями провайдерами хмарних обчислень	Значні збільшення продажів підписок та простіша реклама
2	Збільшення попиту на складні обчислення	Можливість швидкого росту завдяки аналізу популярних високонавантажених алгоритмів	Збільшення продажів

Таблиця 4.7 – Фактори можливостей

№ п/п	Особливості конкуретного середовища	В чоему проявляється дана характеристика	Вплив на діяльність підприємства
1	Олігополія	Існують відриті системи - CloudSim та GridSim	Задають стантарт програмного забезпечення для симуляцій хмарних обчислень
2	Міжнародний рівень конкуретної боротьби	Цифрові продукту інформаційного пошуку не мають кородонів	Універсальність продукту та швидкодія
3	Галузева конкуренція	Конкуренція проходить у галузі аналізу паралельних алгоритмів	Надавати високоефективний продукт для аналізу складних задач
4	Товарно-видова конкуренція за видами товарів	Наявність функцій для повноцінної симуляцій Cloud системи	Спостереження за Cloud системами да додавання нових функцій
5	За характером конкурентних перваг: неціновa	В першу чергу важливі швидкодія та універсальність	Розширення функціоналу, додавання нових моделей у продукт
6	За інтенсивністю - марочна конкуренція	На ринку я аналогічні продукти зі схожим функціоналом, проте їх швидкодія недостатня для серйозних досліджень	Додавання нового функціоналу у продукт

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Назва характеристики	**Характеристика**
Прямі конкуренти	Cloudsim Plus
Потенційні конкуренти	Goolgle, Microsoft, Amazon AWS, Digital Ocean
Постачальники	Amazon, Digital Ocean, Google
Клієнти	KPI, Amazon, інші університети
Товари-замінники	CloudSim, GridSim, CloudSim Plus

Таблиця 4.9 – Аналіз конкуренції за М. Портером (Складові аналізу)

Назва характеристики	**Характеристика**
Прямі конкуренти	Прямі конкуренти - дрібні компанії
Потенційні конкуренти	Потенційні конкуренти - компанії-гіганти
Постачальники	Постачальники хмарних обчислень - потенційні конкуренти
Клієнти	Для клієнтів продуктивність паралельного алгоритма - основна проблема
Товари-замінники	Товари-замінники працюють дуже повільно та непридатні для складних симуляцій

Таблиця 4.10 – Аналіз конкуренції за М. Портером (Висновки)

№ п/п	Фактор конкурентноспромо-жності	Обґрунтування (чинники, що роблять фактор для порівння конкурентних проектів значущим)
1	Швидкодія	Швидкодія значно більша ніж у аналогів написаних на мові Java
2	Готова база простих планувальників	Дає можливість використовувати та досліджувати основні планувальники без велеких зусиль
3	Простота використання	У конкурентів програмні продукти часто дуже складні та потребують багато часу на аналіз різних прикладів перед самою імплементацією симуляції. Також через складну структуру продукту іноді в коді з’являються помилки, які складно виявити на перший погляд.
4	Універсальність	Дозволяє моделювати структури Cloud середовищ будь-якої складності

Таблиця 4.11 – Обґрунтування факторів конкурентноспроможності

#	Фактор конкурентноспроможності	Бали 1-20	Рейтинг відносно Cloudsim Plus						
			-3	-2	-1	0	1	2	3
1	Швидкодія	20							+
2	Готова база простих планувальників	20							+
3	Простота використання	15						+	
3	Універсальність	15		+					

Таблиця 4.12 – Порівняльний аналіз сильних та слабких сторін

Сильні сторони: • Висока швидкодія у порівнянні з аналогічними продуктами • Вбудовані прості планувальники, які легко використовувати • Простота бібліотеки	Слабкі сторони: • Недостатня універсальність оскільки продукт лише у початковому виді • Продукт постачається мовою C++, яка вважається складнішою за Java для людей, які більше математики ніж програмісти
Можливості: • Інтеграція у реальні Cloud системи з метою попереднього аналізу задачі перед саме замовленням хмари • Проводити дослідження планувальників • Просте тестування математичних моделей оскільки симуляції достатньо швидкі	Загрози: • Компанії гіганти можуть випустити власне програмне забезпечення для попереднії симуляцій • Існуючі продукти у розробці також можуть активізуватися та спробувати конкурувати на ринці • Поява нових конкурентів з поріняно схожою швидкодією програмного продукту

Таблиця 4.13 – SWOT- аналіз стартап-проекту

№ п/п	Альтернатива (орієнтований комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Розробка API для інших мов програмування з метою полегшення процесу використання програмного продукту	0.9	0.5 року
2	Додавання специфічних планувальників у симуляційну систему	0.4	0.5 року
3	Побудова математичних моделей популярних задач лінійної алгебри	0.4	2 роки

Таблиця 4.14 – Альтернативи ринкового впровадження стартап-проекту

4.3 Розробка ринкової стратегії проекту

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйти продукт	Орієнтовний помит цільової групи (сегменти)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Університети та дослідницькі центри	Потребують простого інтерфейсу та високої швидкодії. Оскільки ці складові наявні, то споживачі будут задоволені	Університети потребують продукт для швидкої перевірки теорій. Дослідницькі центри - прискорити власні обчислення	Конкуренція із відкритими безкоштовними рішеннями: CloudSim, CloudSim Plus, GridSim	Оскільки існуючі рішення дуже повільні для випадків симуляції складних паралельних задач, то вхід у сегмент дуже легкий
2	Індивідуальні користувачі	Потребують простої бібліотеки з інтуітивною структурою та універсальним дизайном	Попит серед користувачів Amazon AWS, Microsoft Azure, Digital Ocean	Конкуренція із відкритими безкоштовними рішеннями: CloudSim, CloudSim Plus, GridSim	Вихід у цей сегмент буде важчим оскільки існуючі рішення більш універсальні та покривають більшу множину задач

Таблиця 4.15 – Вибір цільових груп потенційних споживачів

№ п/п	Обрана альтернатива розвитку проекту	Стратегія розвитку ринку	Ключові конкуретноспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Набір основної маси користувачів	На початкових етапах давати університетам можливість користуватися безкоштовно продуктом	Наявність простого інтерфейсу та швидкої симуляційної системи	Просування продукту завдяки науковим публікаціям, які використовують даний програмний продукт
2	Розвиток симуляційної системи	Додавання популярних алгоритмів планування, прикладів аналізу моделей простих математичних задач	Розповсюдженість та проста інтеграція для будь-якої популярної платформи	Популяризація програмного продукту та введення підписок

Таблиця 4.16 – Визначення базової стратегії розвитку

№ п/п	Чи є проект першопро-хідцем на ринку	Чи буде компанія шукати нових споживачів чи забирати існуючих у конкурентів	Чи буде компанія копіювати основні харак-теристики товару кон-курента і які?	Стратегія конкурентної поведінки
1	Проект не є першопро-ходцем	Компанія в першу чергу буде забирати споживачів існуючих продуктів	Компанія має на меті в спочатку реалізувати аналогічний функціонал як в існуючих продуктах	Компанія надає схожий продукт, проте у більш зру-чній формі та із зна-чно кращою швидкодією

Таблиця 4.17 – Визначення базової стратегії конкурентної поведінки

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Висока швидкодія, яка дозволить проводити експерименти у системі значно швидше ніж у реальних cloud системах	Розробка системи на мові C++ яка дозволить проводити швидкі та точні симуляції	Значно вища швидкодія порівняно з аналогами та більш зручний процес аналізу алгоритмів у cloud системах	Швидкодія, простота використання, оптимізація паралельних алгоритмів

Таблиця 4.18 – Визначення стратегії позиціонування

4.4 Розроблення маркетингової програми стартап-проекту

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1	Можливість провести експерименти з різними структурами cloud систем перед власне замовленям системи	Товар надає можливість проводити серії симуляцій обчислень, оцінити ефективність структури cloud системи	Конкуренти хоч і надають товари з більшим функціоналом, проте симуляція складних задач проходить дуже довго
2	Збільшення ефективності паралельних алгоритмів	Продукт дозволяє симулювати процес обчислень та знаходити оптимальні параметри алгоритма для певної структури cloud системи	Конкуренти не надають таких послуг

Таблиця 4.19 – Визначення ключових переваг концепції потенційного товару

4.5 Висновки

Дана магістерська дисертація має перспективи зростання до популярного продукту у сфері хмарних обчислень оскільки наявні аналоги написані на мові Java та мають дуже низьку швидкодію, яка взагалі не дозволяє проводити повномаштабні експерименти з метою оцінки її майбутньої продуктивності. Складнощі в першу чергу можуть бути через недостатню універсальність на початкових етапах релізу продукту та можливу пожвавлену

конкуренцію на етапі виходу на ринок. Також конкуренти з великою кількістю ресурсів можуть розпочати розробку власного продукту, що може сильно вплинути на подальший розвиток стартапа. Такі випадки часто трапляються в IT сфері.

ВИСНОВКИ ПО РОБОТІ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У роботі представлена СПЗ, яке дозволяє дуже швидко емулювати процес множення матриць з використанням одного із чотирьх статичних планувальників: minmin, minmax, maxmin та maxmax. Представлена математична модель, яка дозволяє оцінити глобальний мінімум функції часу. Також продемонстровано застосування інших методів з метою отримання такого ж результату.

СПИСОК ЛІТЕРАТУРИ

1. CloudSim: A Framework For Modeling And Simulation Of Cloud Computing Infrastructures And Services. [Електронний ресурс]. — Режим доступу: http://www.cloudbus.org/cloudsim/.
2. CloudSim Plus: A modern, full-featured, highly extensible and easier-to-use Java 8 Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. [Електронний ресурс]. — Режим доступу: http://cloudsimplus.org/.
3. Mayanka, Katyal. Application of Selective Algorithm for Effective Resource Provisioning In Cloud Computing Environment / Katyal Mayanka, Atul Mishra // International Journal on Cloud Computing: Services and Architecture (IJCCSA). — 2014. — 2. — Vol. 4, no. 1. — Pp. 1–10.
4. Srinivasa Prasanna, G. N. Generalised Multiprocessor Scheduling Using Optimal Control / G. N. Srinivasa Prasanna, Bruce R. Musicus // SPAA. — 1991.
5. Smith, Tyler Michael. Pushing the Bounds for Matrix-Matrix Multiplication / Tyler Michael Smith, Robert A. van de Geijn // FLAME Working Not. — 2017. — 2. — no. 83. — Pp. 1–11.
6. Hong, Jia-Wei. I/O complexity: The red-blue pebble game / Jia-Wei Hong, Hsiang-Tsung Kung // Proceedings of the thirteenth annual ACM symposium on Theory of computing. — 1981. — Pp. 326–333.
7. Irony, Dror. Communication Lower Bounds for Distributed-Memory Matrix Multiplication / Dror Irony, Sivan Toledo, Alexander Tiskin // Journal of Parallel and Distributed Computing. — 2004. — 09. — Vol. 64. — Pp. 1017–1026.
8. Coded Computation over Heterogeneous Clusters / Amirhossein Reisizadehmobarakeh, Saurav Prakash, Ramtin Pedarsani, Salman Avestimehr // arXiv. — 2017. — 01.
9. Coded computation for multicore setups / Kang Wook Lee, Ramtin Pedarsani, Dimitris Papailiopoulos, Kannan Ramchandran / IEEE International Symposium on Information Theory (ISIT). — 2017. — 06. — Pp. 2413–2417.
10. Дорошенко, А.Ю. ПРО ОДНУ МОДЕЛЬ ОПТИМАЛЬНОГО РОЗПОДІЛУ РЕСУРСІВ У БАГАТОПРОЦЕСОРНИХ

СЕРЕДОВИЩАХ / А.Ю. Дорошенко, О.П. Ігнатенко, П.А. Іваненко // Проблеми програмування. — 2011. — no. 1. — Pp. 21,28.

11. Nazarathy, Y. A Fluid Approach to Large Volume Job Shop Scheduling / Y. Nazarathy, G. Weiss // Journal of Scheduling. — 2010. — 5. — no. 13. — Pp. 509–529.
12. Бланк, С. Стартап. Настольная книга основателя / С. Бланк, Б. Дорф. — 2 edition. — Москва : Альпина Паблишер, 2014. — P. 614.
13. Дрейпер, У. Стартапы : профессиональные игры Кремниевой долины / У. Дрейпер. — 2 edition. — Москва : Эксмо, 2012. — P. 378.
14. Тиль, П. От нуля к единице : как создать стартап, который изменит будущее / П. Тиль, Б. Мастерс. — Москва : Альпина паблишер, 2015. — P. 188.
15. Харниш, В. Правила прибыльных стартапов : как расти и зарабатывать деньги / В. Харниш. — Москва : Манн, Иванов и Фербер, 2012. — P. 279.
16. Квашнин, А. Как продвигать проекты коммерциализации технологий : серия методических материалов «Практические руководства для центров коммерциализации технологий. — проект EuropeAid «Наука и коммерциализация технологий. — 2006.