

РЕФЕРАТ

Магістерська дисертація: 76 ст. , 10 рис., 28 табл., 21 джерел та 2 додатки.

Темою роботи є "Теоретико-ігровий аналіз планувальників у гетерогенному багатопроцесорному середовищі".

Робота актуальна оскільки проблема ефективних обчислень існує та чимало досліджень проводиться у цьому напрямку із застосуванням різних підходів, у тому числі і теорії ігор.

Метою дослідження є пошук рівноваг та рішень гри множення матриць у розподіленому середовищі з двома користувачами.

Об'єктом дослідження є планувальники типу extr-extr у розподіленому середовищі. Предмет дослідження - пошук рівноваг та інших оптимальних точок у грі одного та двох гравців.

Дослідження проводиться методом наукового моделювання процесу блочного множення матриці. За основу взята потокова модель і у подальшому звужена до дискретної моделі. На другому етапі дослідження проводяться експерименти за допомоги розробленої симуляційної системи, які дозволяють оцінити точність побудованої математичної моделі та дослідити гру на наявність рівноваг.

Новизна результатів полягає у тому, що розроблена симуляційна система, яка дозволяє проводити експерименти та досліджувати хмарне середовище за значно коротший час на простому комп'ютері.

У роботі проведено аналіз планувальників типу extr-extr , розроблена симуляційна система для проведення експериментів та розглянуті альтернативні підходи до пошуку оптимальних стратегій.

Результати данної роботи можна використати при розробці системи розподілених обчислень з метою мінімізації часу виконання паралельних задач в умовах можливості регулювання складностей задач при розбитті їх на підзадачі. Подальші дослідження можуть бути проведені у напрямі аналізу стандартних операцій лінійної алгебри стандарту BLAS.

ПЛАНУВАЛЬНИКИ, МНОЖЕННЯ МАТРИЦЬ, ТЕОРІЯ ІГОР, ХМАРНІ ОБЧИСЛЕННЯ, РІВНОВАГА.

ABSTRACT

The thesis: 76 p. , 10 fig., 28 tabl., 21 sources and 2 appendices.

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

The work is relevant because the problem of efficient computing exists and many studies are conducted in this direction with the application of different approaches, including the theory of games.

The purpose of this study is to search for the equilibriums and solutions of the matrix multiplication game in a distributed environment with two users.

The research is carried out by the method of scientific modeling of the process of block multiplication of the matrix. This work is based on a fluid model, which is considered for a continuous case, is further reduced to a discrete model. At the second stage of the research, experiments are carried out with the help of the developed simulation system, which allow us to estimate the accuracy of the constructed mathematical model and to investigate the game for the presence of equilibrium.

The novelty of the results is that a simulation system is developed that allows you to experiment and explore the cloud environment in a much shorter time on a simple computer. Also, this system allows you to separately examine each component of the function of total time.

In this paper schedulers of type extr-extr were analyzed, developed a simulation system for conducting experiments, and examined alternative approaches to finding optimal strategies.

The results of this work can be used in the development of a distributed computing system in order to minimize the time of parallel tasks execution in conditions of the ability to control the complexity of tasks when partitioning them into subtasks. Further research can be conducted in the direction of the analysis of standard operations of BLAS linear algebra standard.

SCHEDULERS, MATRIX MULTIPLICATION, GAME THEORY, CLOUD COMPUTING, EQUILIBRIUM.

ЗМІСТ

	Ст.
ПЕРЕЛІК СКОРОЧЕНЬ	8
ВСТУП	9
 РОЗДІЛ 1 АНАЛІЗ ЗАДАЧІ ПЛАНУВАННЯ У ХМАРНОМУ СЕ-	
РЕДОВИЩІ	10
1.1 Планувальники у розподілених обчисленнях	10
1.2 CloudSim як засіб для симуляції хмарних обчислень	14
1.2.1 Сильні сторони	15
1.2.2 Слабкі сторони	16
1.2.3 CloudSim Plus	17
1.3 Аналіз існуючих робіт	18
Висновки до розділу	19
 РОЗДІЛ 2 ПОБУДОВА МАТЕМАТИЧНОЇ МОДЕЛІ МНОЖЕН-	
НЯ МАТРИЦЬ	20
2.1 Потокова модель задачі множення матриць	20
2.2 Аналіз штрафів від величини розбиття	22
2.3 Дискретна версія моделі обчислень множення матриць ...	25
2.4 Визначення з теорії ігор	27
2.5 Числові методи знаходження рівноваг Неша	29
2.5.1 Алгоритм "Support enumeration"	30
2.5.2 Алгоритм "Vertex enumeration"	31
2.6 Некооперативна ігрова модель планування множення ма-	
триць для двох користувачів	32
2.7 Імплементация імітаційної моделі	34
Висновки до розділу	36
 РОЗДІЛ 3 АНАЛІЗ РЕЗУЛЬТАТІВ СИМУЛЯЦІЙ	37
3.1 Структура симуляційного програмного забезпечення	37
3.2 Ілюстрація результатів симуляції	40
3.2.1 Симуляція для одного користувача	40
3.2.2 Аналіз штрафів за розбиття	41
3.2.3 Симуляція для двох користувачів	45
3.3 Застосування оптимізації до знаходження мінімуму	48
Висновки до розділу	50

РОЗДІЛ 4	Керування стартапом проекту	51
4.1	Опис ідеї проекту	51
4.2	Аналіз ринкових можливостей запуску стартап-проекту ..	55
4.3	Розробка ринкової стратегії проекту	64
4.4	Розроблення маркетингової програми стартап-проекту....	68
	Висновки до розділу	72
ВИСНОВКИ ПО РОБОТІ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ		
ДОСЛІДЖЕНЬ		74
ПЕРЕЛІК ПОСИЛАНЬ		75
ДОДАТОК А ІЛЮСТРАТИВНІ МАТЕРІАЛИ ДОПОВІДІ		77
ДОДАТОК Б ЛІСТИНГ КОДУ		81

ПЕРЕЛІК СКОРОЧЕНЬ

СПЗ - симуляційне програмне забезпечення

ОВ - обчислювальний вузол

ХС - хмарне середовище

ММ - множення матриць

ІМ - імітаційна модель

ОМ - обчислювальна мережа

ПП - програмний продукт

ВСТУП

Хмарні обчислення на даний момент вважається чимось достатньо простим та доступним, 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 будуть 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 першу 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 змінюється 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

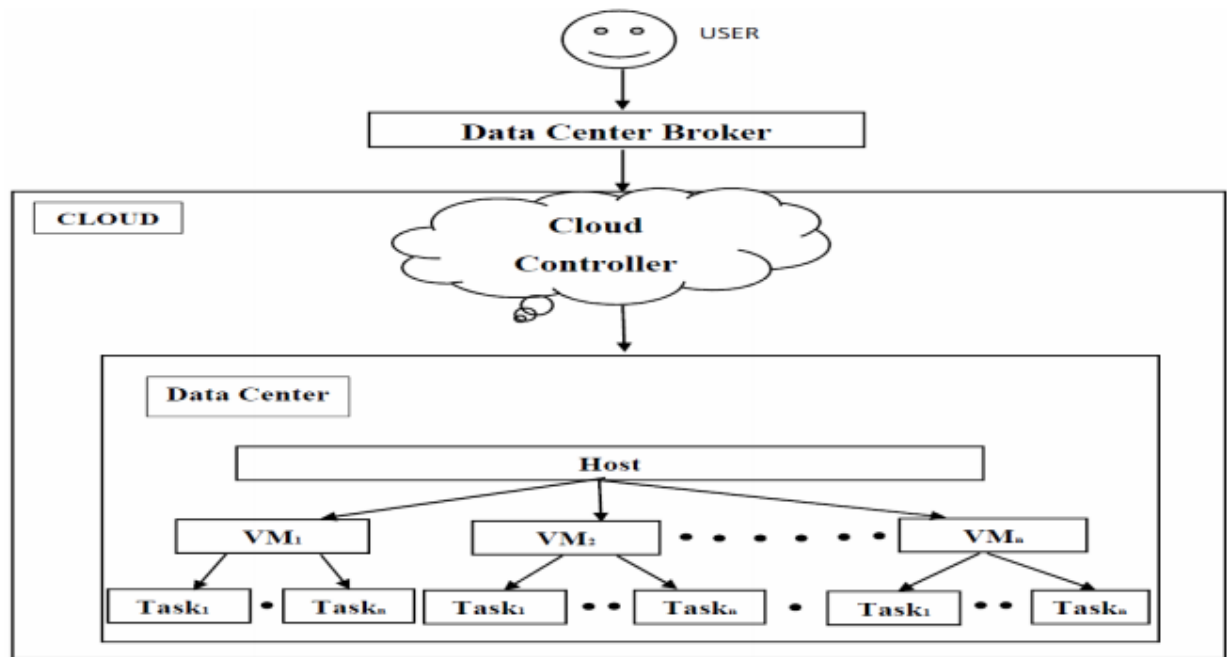


Рисунок 1.1 – Ілюстрація структури хмарної ОМ

На Рис. 1.1 проілюстрована структура простої хмарної системи з одним сервером. Брокер являється посередником між користувачем та хмарию 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 несамоостійні 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 обчислювальний вузол, який вміє лише отримувати вхідні дані, виконувати задачу та відправляти результат у дата центр. Часто буває таке, що декілька віртуальних машин працюють на одній фізичній машині, це звичайне явище, яке обумовлене тим, що дешевше на сервері із 64 чи 128 обчислювальними ядрами встановити багато віртуальних машин по 4 обчислювальних ядра. Часто користувачами не потрібні багато-ядерні машини.

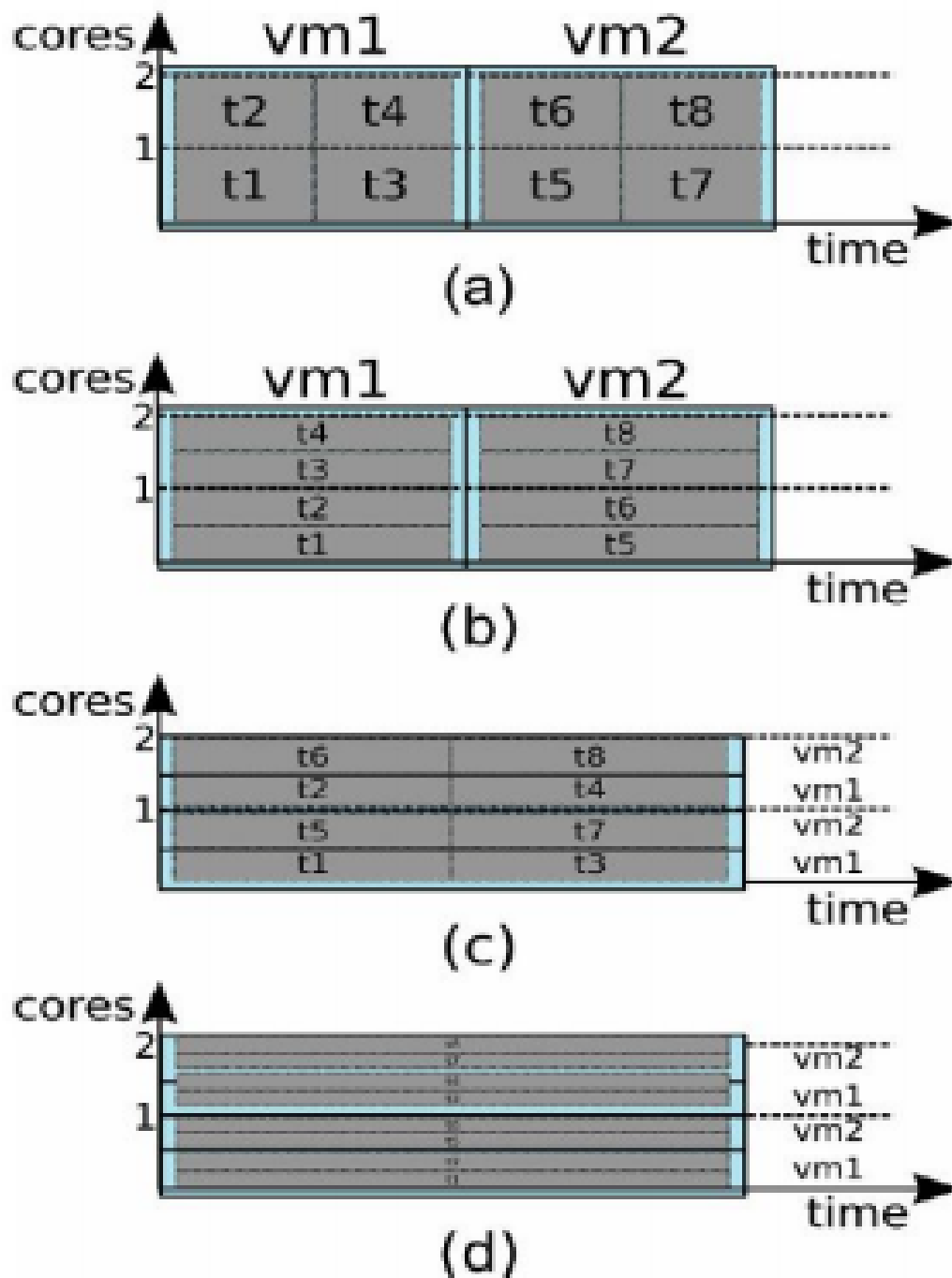


Figure 2. Effects of scheduling policies on task execution:
(a) Space-shared for VMs and Tasks, (b) Space-share for VMs and Time-shared for tasks, (c) Time-shared for VMs, Space-shared for tasks, and (d) Time-shared for both VMs and Tasks [7]

Рисунок 1.2 – Ілюстрація планування задач для time та space shared

Існує дві політики планування: space shared та time shared Рис. 1.2. У політиці space shared задачі у черзі розділяють між собою логічні ядра процесора обчислювального вузла і у випадку якщо усі ядра зайняті, то задача чекає звільнення ядра. Саме ця політика і використовується 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 додаються до черги та очікують свого запуску свого виконання

Time shared політика в свою чергу означає що задачі у черзі 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 посилення на алгоритм планування Round Robin.

Етапи роботи time 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 додаються до черги та зразу запускаються і працюють у режимі спільного використання процесорного часу

З цього можна зробити висновки, що space shared політика задає послідовне виконання задач обчислювальними вузлами, а time shared - паралельне. Time shared політика рідко використовується у багатопроцесорних середовищах коли ОВ розташовані далеко один від одного. Також вона потребує спеціальної технології переключення між задачами, яка забезпечить дуже швидке переключення між задачами. Але такі технології зазвичай спеціалізовані і непридатні для виконання будь-яких задач.

У цій роботі використовується політика space shared отскільки саме вона передбачає виконання лише одної задачі на обчислювальному вузлі у кожен момент часу.

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 ІТ-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 налаштовувати 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. Додавання 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 створити 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 як 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 симуляційний 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 потрібно 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 матрицями: $C := A * B + C$. 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], в 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 впливає 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 робіт на тематику ефективного множення матриць це [8]. В роботі розглядається операція $y := Ax$ для матриці $A \in \mathbb{R}^{m \times n}$ 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 конфігурації обчислювального середовища на базі Amazon AWS завдяки побудові моделі. Знаходиться компроміс між вибором набору кластерів із запропонованих тарифних планів та швидкістю виконання операцій. Для пошуку оптимальної конфігурації запропонований евристичний пошук. На основі цієї роботи була запропонована покращена схема кодування та проведені порівняння з існуючими [9].

Висновки до розділу

У цьому розділі було розглянуто структуру принципи роботи об'єкта дослідження - хмачного середовища. ХС у наші дні це дуже поширений інструментр для виконання задач користувачів будь-якої складності та вважається, що воно замінить у певному сенсі домашні обчислювальні машини.

Розглянуті популярні засоби для симуляції хмарного середовища - CloudSim, CloudSim Plus та GridSim. Ці засоби дуже популярні та часто використовуються у бакалаврських чи магістерських роботах для тестування спроектованих планувальників. Ці СПЗ не підходять для даної роботи оскільки вони написані на мові Java та мають низьку швидкодію у випадку великої кількості задач.

Із огляду літератури можна зробити висновки, що тема актуальна і нові дослідження проводяться навіть для таких простих задач як множення матриць чи добутку матриці на вектор.

РОЗДІЛ 2 ПОБУДОВА МАТЕМАТИЧНОЇ МОДЕЛІ МНОЖЕННЯ МАТРИЦЬ

2.1 Потокова модель задачі 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:

$$T(N, n) \longrightarrow \min \quad (2.1)$$

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 корелюють 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 такі 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:

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

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

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

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

Згідно 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:

$$\mu_X(p) = \inf \lambda > 0 : p \in \lambda X \quad (2.4)$$

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:

$$R(n) = \frac{R}{N n^2} \quad (2.5)$$

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 на-

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

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

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 :

$$d_i = \frac{x_i}{\sum_{i=1,\dots,m} x_i} = \frac{x_i}{k} \quad (2.7)$$

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

Перепишемо 2.6 за допомоги d_i :

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

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:

$$T_i = d_i \frac{N^3}{p_i} + d_i \frac{N^2 (1 + 2 * \frac{N}{n})}{q} + d_i \frac{N^2}{n^2} l \quad (2.11)$$

$$T_s(x, X(n)) = \max_{i=1, \dots, m} T_i$$

Виділимо окремі складові T_i :

$$A_i = \frac{N^3}{p_i}$$

$$B_i = \frac{N^2 (1 + 2 * \frac{N}{n})}{q}$$

$$C_i = \frac{N^2}{n^2} l$$

$$T_i = d_i (A_i + B_i + C_i) \quad (2.12)$$

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.

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

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 масштабом оскільки при малих n_1 це відношення дуже велике.

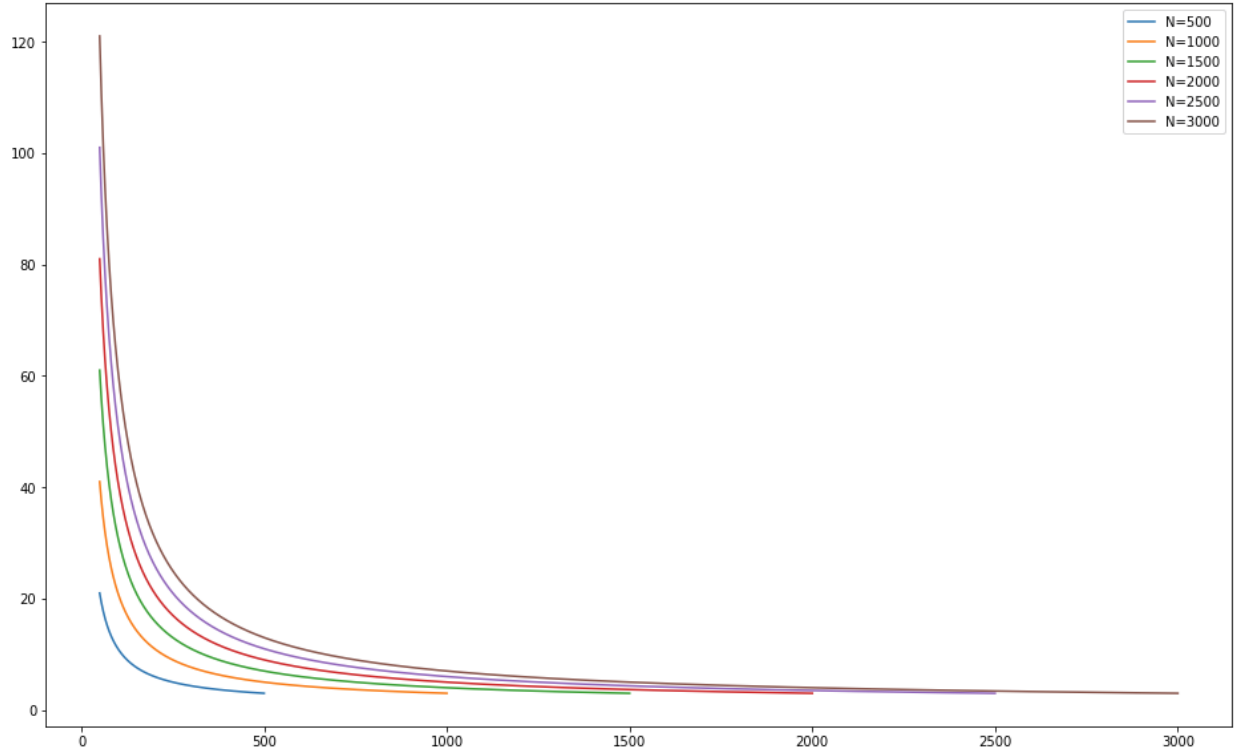


Рисунок 2.1 – Графік залежності відношення $\frac{B_i^{n_1}}{B_i^N}$ від $n_1 \in [50, N]$ при фіксованих N

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 $X(n)$:

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

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 $extr_2$. \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:

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

Також варто пояснити саму суть пошуку найкращого розбиття. Основна проблема у тому, що для великих розбиттів планування може бути недостатньо 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 виконуються повністю паралельно, але четверта буде виконуватись на одному вузлі, в той час як інші 2 будуть 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 розглянути 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 Визначення з теорії ігор

Нехай задана гра для n осіб (S, f) , де S - профіль стратегій $S = S_1 \times S_2 \times \dots \times S_n$ та f - функція виграшу $f = (f_1(x), f_2(x), \dots, f_n(x))$ для $x \in S$.

Для деякого набору стратегій гравців $x \in S$ позначимо за x_{-i} вектор x без складової i , тобто стратегіями усіх гравців крім i -го.

Для i -го гравця кажуть, що x_i^1 домінує x_i^2 , $x_i^1, x_i^2 \in S_i$, якщо $f_i(x_i^1, x_{-i}) > f_i(x_i^2, x_{-i})$, $x_{-i} \in S_1 \times S_2 \times \dots \times S_{i-1} \times S_{i+1} \times \dots \times S_n$. Також кажуть, що x_i^2 домінована x_i^1 . Також домінування може бути нестрогим, для цього у визначенні потрібно використовувати нестрогу нерівність.

Таблиця 2.1 – Проста гра двох осіб у матричній формі

	1	2	3
1	1, 2	2, 2	3, 5
2	2, 1	3, 4	4, 3
3	3, 5	3, 7	5, 8

Простіше за визначення демонструвати для ігор двох гравців, оскільки їх можна представити у вигляді матриць 2.1. У ній для першого гравця стратегія "2" домінує "1" та стратегія "3" нестрого домінує "2". Домінуючі стратегії допомагають при аналізі гри, оскільки фактично домінування одної стратегії "А" над "Б" дозволяє не розглядати стратегію "Б" у подальшому. У випадку раціональності гравця він ніколи не вибере доміновану стратегію.

Неформальне визначення рівноваги за Нешем звучить так: у точці рівноваги жоден з гравців не може збільшити свій виграш шляхом вибору іншої стратегії при фіксованих стратегіях інших гравців.

Запишемо формальне визначення рівноваги за Нешем.

x^* є рівновагою за Нешем якщо:

$$\forall i, x_i \in S_i : f_i(x_i^* | x_{-i}^*) > f_i(x_i, x_{-i}^*) \quad (2.17)$$

Найпопулярнішою демонстрацією рівноваги за Нешем є дилема в'язня.

Таблиця 2.2 – Ілюстрація дилеми в'язня як гри у матричній формі

	Співпраця	Зрада
Співпраця	2 , 2	10 , 1
Зрада	1 , 10	5 , 5

Таблиця 2.2 показує типові визначення дилеми в'язня. У таблиці записані програші учасників, тобто кожен з них намагається мінімізувати свій програв і таким чином максимізувати виграш.

Пояснюється за ситуація тим, що двох в'язнів запитують про свідчення проти співника. Якщо один в'язень надає свідчення про іншого, то йому трохи вкорочують строк перебування у в'язниці, а другому навпаки збільшують. Також якщо вони обидва будуть мовчати, то їм однаковий строк, який відносно виглядає оптимально порівняно з іншими. Проте якщо вони обидва будуть свідчити, то їм обом строк 5 років.

Як можна побачити з матриці, домінуючих стратегій немає. Проте є рівнована за Нешем. Рівновага у парі стратегій ("Зрада", "Зрада"). Найцікавіше тут те, що точка рівноваги показує програші значно більші ніж у стратегії ("Співпраця", "Співпраця"). Проте логіка при розгляданні рівноваги за Нешем полягає у пошуку найкращої відповіді на певну фіксовану стратегію опонента.

Ще одне неформальне визначення рівноваги за Нешем є "найкраща відповідь на найкращу відповідь".

Особливу увагу надають цій рівновазі саме через її застосування у теорії конфліктних ситуацій, оскільки саме точки рівноваги за Нешем часто описують ситуації, з яких важко вийти. Причому вихід з таких ситуацій не вигідний нікому, хоч і існують інші точки у грі, які можуть мати значно більші виграші.

2.5 Числові методи знаходження рівноваг Неша

Для гри двох гравців визначені деякі методи знаходження рівноваг Неша. У [12] проводиться аналіз складності задачі пошуку рівноваг, та доведено, що пошук це задача PPAD складності - "Polynomial Parity Arguments on Directed graphs". Це впливає в першу чергу з того, що рівновага Неша обов'язково існує у мішаних стратегіях. Задачі PPAD характеризуються саме тим, що їх розв'язок обов'язково існує, хоч його і складно знайти.

У книзі [13] описані алгоритми для знаходження рівноваг і основними з них є "Support enumeration" та "Vertex enumeration". Ці алгоритми реалізовані і бібліотеці "NashPy" [14]. Опишемо їх.

Основні визначення.

Мішаною стратегією гравця з профілем стратегій S називають вектор $k = \dim S, x \in R^k : 0 \leq x_i \leq 1, \sum_{i=1}^k x_i = 1$. Тобто гравець вибирає чисті стратегії із S з ймовірностями x_i . Носієм мішаної стратегії x назовемо множину $T = \{i : x_i > 0\}$.

Біматричною грою назовемо гру двох осіб у матричній формі, де номери строк відповідають за стратегії першого гравця, а номери колонок за стратегії другого. Позначимо профілі стратегій гравців за $S_1, S_2 : \dim S_1 = m, \dim S_2 = n$. Тоді виграші задаються двома матрицями $A, B \in \mathbb{R}^{m \times n}$ для гравця 1 та 2 відповідно.

Найкращою відповіддю (best response) гравця 1 на мішану стратегію гравця 2 y є така мішана стратегія x , що максимізує його очікуваний виграш: $x^T A y$. Аналогічно визначається найкраща відповідь гравця 2 мішана стратегія y на мішану стратегію гравця 1: $y : x^T B y \rightarrow \max$.

Визначення рівноваги Неша у мішаних стратегіях:

x, y — мішані стратегії гравців 1 та 2 відповідно :

x — найкраща відповідь гравця 1 на y (2.18)

y — найкраща відповідь гравця 2 на x

Тобто повторюється одне з визначень рівноваги за Нешем - "найкраща відповідь на найкращу відповідь".

Запишемо визначення найкращої відповіді у більш зручному вигляді для подальших описів алгоритмів. Нехай x, y - машані стратегії гравців 1 та 2 відповідно, тоді:

$$\begin{aligned}
 M &= 1, \dots, m \\
 \left(x \text{ найкраща відповідь на стратегію } y \right) &\Leftrightarrow \\
 &\Leftrightarrow \left(\forall i \in M : x_i > 0 \Rightarrow (Ay)_i = u = \max\{(Ay)_k | k \in M\} \right)
 \end{aligned} \tag{2.19}$$

Доведення цього факту дуже просте:

$$\begin{aligned}
 x^T Ay &= \sum_{i=1}^m x_i (Ay)_i = \sum_{i=1}^m x_i (u - (u - (Ay)_i)) = \\
 &= u - \sum_{i=1}^m x_i (u - (Ay)_i)
 \end{aligned} \tag{2.20}$$

З 2.20 ми маємо, що $x^T Ay \leq u$ оскільки $x_i \geq 0$ та $u - (Ay)_i \geq 0 \forall i = 1, \dots, m$. Рівність досягається тільки у випадку виконання імплікації $x_i > 0 \Rightarrow (Ay)_i = u$. Аналогічно до $u = \max\{(Ay)_k | k \in \{1, \dots, m\}\}$ визначається і $v = \max\{(x^T A)_k | k \in \{1, \dots, n\}\}$.

Гру називають недегенеративною, якщо в ній не існує такої мішаної стратегії з носієм потужності k , на яку є більше ніж k кращих відповідей у чистих стратегіях. Якщо це не виконується, то гру називають дегенеративною.

Також є твердження, що для будь-якої рівноваги Неша (x, y) у недегенеративній біматричній грі x та y мають однакові розмірності носіїв.

Саме завдяки цим фактам і з'явився метод "Support enumeration".

2.5.1 Алгоритм "Support enumeration"

Позначимо за $M = 1, \dots, m, N = 1, \dots, n$ та $l = \min\{m, n\}$.

Для $k = 1, \dots, l$:

1. Вибираємо наступну пару підмножин $I \subset M, J \subset N$.
2. Розв'язуємо систему $\sum_{i \in I} x_i b_{ij} = v$ для $j \in J$ та з обмеженням $\sum_{i \in I} x_i = 1$.
3. Розв'язуємо систему $\sum_{j \in J} a_{ij} y_j = u$ для $i \in I$ та з обмеженням $\sum_{j \in J} y_j = 1$.
4. Перевіряємо, що $x \geq 0$ та $y \geq 0$ і виконується [2.19](#).

Таким перебором ми в шукаємо усі можливі пари рівноваг у мішаних стратегіях з різними потужностями носіїв починаючи з 1, тобто рівноваг у чистих стратегіях. Проте метод працює дуже довго навіть для малих матриць, оскільки переборів різних комбінацій проводиться дуже багато. Фактично потрібний цикл з вирішенням систем лінійних рівнянь для кожної пари.

2.5.2 Алгоритм "Vertex enumeration"

Цей алгоритм базується на теорії політопів. Вводяться такі політопи: $P = \{x \in \mathbb{R}^n | x \geq \mathbf{0}, B^T x \leq \mathbf{1}\}$ та $Q = \{y \in \mathbb{R}^m | Ay \leq \mathbf{1}, y \geq \mathbf{0}\}$. Ці політопи мають повну розмірність оскільки припускається, що A, B^T мають невід'ємні елементи та не мають повністю нульових стовбчиків.

Алгоритм проходить по усім парам вершин x з $P - \{\mathbf{0}\}$ та y з $Q - \{\mathbf{0}\}$. Якщо пара (x, y) повністю пронумерована, то це рівновага за Нешем. Повністю пронумерована значить, що усі значення з множини $M \cup N$ зустрічаються у при номерах координат x чи y .

2.6 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 результату усіх відісланих ним задач, тобто момент, коли він отримає останній блок матриці результату.

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 як можна 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:

- Стратегії користувачів $n_i, i = 1, \dots, k$ впорядковані за зростанням
- Стратегії користувачів $n_i \in \text{дільники } N$

- 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^*

Розглянемо планувальники min-min та min-max.

Нехай користувачі вибрали розрізання n_1, n_2 , які являються їх стратегіями у грі. Користувачі відправляють свої задачі, та отримують результати. Часи повернення усіх задач позначимо за $T_1(n_1, n_2), T_2(n_1, n_2)$, які є власне програшами користувачів.

Враховуючи специфіку планувальників помітимо такі властивості часів повернення від розрізань:

- Якщо $n_1 < n_2$, то виграш першого користувача дорівнює $T_d(n_1)$, а другого $T_d(n_1) + T_d(n_2)$
- Якщо $n_1 > n_2$, то виграш першого користувача дорівнює $T_d(n_1) + T_d(n_2)$, а другого $T_d(n_2)$
- Якщо $n_1 = n_2$, то виграші користувачів однакові та дорівнюють $2T_d(n_1)$

Такі властивості з'являються через те, що ці планувальники пріоритизують задачі з меншим обсягом роботи і тому користувач, який вибрав розрізання менше, повністю окупує обчислювальні ресурси. Задачі другого користувача підуть на виконання лише після того як усі задачі першого користувача будуть вилучені з черги очікування. Це явище і породжує конфлікт між двома гравцями, оскільки кожен з них хоче мінімізувати свій програш - час виконання власних задач. Пороте у минулих підрозділах ми розглянули форму штрафів для різних розбиттів і зменшувати розмір розбиття починаючи з деякого моменту вносить дуже великий штраф за пересилання. Особливо це помітно у випадках коли ми розглядаємо розбиття $n : n \mid N$.

Твердження.

Нехай виконується нерівність

$$2T_d(n_{j^*}) \leq T_d(n_{j^*-1}) \quad (2.21)$$

тоді пара стратегій (n_{j^*}, n_{j^*}) - рівновага Неша.

Доведення.

Розглянемо стратегію (n_{j^*}, n_{j^*}) у випадку якщо нерівність 2.21 виконується, тоді отримуємо такі факти:

1. За властивостями планувальника якщо будь-який з користувачів змінить свою стратегію на $n : n > n_{j^*}$, тоді його програш тільки збільшиться, оскільки тоді перший користувач буде мати розмір розрізання менший і його виконані задачі в першу чергу.
2. При зміні стратегії на n_{j^*-1} користувач виграє право першочергового виконання саме його набору задач, проте завдяки нерівності 2.21 ця зміна стратегії не зменшить його програш.
3. При зміні стратегії на $n : n < n_{j^*-1}$ штрафні частини функції часу вже більше впливають на час і тому у реальних випадках ці стратегії дадуть значно більший штраф. Цей ефект краще всього буде проілюстровано у практичній частині в наведеній таблиці окремих компонент часу виконання в залежності від розбиття.

Наслідок.

Отримана рівновага Парето неефективна, оскільки $T_1(n_{j^*}, n_{j^*}) < T_1(n_{j^*-1}, n_{j^*-1})$ та $T_2(n_{j^*}, n_{j^*}) < T_2(n_{j^*-1}, n_{j^*-1})$.

2.7 Імплементация імітаційної моделі

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

Розглянемо задачу множення двох $N \times N$ матриць $M1$ та $M2$. Позначимо за n кількість рядків матриці $M1$ та відповідно стовбців матриці $M2$. Таким чином отримуємо 2 підматриці $n \times N$ та $N \times n$ 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 r . Тому до основних задач ще потрібно додати задачі множення матриць $t \times N$ та $N \times n$, матриць $n \times N$ та $N \times t$ і одну задачу множення $t \times N$ та $N \times t$.

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 виразити у одиницях операцій додавання як:

$$Complexity(N1, N2, N3) = N1 * N3 * (N2 * AM + N2 - 1) \quad (2.22)$$

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 :

$$T_{processing} = \frac{C}{P} \quad (2.23)$$

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

$$T_{transfer} = latency + \frac{N1 * N2 + N2 * N3 + N1 * N3}{bandwidth} \quad (2.24)$$

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

Загальний час на обробку задачі отримується з урахуванням 2.23 та 2.24:

$$T = T_{processing} + T_{transfer} \quad (2.25)$$

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 останньої його задачі.

Висновки до розділу

У розділі побудована модель задачі множення матриць у гетерогенному розподіленому середовищі для одного та двох користувачів з урахуванням штрафів за пересилки та затримок. Запропонована потокова модель, яка спочатку сформульована для неперервного випадку, а потім звужена до дискретного. Таким чином модель дуже точно дозволяє оцінити час виконання задач у розподіленому середовищі.

Розглянуті базові визначення з теорії ігор та представлені числові методи по знаходженню рівноваг Неша у біматричній грі. Базовий метод "Support enumeration" простий у розумінні, проте складний в сенсі обчислювальної складності. Другий розглянутий метод "Vertex enumeration", який дозволяє більш ефективно перебирати можливі пари стратегій для перевірки їх на рівновагу.

Проведено аналіз ігрової задачі множення матриць двома користувачами у спільному гетерогенному розподіленому середовищі. Сформульовано твердження, яке потрібно перевірити практично на побудованій симуляційній системі.

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.

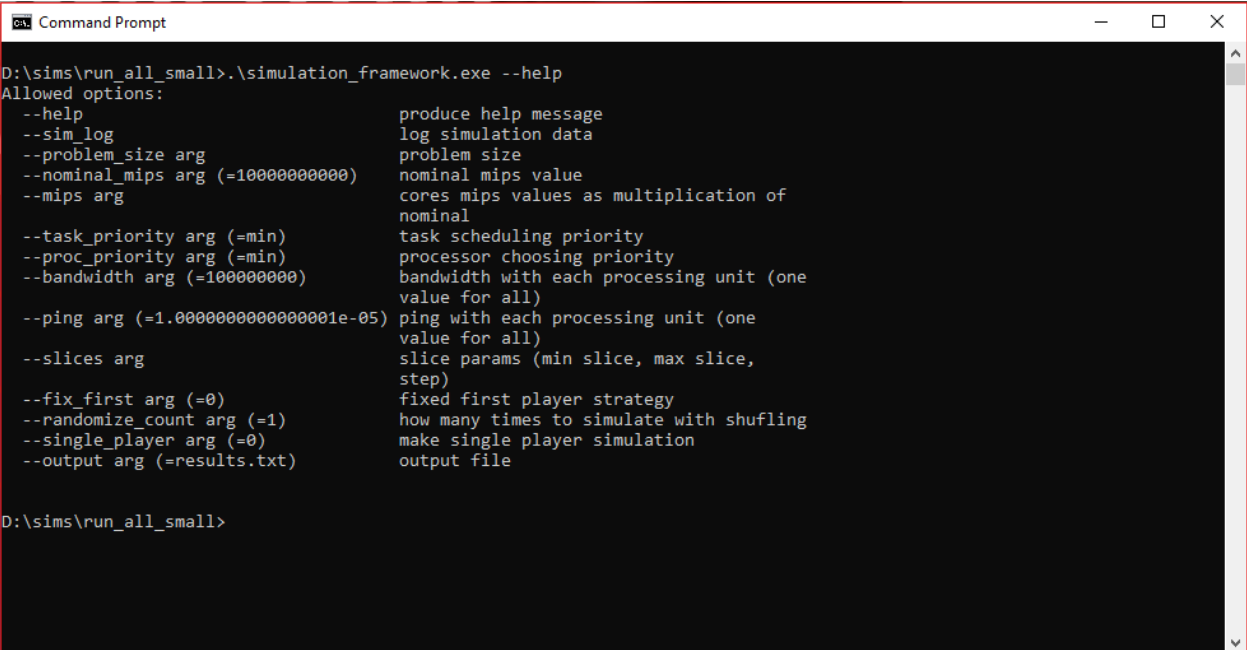
Модуль обробки параметрів дозволяє задавати розмір матриць, режим одноно гравця чи двох, межі перебору стратегій розрізання, параметри планувальника, характеристики обчислювальних модулів, параметрів мережі – bandwidth та latency без перековпіляції програми через параметри командного рядка.

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:

1. Перемішування списку очікуючих задач з метою емуляції отримання задач у випадковому порядку.
2. Сортуння списку очікуючих задач по складності у відповідності до пріоритетності задач та списку обчислювальних вузлів по потужності у відповідності до пріоритетності обчислювальних вузлів. Наприклад для minmax планувальника список очікуючих задач буде відсортованим від простої до складної, а список обчислювальних вузлів від потужного до повільного.
3. 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 пріоритетом по найменшому часу завершення.

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



```

D:\sims\run_all_small>.simulation_framework.exe --help
Allowed options:
--help                produce help message
--sim_log             log simulation data
--problem_size arg    problem size
--nominal_mips arg (=1000000000) nominal mips value
--mips arg            cores mips values as multiplication of
                        nominal
--task_priority arg (=min) task scheduling priority
--proc_priority arg (=min) processor choosing priority
--bandwidth arg (=100000000) bandwidth with each processing unit (one
                        value for all)
--ping arg (=1.0000000000000001e-05) ping with each processing unit (one
                        value for all)
--slices arg          slice params (min slice, max slice,
                        step)
--fix_first arg (=0)  fixed first player strategy
--randomize_count arg (=1) how many times to simulate with shuffling
--single_player arg (=0) make single player simulation
--output arg (=results.txt) output file

D:\sims\run_all_small>

```

Рисунок 3.1 – Скріншот вікна з описом параметрів програми

На Рис. 3.1 зображено основний інтерфейс програми. Програма має деякі обов'язкові параметри та опціональні - тобто такі, для яких вписані значення за вмовчанням, але при потребі їх можна змінити.

Список обов'язкових параметрів:

1. *problem_size* - цей параметр відповідає за визначення розміру матриць, симуляція яких буде проводитись. Розмір визначається одним числом оскільки ми матриці вважаємо квадратними - $N \times N$.

2. *mips* - цей параметр приймає список чисел, які визначають кількість обчислювальних вузлів у розподіленому середовищі та їх потужності. Для простоти було вирішено приймати потужності як коефіцієнти при деякій номінальній потужності, яка задається опціональним параметром *nominal_mips*. Наприклад параметри "1 1.5 5 6" задають 4 обчислювальних вузла з потужностями $1 * \text{nominal_mips}$, $1.5 * \text{nominal_mips}$, $5 * \text{nominal_mips}$, $6 * \text{nominal_mips}$
3. *slices* - параметер відповідає за вибір набору стратегій, симуляція яких буде проводитись. У випадку двох гравців це буде декартовий добуток цієї множини с собою.

Список опціональних параметрів:

1. *nominal_mips* - задає номінальний множник потужностей ОВ. За вмовченням вибраний такий, що відповідає за множення матриць розмірів 2000 за 1.6 секунди, що відносно відповідає потужності одного ядра у сучасних комп'ютерах.
2. *bandwidth* - регулює ширину каналу для передачі даних з кожним ОВ.
3. *ping* - виставляє штраф за з'єднання.
4. *single* - булевий параметр $\{0, 1\}$, який дозволяє проводити симуляції лише для одного користувача.
5. *task_priority* - визначає пріоритетність вибору задач. Може приймати 2 значення - *min* чи *max*. При виборі *min* спочатку виконуються задачі з меншим обсягом обчислень, а для *max* навпаки - з більшим. За вмовчуванням береться режим *min*.
6. *proc_priority* - аналогічно до *task_priority* визначає правило вибору вільного процесора. Для *min* задача планується на вільний процесор з найменшою потужністю, для *max* з найбільшою.

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.

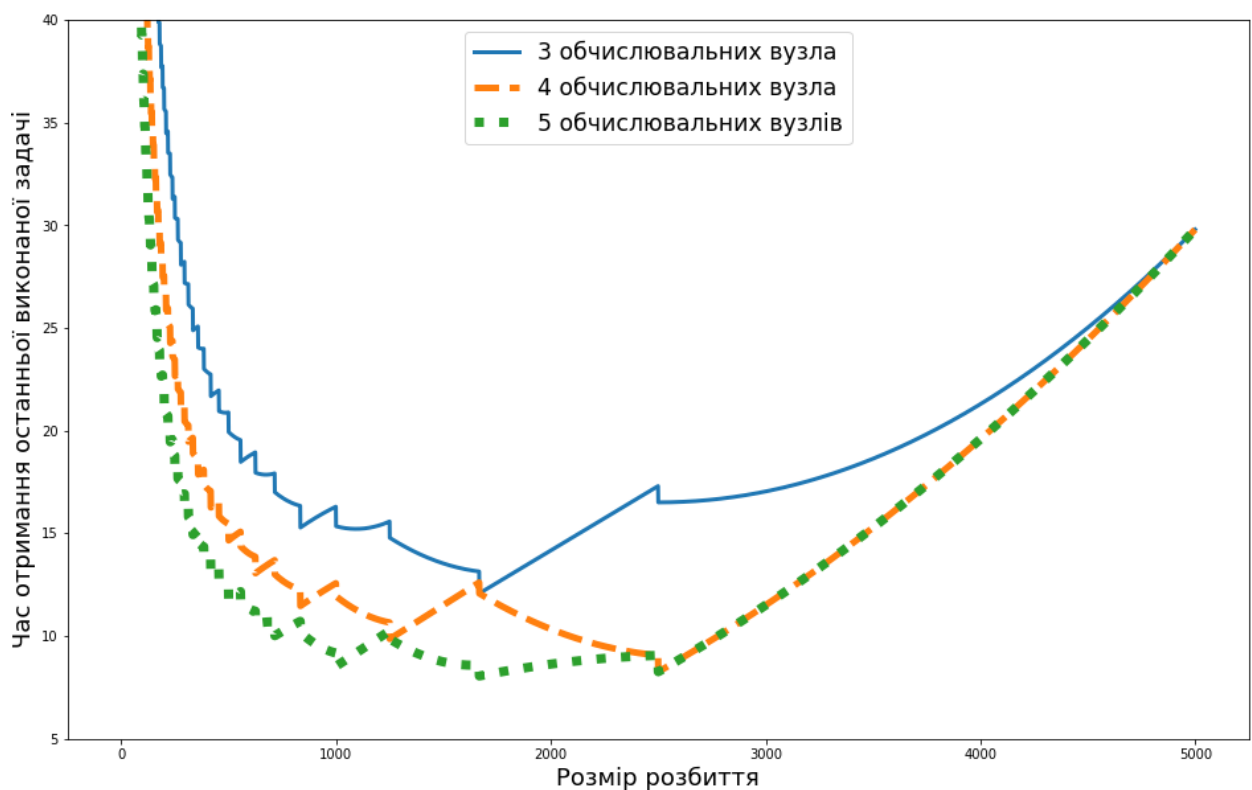


Рисунок 3.2 – Графік залежності часу виконання всіх задач користувача від розміру розрізання для різних кількостей обчислювальних вузлів

На Рис. 3.2 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.

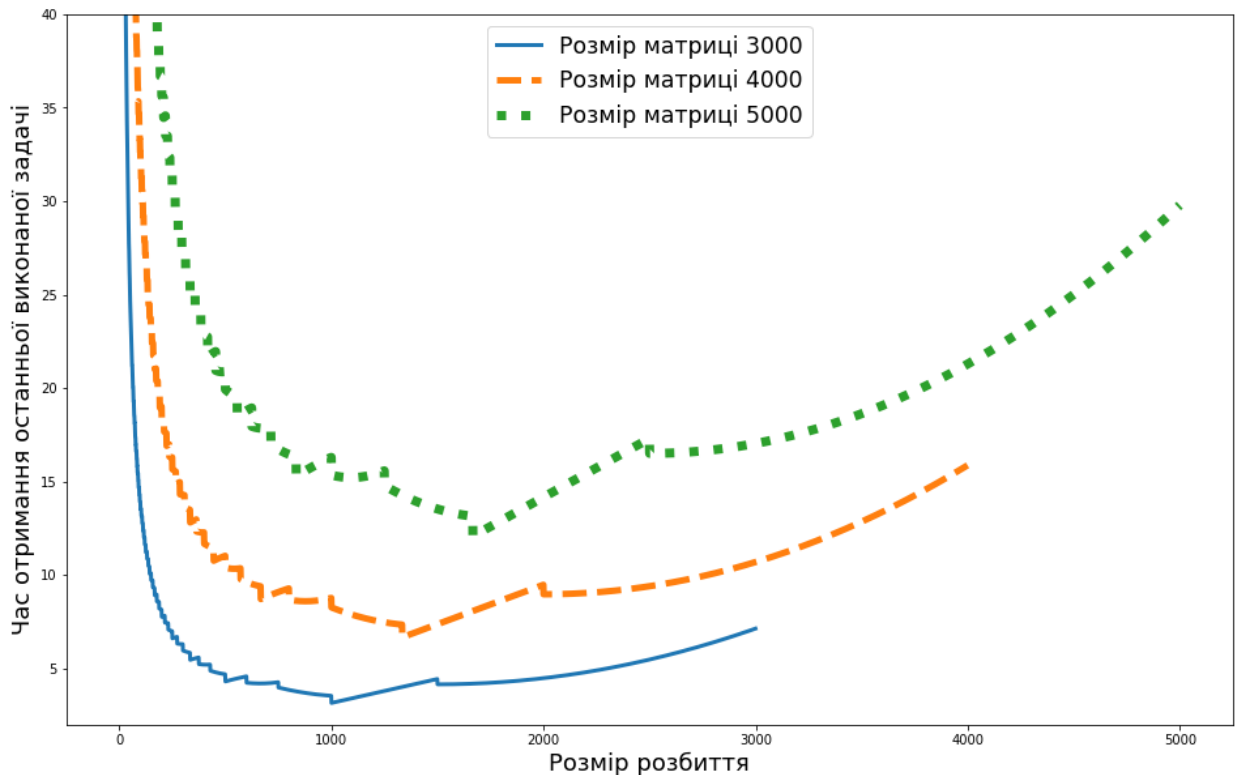


Рисунок 3.3 – Графік залежності часу виконання всіх задач користувача від розміру розбиття для різних розмірів матриць

На Рис. 3.3 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 пропорційна 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 досліджуваних параметри це: ping , bandwidth та mips . Якщо поставити $\text{bandwidth} = \inf$ та $\text{mips} = \inf$, тоді складові часу, які відповідають за передачу даних та саме обчислення,

обнуляться і ми будемо мати чисто лише час, який був спричинений затримками пакетів.

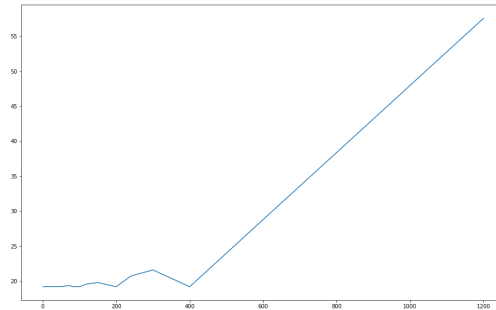
Проводити експерименти будемо для конфігурацій із заниженими потужностями обчислювальних 3х вузлів $6e7$ операцій/секунду, малим значенням пропускної здатності $8e7$ біт/секунду та затримкою пакетів в 1 мілісекунду.

Таблиця 3.1 наглядно показує для випадку множення матриць розмірів 1200 штрафів. Також можемо побачити велику різницю між 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.

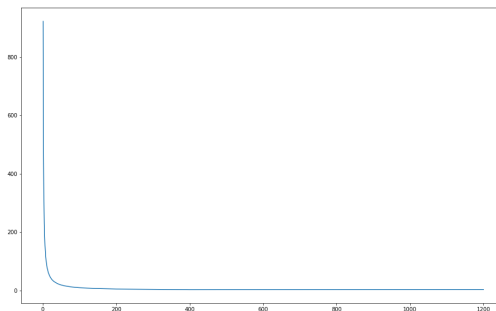
Таблиця 3.1 – Таблиця штрафів від величини розрізання

	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

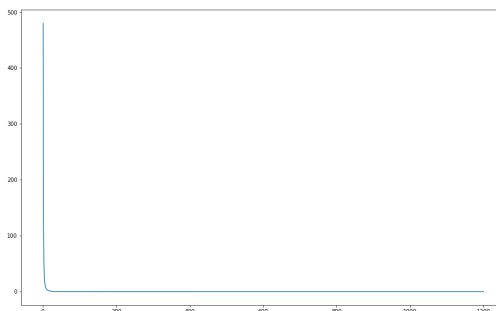
Розглянемо окремо графіки трьох складових загального часу: виконання задач, передачі даних та затримок.



(а) Графік часу обчислень від розміру розрізання



(б) Графік часу передачі даних від розміру розрізання



(в) Графік часу затримок від розміру розрізання

Рисунок 3.4 – Three subfigures

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

3.2.3 Симуляція для двох користувачів

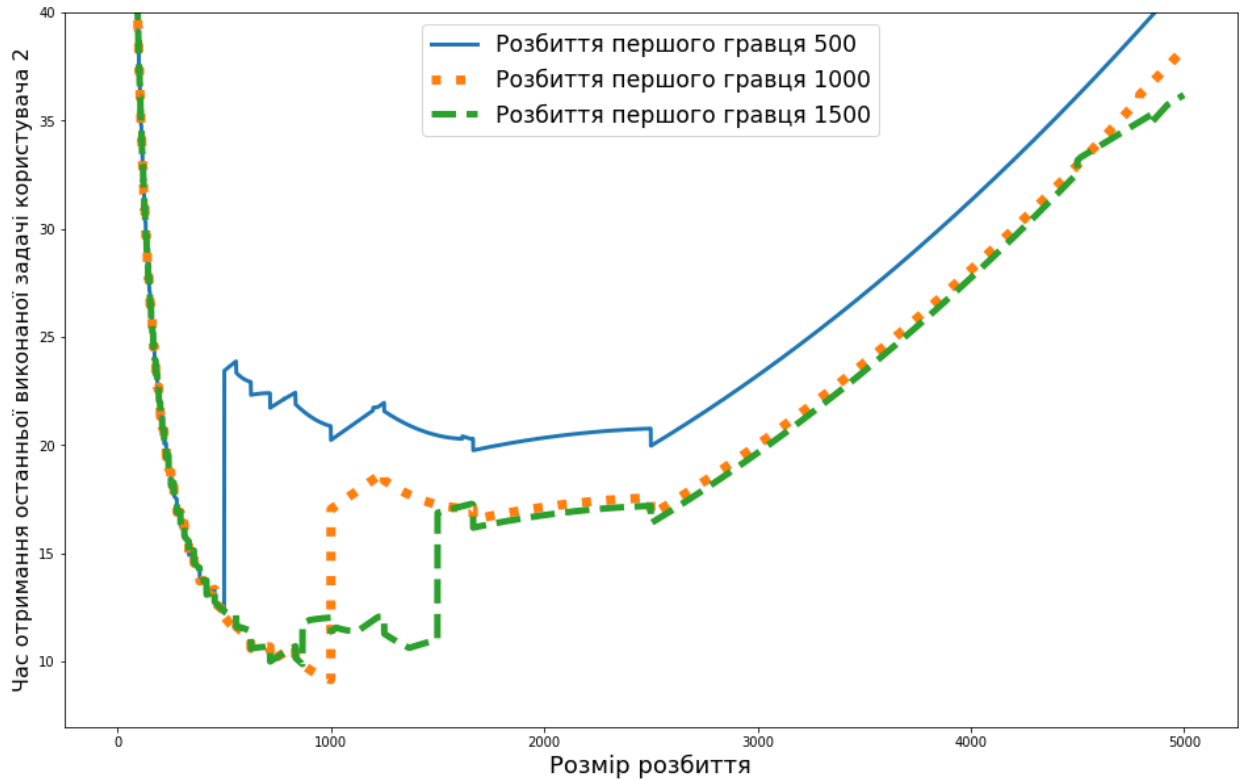


Рисунок 3.5 – Графік залежності часу виконання всіх задач другого користувача від розміру його стратегії розрізання при фіксованих стратегіях першого користувача

На Рис. 3.5 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 *min**min* та *min**max* оскільки вони в першу чергу виконують найлегші задачі, тому користувач, що вибрав менше розбиття, має менший час виконання усіх його задач.

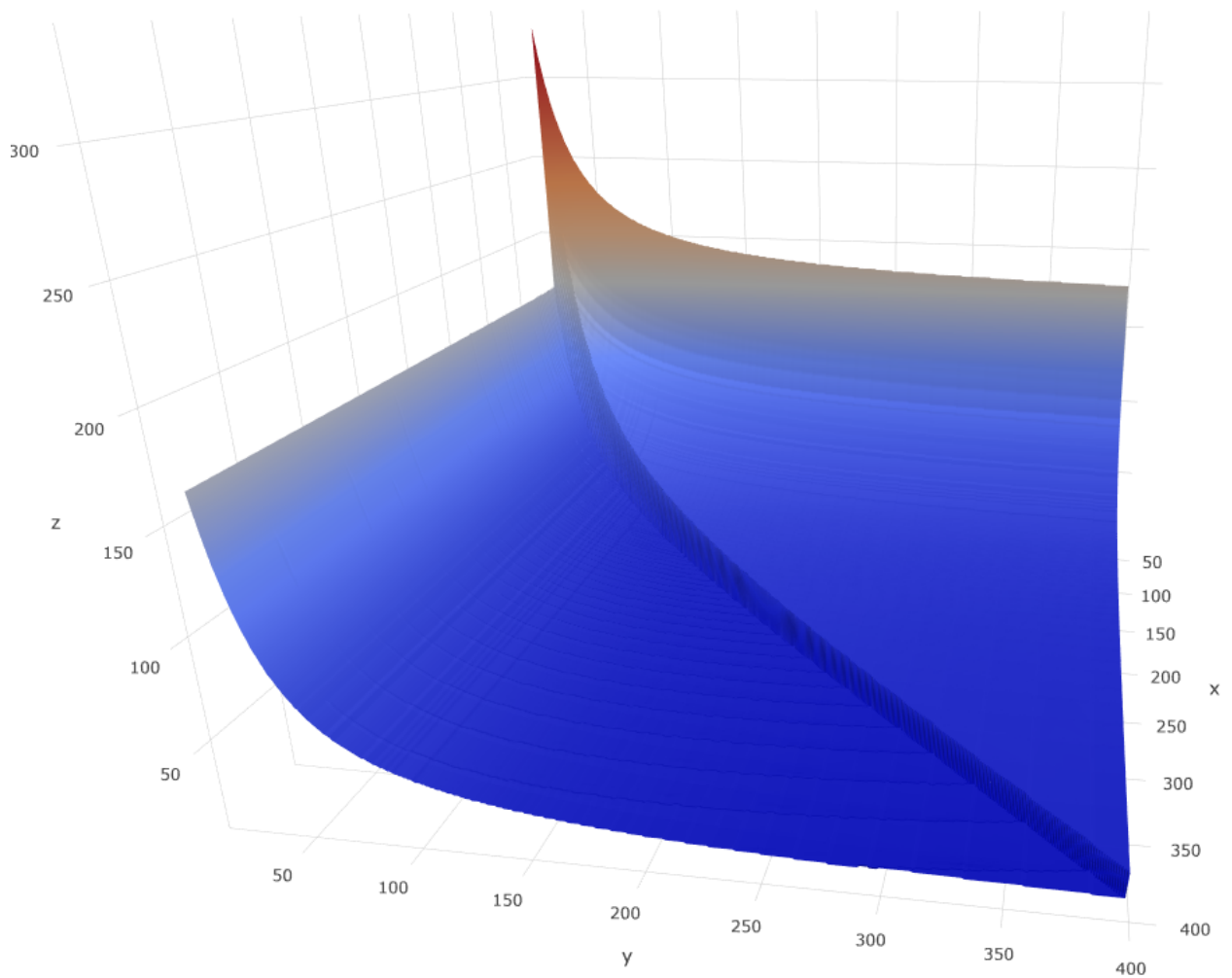
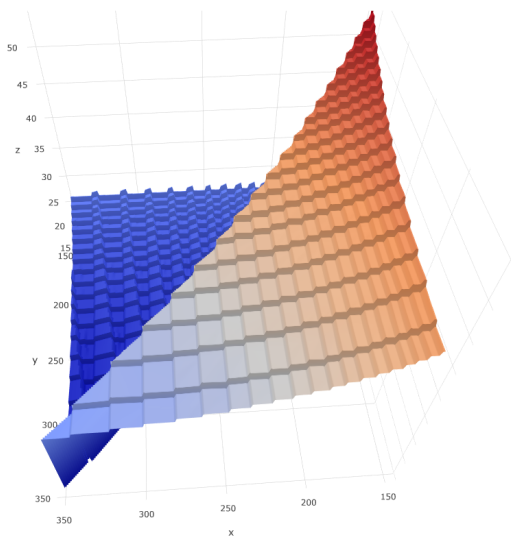
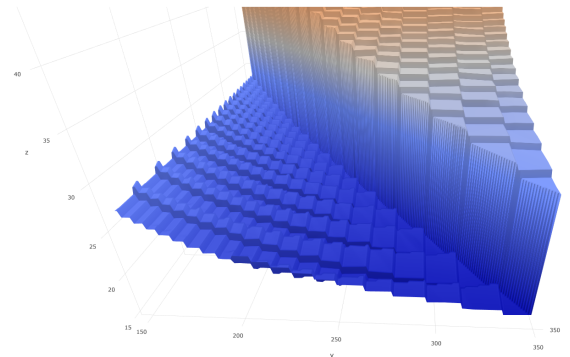


Рисунок 3.6 – Графік залежності часу виконання всіх задач другого користувача для всіх комбінацій стратегій обох користувачів з відрізка $[20, 400]$

На перший погляд поверхня, яка отримана шляхом симуляції усіх можливих пар стратегій обох користувачів з відрізка $[20, 400]$, може здаватися гладкою та випуклою Рис. 3.6. Проте, пам'ятаючи природу графіків при фіксованій стратегії першого користувача на Рис. 3.5, слід подивитися на Рис. 3.6 більш ретельно, наприклад побудувати поверхню усіх комбінацій стратегій з відрізка $[150, 350]$.



(а) Верхня частина графіка



(б) Нижня частина графіка

Рисунок 3.7 – Графік залежності часу виконання всіх задач другого користувача для всіх комбінацій стратегій обох користувачів з відрізка $[150, 350]$, 3.7a - фокус на верхню частину поверхні, 3.7б - фокус на нижню частину поверхні

При кращій деталізації можна чітко побачити, що на Рис. 3.7a спостерігається форма сходинок по всій верхній частині графіка і вона не така проблемна, як нижня частина, показана на Рис. 3.7б. 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.

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

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

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.

3.3 Застосування оптимізації до знаходження мінімуму

Спробуємо застосувати метод оптимізації з метою спуску до мінімуму. Перевіримо, що в залежності від початкової точки оптимізація буде застрягати у локальному мінімумі.

Застосуємо звичайний по координаті, яка зменшує час більше всього і запустимо його ітеративно.

На виході з симуляційної системи ми отримуємо квадратну матрицю для обох гравців з їх часами отримання усіх задач. Позначимо їх так само як і у теоретичному розділі за $A, B \in \mathbb{R}^{N \times N}$ як програші гравців 1 та 2 відповідно. Проте оскільки ми знаємо, що матриці при багаторазовому повторенні експерименту та усередненні результатів, такі, що $A = B^T$. То будемо розглядати оптимізацію часу лише одного гравця, наприклад першого.

Нехай задана початкова точка (a_0, b_0) як початкові стратегії користувачів. З цієї точки стартує алгоритм.

Етапи роботи алгоритма спуску:

1. Задаємо початкову точку $p_0 = (a_0, b_0)$, час у точці p_0 поставимо за $t_0 = A_{a_0 b_0}$. Поставимо $k=0$.
2. Шукаємо найменший час по квадрату зі сторонами в 3 елементи матриці і центром у точці p_k .
3. Вибираємо точку з найменшим часом та ставимо її як p_{k+1} . Якщо $p_{k+1} = p_k$, то зупиняємо алгоритм.
4. $k = k + 1$, повертаємося на крок 2.

Перевіримо як працює алгоритм. Для перевірки проведемо симуляцію для розміру матриць 10000 по розрізанням, які є дільниками 10000, нехтуємо розширенням симуляцій з блоками. Знайдемо глобальний мінімум та спробуємо запустити оптимізацію з різних стартових точок. Потужності обчислювальних вузлів поставимо у співвідношенні 2:2:1. Тобто три обчислювальних вузла.

Таблиця 3.3

№	Початкова стратегія	Початковий час	Кінцева стратегія	Кінцевий час
1.	(2,2)	6670.72055426	(1250, 2000)	11.0248749
2.	(10000,10000)	14.65571875	(5000,10000)	13.7184062
3.	(5,25)	1335.36618371	(1000,1250)	11.0248749
4.	(25,5)	1604.09384557	(1000,1250)	11.0248749
5.	(25,5)	1604.09384557	(1000,1250)	11.0248749
6.	(625,40)	184.069188072	(1000,1250)	11.0248749

Як бачимо з таблиці 3.3, оптимізація працює непогано в сенсі що дозволяє вибратись з неефективних точок. Проте оптимізація застрагяє і не дохо-

дить до глобального мінімуму. І як ми пам'ятаємо за Рис. 3.7, при розгорнутій симуляції з розбиттями, які не є дільниками розмірності матриці, то там канави зустрічаються дуже часто, що говорить про повну неможливість використовувати методи оптимізації для знаходження оптимальної точки, проте у випадку дільників розмірності матриці оптимізація допомагає вибратись з поганих точок до відносно хороших, які по часу дуже близько до глобального мінімуму.

Висновки до розділу

У цьому розділі були проведені експерименти за допомоги побудованої СПЗ. Ці експерименти дозволили перевірити основну модель на основі потокової та особливо характер штрафів.

За допомоги додаткової бібліотеки "nashpy" на мові Python було проведено пошук рівноваг за Нешем у грі двох користувачів та отримано результати.

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 математичні 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 великими 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 проекту є пошук найефективніших шляхів паралельного виконання дрібних задач та використання симуляційних систем для аналізу ефективності розбиття задачі на підзадачі.

Таблиця 4.1 – Опис ідеї стартап-проекту

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

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

№ п/п	Техніко- економічні характери- стики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (ней- тральна сторона)	S (сильна сторона)
		Мій про- ект	Cloud Sim	Grid Sim	Amazon AWS			
1	Швидка си- муляційна система	продукт на мові C++	бібліо- тека на Java	бібліотека на Java	повно- машта- бна система обчи- слень	потребує ретельного аналізу cloud систем	доступна платфор- ма для наукових дослід- жень плану- вальни- ків	дозволяє оцінити ефе- ктивність cloud системи
2	Ефективна аналітика паралельних алгоритмів	додаткові функції для по- шуку опти- мальних конфі- гурацій алгори- тма	-	-	розроб- ка вла- сного пакету аналіти- ки	потребує зна- чних витрат у дослідження	-	дозволяє значно прискорити об- числення, що у майбутньо- му дозволить економити на обчисленнях у cloud системах

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її ре- алізації	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

4.2 Аналіз ринкових можливостей запуску стартап-проекту

Таблиця 4.4 – Попередня характеристика потенційного ринку 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 4.5 – Характеристика потенційних клієнтів 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 4.6 – Фактори загроз

№ 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 4.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Співробітництво з платформами хмарних обчислень	Можливість просування продукту напряду на платформах завдяки угоді з компаніями провайдерів 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 4.8 – Ступеневий аналіз конкуренції на ринку

№ п/п	Особливості конкуретного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства
1	Олігополія	Існують відриті системи - CloudSim та GridSim	Задають стандарт програмного забезпечення для симуляцій хмарних обчислень
2	Міжнародний рівень конкуретної боротьби	Цифрові продукту інформаційного пошуку не мають кордонів	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 конкуретних перваг: нецінова	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 4.9 – Аналіз конкуренції за М. Портером (Висновки)

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

Таблиця 4.10 – 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 4.11 – Обґрунтування факторів конкурентноспроможності

№ п/п	Фактор конкурентноспроможності	Обґрунтування (чинники, що роблять фактор для порівняння 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 великих 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 4.12 – Порівняльний аналіз сильних та слабких сторін

#	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 4.13 – SWOT- аналіз стартап-проекту

<p>Сильні сторони:</p> <ul style="list-style-type: none"> • Висока швидкодія у порівнянні з аналогічними продуктами • Вбудовані прості планувальники, які легко використовувати • Простота бібліотеки 	<p>Слабкі сторони:</p> <ul style="list-style-type: none"> • Недостатня універсальність оскільки продукт лише у початковому виді • Продукт постачається мовою C++, яка вважається складнішою за Java для людей, які більше математики ніж програмісти
<p>Можливості:</p> <ul style="list-style-type: none"> • 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 	<p>Загрози:</p> <ul style="list-style-type: none"> • Компанії гіганти можуть випустити власне програмне забезпечення для попередньої симуляцій • Існуючі продукти у розробці також можуть активізуватися та спробувати конкурувати на ринці • Поява нових конкурентів з порівняно схожою швидкістю програмного продукту

Таблиця 4.14 – Альтернативи ринкового впровадження 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

4.3 Розробка ринкової стратегії проекту

Таблиця 4.15 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйти продукт	Орієнтовний попит 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 будуть 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 інтуїтивною 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 4.16 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія розвитку ринку	Ключові конкурентно-спроможні 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 4.17 – Визначення базової стратегії конкурентної поведінки

№ 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 характеристики товару конкурента 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 зручній формі та із значно кращою швидкодією

Таблиця 4.18 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувану 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

4.4 Розроблення маркетингової програми стартап-проекту

Таблиця 4.19 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1	Можливість провести експерименти з різними структурами cloud систем перед власне замовленням 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
3	Дослідницька діяльність у сфері хмарних обчислень	СПЗ надає можливість практично перевіряти гіпотези чи проводити дослідження емпіричним шляхом	ПП конкурентів непридатний до дослідження високонавантажених сценаріїв

Таблиця 4.20 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Задовольняє потребу у швидкому СПЗ, яке дозволить ефективно проводити симуляції складних сценаріїв навіть на простих портативних комп'ютерах		
II. Товар у реальному виконанні	Властивості / характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Висока швидкість 2. Простота поширення 3. Низька вартість обслуговування		
	Якість: продукт надає високу швидкодію симуляцій навіть на непотужних комп'ютерах		
	Пакування: продукт розповсюджується дистрибутивно		
	Марка: HighLoadComp, назва товару - HighLoadSim		
III. Товар із підкріпленням	До продажу:	базова симуляційна система + підтримка клієнтів	
	Після продажу:	СПЗ з додатковими послугами розробки ефективного паралельного алгоритма	
За рахунок чого потенційний товар буде захищено від копіювання: даний продукт легко скопіювати, проте оскільки захист коду можливий, то значить для копіювання потрібно буде розробляти СПЗ з нуля, що тягне за собою великі затрати на розробку			

Таблиця 4.21 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	Товари замінники - ХС, рівень цін на які достатньо високий	Товари аналоги безкоштовні, проте вони не підходять для симуляції високонавантажених сценаріїв	Рівень доходів студентів та викладачів низький	Верхня межа 3 \$, верхня 5\$ за підписку

Таблиця 4.22 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Велика кількість поодиноких клієнтів користується сервісом та з часом кількість користувачів збільшується (оскільки заощаджують обидві сторони)	(не застосовується)	1	Збут через онлайн-системи чи співпрацю з університетами

Таблиця 4.23 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Поступово вводиться сервісний збір за користування чисто онлайн системою	Внутрішня система комунікацій через сервісну систему	Заощадження на дослідженнях такого складного середовища як ХС	Поширити інформацію про СПЗ, яке надає значно більшу швидкодію та просте у використанні	Рекламу краще проводити через університети чи дослідницькі центри, які проводять дослідження у сфері хмарних обчислень

Висновки до розділу

Дана магістерська дисертація має перспективи зростання до популярного продукту у сфері хмарних обчислень. Існуючі бібліотеки хоч і доступні безплатно, проте мають дуже низьку швидкодію, яка взагалі не дозволяє проводити повномаштабні експерименти з метою оцінки її майбутньої продуктивності.

Розробка продукту не потребує значних вкладів, оскільки для написання проекту достатньо і трьох досвідчених програмістів. Після завершення основної фази розробки та початку реклами проекту. Спочатку краще всього зробити деяку версію продукту для університетів та студентів, оскільки саме для них в першу чергу може бути корисним цей продукт.

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 ІТ сфері.

Проте часто великим компаніям дешевше купити невелику компанію чи стартап, та інтегрувати їх продукт у свою інфраструктуру, якщо якість продукту на достаньому рівні та задовольняє основні потреби користувачів. Ми вважаємо, що такий сценарій буде найкращим із усіх можливих, оскільки значно полегшить рекламу та дистрибуцію СПЗ.

ВИСНОВКИ ПО РОБОТІ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У роботі проведено аналіз задачі множення матриць методом побудови потокової моделі для оцінки часу виконання задач одного користувача, а далі узагальнено для двох користувачів. Також чітко показано, що складність обчислень не збільшується в залежності від кількості ОВ, їх потужностей та інших параметрів ХС. Показано як впливає розмір розбиття на величину штрафів за пересилки. Також ця методологія може бути використана для аналізу інших задач, наприклад основних сценаріїв стандарту BLAS (Basic Linear Algebra Subprograms), які є основою усіх наукових обчислень навіть у великих проектах.

Розроблено СПЗ, яке дозволяє дуже швидко емулювати процес множення матриць з використанням одного із чотирьох статичних планувальників: `minmin`, `minmax`, `maxmin` та `maxmax`. Також це СПЗ універсальне та допускає розширення, удосконалення з можливістю переходу у реальний продукт, на базі якого можливо буде емпірично визначати параметри розбиття чи перевірки інших математичних моделей задач.

У подальшому також можна розглянути двох гравців як незалежні штучні інтелекти, які навчаються вибирати розрізання, та мають єдиний сигнал зворотнього зв'язку - час отримання усіх своїх задач. Таким чином можна дослідити деякі алгоритми навчання з підкріпленням, оскільки це добре підходить під концепцію гри де виграші на момент t якраз і будуть зворотнім сигналом. Основними алгоритмами у цій сфері є: Q-learning, SARSA, DQN, DDPG.

ПЕРЕЛІК ПОСИЛАНЬ

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 // Proceedings of the Third Annual ACM Symposium on Parallel Algorithms and Architectures. — SPAA '91. — New York, NY, USA: ACM, 1991. — Pp. 216–228. [Електронний ресурс]. — Режим доступу: <http://doi.acm.org/10.1145/113379.113399>.
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. Reisizadehmobarakeh, Amirhossein. Coded Computation over Heterogeneous Clusters.
9. Lee, Kang Wook. Coded computation for multicore setups. — 2017. — 06. [Електронний ресурс]. — Режим доступу: <https://arxiv.org/abs/1706.02670>.

[//www.researchgate.net/publication/319375348_Coded_computation_for_multicore_setups](http://www.researchgate.net/publication/319375348_Coded_computation_for_multicore_setups).

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. — № 1. — С. 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. Daskalakis, Constantinos. The Complexity of Computing a Nash Equilibrium / Constantinos Daskalakis, Paul Goldberg, Christos H. Papadimitriou // SIAM Journal on Computing. — 2009. — 02. — Vol. 39. — Pp. 195–259.
13. Algorithmic Game Theory / Noam Nisan, Tim Roughgarden, Eva Tardos, Vijay V. Vazirani. — New York, NY, USA: Cambridge University Press, 2007. — P. 775.
14. NashPy. [Електронний ресурс]. — Режим доступу: <https://nashpy.readthedocs.io/>.
15. Бланк, С. Стартап. Настольная книга основателя / С. Бланк, Б. Дорф. — 2 изд. — Москва : Альпина Паблишер, 2014. — С. 614.
16. Дрейпер, У. Стартапы : профессиональные игры Кремниевой долины / У. Дрейпер. — 2 изд. — Москва : Эксмо, 2012. — С. 378.
17. Тиль, П. От нуля к единице : как создать стартап, который изменит будущее / П. Тиль, Б. Мастерс. — Москва : Альпина паблишер, 2015. — С. 188.
18. Коэн, Д. Стартап в Сети : мастер-классы успешных предпринимателей / Д. Коэн, Фелд Б. — 2 изд. — Москва : Альпина паблишер, 2013. — С. 337.
19. Маллинс, Дж. Поиск бизнес-модели : как спасти стартап, вовремя сменив план / Дж. Маллинс, Комисар Р. — Москва : Маннр, 2012. — С. 329.
20. Харниш, В. Правила прибыльных стартапов : как расти и зарабатывать деньги / В. Харниш. — Москва : Манн, Иванов и Фербер, 2012. — С. 279.
21. Как продвигать проекты коммерциализации технологий : серия методических материалов «Практические руководства для центров коммерциализации технологий.

ДОДАТОК А. ІЛЮСТРАТИВНІ МАТЕРІАЛИ ДОПОВІДІ

Теоретико-ігровий аналіз планувальників у гетерогенному багатопроцесорному середовищі

студент 6-го курсу
КА-61м, Одобеску Владислав

Інститут прикладного системного аналізу
керівник: доц. Ігнатенко Олексій Петрович



1 / 7

студент 6-го курсу КА-61м, Одобеску Владислав

Теоретико-ігровий аналіз планувальників у гетерогенному баг

Актуальність роботи



2 / 7

студент 6-го курсу КА-61м, Одобеску Владислав

Теоретико-ігровий аналіз планувальників у гетерогенному баг

Постановка задачі



3/7

студент 6-го курсу КА-61м, Одобеску Владислав

Теоретико-ігровий аналіз планувальників у гетерогенному баг

Постановка задачі



4/7

студент 6-го курсу КА-61м, Одобеску Владислав

Теоретико-ігровий аналіз планувальників у гетерогенному баг

Висновки



5 / 7

студент 6-го курсу КА-61м, Одобеску Владислав

Теоретико-ігровий аналіз планувальників у гетерогенному баг

Шляхи подальшого розвитку



6 / 7

студент 6-го курсу КА-61м, Одобеску Владислав

Теоретико-ігровий аналіз планувальників у гетерогенному баг

Дякую за увагу.



ДОДАТОК Б. ЛІСТИНГ КОДУ

Код симулятора на мові C++

appendicies/cppcode/main.cpp

```

1  #include <list>
    #include <vector>
    #include <iostream>
    #include <iomanip>
    #include <fstream>
6  #include <functional>

    #include <boost/algorithm/string.hpp>
    #include <boost/program_options.hpp>
    #include <valarray>
11

    #include <smpp/mmsim.hpp>
    #include <smpp/smpp.hpp>
    #include <smpp/task.hpp>
    #include <smpp/processor.hpp>
16 #include <smpp/task_processor.hpp>

    namespace po = boost::program_options;

    template<typename It>
21 void write_to_stream(std::ostream& stream, It begin, It end, std::string
        separator = "--")
    {
        stream << *begin;
        ++begin;
        while (begin!=end)
26     {
            stream << separator << *begin;
            ++begin;
        }
    }
31

int main(int argc, char* argv[])
{
    typedef smpp::SimpleTask          task;
    typedef smpp::Processor           processor;
36    typedef smpp::TaskProcessorWithTransfer task_processor;

```

```

typedef std::function<std::vector<double>(const size_t, const
std::vector<smpp::Processor>& procs, std::vector<smpp::SimpleTask>&&, const
smpp::TaskProcessor&)> fsimulator;

try
41 {
    po::options_description desc("Allowed options");
    desc.add_options()
        ("help" , "produce help message")
        ("sim_log" , "log simulation data")
46 //general simulation params
        ("problem_size" , po::value<size_t>()->required()
            , "problem size"
        )

        ("nominal_mips" , po::value<double>()->default_value(1e10)
            , "nominal mips value"
        )

        ("mips" ,
po::value<std::vector<double>>()->required()->multitoken(), "cores mips
values as multiplication of nominal"
        )
        ("task_priority" ,
po::value<std::string>()->default_value("min") , "task scheduling
priority"
        )
51 ("proc_priority" ,
po::value<std::string>()->default_value("min") , "processor
choosing priority"
        )
        // task processor
        ("bandwidth" , po::value<double>()->default_value(1e8)
            , "bandwidth with each processing unit (one value for all)"
        )

        ("ping" , po::value<double>()->default_value(1e-5)
            , "ping with each processing unit (one value for all)"
        )

        // slice params
56 ("slices" ,
po::value<std::vector<size_t>>()->multitoken()->required(), "slice params
(min slice, max slice, step)"
        )
        ("fix_first" , po::value<size_t>()->default_value(0)
            , "fixed first player strategy"
        )

        ("randomize_count" , po::value<size_t>()->default_value(1)
            , "how many times to simulate with shuffling"
        )

        ("single_player" , po::value<bool>()->default_value(false)
            , "make single player simulation"
        )
    }

```

```

61         ("output"
po::value<std::string>()->default_value("results.txt") , "output_file"
        )
        ;

po::variables_map vm;
66 po::store(po::parse_command_line(argc, argv, desc), vm);

if(vm.count("help"))
{
    std::cout << desc << std::endl;
71     return 0;
}

po::notify(vm);

76 const size_t problem_size = vm["problem_size"].as<size_t>();
const double nominal_mips = vm["nominal_mips"].as<double>();
std::vector<double> mips = vm["mips"].as<std::vector<double>>();
std::vector<processor> procs;
procs.reserve(mips.size());
81 std::for_each(mips.begin(), mips.end(),
    [nominal_mips, &procs](double val)
    {
        procs.emplace_back(nominal_mips*val);
    });

86 auto task_priority = vm["task_priority"].as<std::string>();
std::transform(task_priority.begin(), task_priority.end(),
task_priority.begin(), ::tolower);
task::comparator task_comparator;
if(task_priority == "min")
91     task_comparator= task::small_first();
else if (task_priority == "max")
    task_comparator = task::large_first();
else
    throw
po::validation_error(po::validation_error::invalid_option_value,
"task_priority");

96 auto proc_priority = vm["proc_priority"].as<std::string>();
std::transform(proc_priority.begin(), proc_priority.end(),
proc_priority.begin(), ::tolower);
processor::comparator proc_comparator;
if (proc_priority == "min")
101     proc_comparator = processor::slow_first();

```



```

else if (proc_priority == "max")
    proc_comparator = processor::fast_first();
else
    throw
po::validation_error(po::validation_error::invalid_option_value,
"proc_priority");
106

    const double bandwidth = vm["bandwidth"].as<double>();
    const double ping = vm["ping"].as<double>();
    std::unique_ptr<task_processor> tp =
std::make_unique<task_processor>(bandwidth, ping);

111    const std::vector<size_t> slices_arr =
vm["slices"].as<std::vector<size_t>>();
    std::vector<size_t> slices;
    if (slices_arr.size() == 3)
    {
        const size_t slice_start = slices_arr[0];
116        const size_t slice_end = slices_arr[1];
        const size_t slice_step = slices_arr[2];
        slices.reserve((slice_end - slice_start + 1) / slice_step);
        for (size_t i = slice_start; i < slice_end; i += slice_step)
            slices.push_back(i);
121    }
    else
    {
        slices = slices_arr;
    }

126

    const size_t fix_first = vm["fix_first"].as<size_t>();

    const size_t randomize_count = vm["randomize_count"].as<size_t>();
    const bool do_shuffle = randomize_count != 0;
131    const bool single_player = vm["single_player"].as<bool>();

    const bool sim_log = vm.count("sim_log") > 0;
    std::ofstream sim_log_file;

136    std::string fname = vm["output"].as<std::string>();
    if (fname == "auto")
    {
        std::stringstream ss;
        ss << "sim_" << proc_priority << "_" << task_priority;
141        ss << "_n_" << problem_size;
        ss << "_m_" << std::scientific << std::setprecision(3) <<
nominal_mips;

```

```

        ss << "_bw_" << std::scientific << std::setprecision(3) <<
bandwidth;
        ss << "_p_" << std::scientific << std::setprecision(3) << ping;
        ss << "_rd_" << randomize_count;
146         if (single_player)
            ss << "single";
        ss << ".txt";
        fname = ss.str();
    }
151     if (sim_log)
    {
        sim_log_file = std::ofstream(fname + ".log");
        sim_log_file.precision(20);
    }

156     std::ofstream file(fname);
    file.precision(20);
    if (!file.is_open())
        throw std::runtime_error("couldn't open file");

161     file << "ProblemSize=" << problem_size << "|||NominalMips=" <<
nominal_mips << "|||Bandwidth=" << bandwidth << "|||Ping=" << ping <<
std::endl;
    file << "Slices=";
    write_to_stream(file, slices.begin(), slices.end());
    file << std::endl;
166     file << "MipsMultipliers=";
    write_to_stream(file, mips.begin(), mips.end());
    file << std::endl;

171     if(single_player)
    {
        file << "Slice,Time" << std::endl;
        for (const auto& i : slices)
        {
176             file << i << ',';
            auto tasks = smpp::mmsim::create_tasks<task>(problem_size, { i});
            auto result = simulate(procs, proc_comparator, tasks,
task_comparator, *tp, 1, false, sim_log);
            if(sim_log)
            {
181                 sim_log_file << "Log for slice=" << i << std::endl;
                std::for_each(result.second.begin(),
result.second.end(), [&sim_log_file](auto& val)
                {
                    sim_log_file << val << std::endl;

```

```

        });
186     }
        file << result.first[0];
        file << std::endl;
        file.flush();
    }
191 }
    else
    {
        file << "Slice_First,Slice_Second,Time_First,Time_Second" <<
std::endl;
        const std::vector<size_t> v{ fix_first };
196 const std::vector<size_t>& f_s = fix_first == 0 ? slices : v;
        for (const auto& i : f_s)
            for (const auto& j : slices)
            {
                file << i << ',' << j << ',';

201

                std::valarray<double> times_array;
                task_processor::return_type processed_tasks;
                auto tasks_main =
smpp::mmsim::create_tasks<task>(problem_size, { i, j });
                std::tie(times_array, processed_tasks) = simulate(procs,
proc_comparator, tasks_main, task_comparator, *tp, 2, do_shuffle, sim_log);
206 if (sim_log)
                {
                    sim_log_file << "Log_for_slice1=" << i << "|slice2=" << j
<< std::endl;

                    sim_log_file.flush();
                    std::for_each(processed_tasks.begin(),
processed_tasks.end(), [&sim_log_file](auto& val)
211 {
                        sim_log_file << val << std::endl;
                    });
                    sim_log_file.flush();
                }
216 for (size_t times = 1; times < randomize_count; ++times)
                {
                    auto tasks =
smpp::mmsim::create_tasks<task>(problem_size, { i, j });
                    times_array += simulate(procs, proc_comparator, tasks,
task_comparator, *tp, 2, do_shuffle).first;
                }
221 if (randomize_count != 0)
                    times_array /= randomize_count;
                    file << times_array[0] << ',' << times_array[1];
                    file << std::endl;

```

```

        file.flush();
226     }
    }
    catch(const std::exception& ex)
    {
231         std::cerr << "Exception_" << ex.what() << std::endl;
    }
    catch (...) {
        std::cerr << "Exception_of_unknown_type" << std::endl;
    }
236     return 0;
}

```

appendicies/cppcode/smpp/mmsim.hpp

```

#pragma once
#include<list>
3 #include<vector>
#include<tuple>

namespace smpp
8 {
    namespace mmsim
    {
        inline size_t calculate_number_of_tasks(const size_t problem_size, const
size_t slice_size)
        {
13             const auto n_full = problem_size / slice_size;
            const auto partial = (problem_size % slice_size) != 0;
            if (partial)
                return n_full * (n_full + 2) + 1;
            return n_full * n_full;
18         }

        inline size_t calculate_number_of_tasks(const size_t problem_size, const
std::list<size_t>& slice_sizes)
        {
            size_t accumulated_size = 0;
23             for (auto& slice_size : slice_sizes)
                accumulated_size += calculate_number_of_tasks(problem_size,
slice_size);
            return accumulated_size;
        }
    }
}

```

```

28     inline auto calculate_complexity(const size_t m, const size_t n, const
size_t k, double multiplication_to_addition = 1.0)
    {
        return std::make_pair(m * k*(n*multiplication_to_addition + n - 1),
m*n + n*k + m*k);
    }

33     inline auto calculate_complexities(const size_t problem_size, const
size_t slice_size)
    {
        typedef decltype(calculate_complexity(size_t(), size_t(), size_t(),
double())) compl_t;
        const auto n_full = problem_size / slice_size;
        const auto partial = problem_size - n_full * slice_size;

38         auto p1 = calculate_complexity(slice_size, problem_size, slice_size);
        if (partial == 0)
        {
            return std::make_tuple(n_full, p1, compl_t(0.0, 0), compl_t(0.0,
0));
43         }
        auto p2 = calculate_complexity(partial, problem_size, slice_size);
        auto p3 = calculate_complexity(partial, problem_size, partial);

        return std::make_tuple(n_full, p1, p2, p3);

48     }

    /*
    * Task should have static create method (double complexity, size_t
n_numbers, size_t userid)
    */
53     template<typename Task>
        std::vector<Task> create_tasks(const size_t problem_size, const
std::list<size_t>& slice_sizes)
        {
            typedef decltype(calculate_complexity(size_t(), size_t(), size_t(),
double())) compl_t;

58             const auto n_tasks = calculate_number_of_tasks(problem_size,
slice_sizes);
            std::vector<Task> tasks;
            tasks.reserve(n_tasks);
            typename Task::userid_type user_id = 0;
            for (auto& slice_size : slice_sizes)
63             {
                size_t n;
                compl_t p1, p2, p3;

```

```

        std::tie(n, p1, p2, p3) = calculate_complexities(problem_size,
slice_size);
        tasks.insert(tasks.end(), n*n, Task::create(p1.first, p1.second,
user_id));
68         if (p2.first != 0.0)
            {
                tasks.insert(tasks.end(), 2 * n, Task::create(p2.first,
p2.second, user_id));
                tasks.insert(tasks.end(), 1, Task::create(p3.first,
p3.second, user_id));
            }
73         ++user_id;
    }

    return tasks;
}
78 }
}

```

appendicies/cppcode/smpp/priority_queue.hpp

```

1 #pragma once

#include <vector>
#include <algorithm>

6 namespace smpp
{
    template <typename Type, typename Compare = std::less<Type>>
    class priority_queue
    {
11     public:
        typedef Type value_type;
        typedef std::vector<Type> container_type;
        typedef Compare value_compare;
        typedef typename container_type::size_type size_type;
16        typedef typename container_type::reference reference;
        typedef typename container_type::const_reference const_reference;

        explicit priority_queue(const Compare& compare = Compare())
            : comparator{ compare }
21        { }

        void push(value_type element)
        {
            container.push_back(std::move(element));
            std::push_heap(container.begin(), container.end(), comparator);
26        }
    }
}

```

```

template<typename... Args>
void emplace(Args... args)
{
31     container.push_back(value_type(std::forward<Args>(args)...));
        std::push_heap(container.begin(), container.end(), comparator);
}

value_type pop()
36 {
    std::pop_heap(container.begin(), container.end(), comparator);
    value_type result = std::move(container.back());
    container.pop_back();
    return std::move(result);
41 }

bool empty() const
{
    return container.empty();
46 }

const_reference top() const
{
    return container.front();
51 }

private:
    container_type container;
    value_compare comparator;
56 };
}

```

appendices/cppcode/smpp/processor.hpp

```

#pragma once

3 #include <functional>

namespace smpp
{
    struct Processor
8    {
        typedef std::function<bool(const Processor&, const Processor&)>
        comparator;

        explicit Processor(const double mips)
            : mips(mips)
    }
}

```

```

13         {

        }

        Processor(const Processor&) = default;
18        Processor(Processor&&) = default;
        Processor& operator=(Processor&&) = default;
        Processor& operator=(const Processor&) = default;
        ~Processor() = default;

23        double time_to_complete(const double complexity) const
        {
            return complexity / mips;
        }

28        static comparator slow_first()
        {
            return [](const Processor& l, const Processor& r) -> bool { return
l.mips < r.mips; };
        }

33        static comparator fast_first()
        {
            return [](const Processor& l, const Processor& r) -> bool { return
l.mips > r.mips; };
        }

38        double mips;
    };
}

namespace std
43 {
    template<>
    struct less<smpp::Processor>
    {
        typedef smpp::Processor val;
48        constexpr bool operator()(const val& l, const val& r) const
        {
            return l.mips < r.mips;
        }

53    };

    template<>
    struct greater<smpp::Processor>
    {

```



```

58     typedef smpp::Processor val;
        constexpr bool operator()(const val& l, const val& r) const
        {
            return l.mips > r.mips;
        }
63
    };
}

```

appendicies/cppcode/smpp/smpp.hpp

```

#pragma once

#include <vector>
#include <valarray>
5  #include <list>
#include <functional>
#include <random>
#include <set>

10 #include <smpp/processor.hpp>
#include <smpp/task.hpp>
#include <smpp/priority_queue.hpp>
#include <smpp/task_competition.hpp>
#include <smpp/task_processor.hpp>
15
namespace smpp
{
    auto simulate(
        std::vector<Processor> procs, Processor::comparator proc_comp,
20     std::vector<SimpleTask>& tasks, SimpleTask::comparator task_comp,
        const TaskProcessor& tprocessor,
        const SimpleTask::userid_type n_user_hint,
        const bool shuffle = true,
        const bool return_processed = false
25     )
    {
        auto& tasks_to_process = tasks;

        if (shuffle)
30     {
            std::random_device rd;
            std::mt19937 g(rd());
            std::shuffle(tasks_to_process.begin(), tasks_to_process.end(), g);
        }
35     // sort tasks
        std::sort(tasks_to_process.begin(), tasks_to_process.end(), task_comp);
    }
}

```

```

// sort processors
std::sort(procs.begin(), procs.end(), proc_comp);

40 auto processed_tasks = tprocessor(procs, tasks_to_process);

std::set<SimpleTask::userid_type> idx;
std::valarray<double> times(n_user_hint);

45 //size_t previous_idx = std::numeric_limits<size_t>::max();
for (auto curr = processed_tasks.crbegin(); curr !=
processed_tasks.crend(); ++curr)
{
    auto curr_idx = curr->task->userid;
    if (idx.insert(curr_idx).second)
50 {
        times[curr_idx] = curr->time_end;
        if (idx.size() == n_user_hint)
            break;
    }
55 }

return std::make_pair(std::move(times), return_processed ?
std::move(processed_tasks) : TaskProcessor::return_type());
}
}

```

appendicies/cppcode/smpp/task.hpp

```

1 #pragma once

#include <functional>

namespace smpp
6 {
    struct SimpleTask
    {
        typedef uint8_t userid_type;

11     typedef std::function<bool(const SimpleTask&, const SimpleTask&)>
        comparator;

        static SimpleTask create(const double complexity, const size_t n_numbers,
        const userid_type userid)
        {
            return SimpleTask(complexity, 8*sizeof(double)*n_numbers, userid);
16     }
}

```

```

        explicit SimpleTask(const double complexity, const size_t
bits_to_transfer, const userid_type userid)
            : complexity(complexity), bits_to_transfer(bits_to_transfer),
userid(userid)
        {
21
        }

        static comparator small_first()
        {
26
            return [](const SimpleTask& l, const SimpleTask& r) -> bool { return
l.complexity < r.complexity; };
        }

        static comparator large_first()
        {
31
            return [](const SimpleTask& l, const SimpleTask& r) -> bool { return
l.complexity > r.complexity; };
        }

        SimpleTask(const SimpleTask&) = default;
        SimpleTask(SimpleTask&&) = default;
36
        SimpleTask& operator=(SimpleTask&&) = default;
        SimpleTask& operator=(const SimpleTask&) = default;

        friend std::ostream& operator<< (std::ostream& stream, const SimpleTask&
task)
        {
41
            stream << task.complexity << ',' << task.bits_to_transfer << ',' <<
int(task.userid);
            return stream;
        }

        double    complexity;
46
        size_t    bits_to_transfer = 0;
        userid_type userid        = 0;
    };
}

51 namespace std
{
    template<>
    struct less<smpp::SimpleTask>
    {
56
        typedef smpp::SimpleTask val;
        constexpr bool operator()(const val& l, const val& r) const
        {

```

```

        return l.complexity < r.complexity;
    }

61     };

    template<>
    struct greater<smpp::SimpleTask>
66     {
        typedef smpp::SimpleTask val;
        constexpr bool operator()(const val& l, const val& r) const
        {
            return l.complexity > r.complexity;
71         }

        };
    }

```

appendicies/cppcode/smpp/task_competition.hpp

```

1  #pragma once

    namespace smpp
    {
        template<typename T>
6         T& get_ref(T& t)
        {
            return t;
        }

        template<typename T>
11        T& get_ref(T* t)
        {
            return *t;
        }

16        template<typename Task>
        struct task_competition
        {
            task_competition(
21                const double time_start,
                const double time_end,
                const size_t worker_index,
                Task&& task
            )
26                : time_start(time_start), time_end(time_end),
                worker_index(worker_index), task(std::move(task))
            {

```

```

    }

31     task_completion(task_completion&&) = default;
    task_completion& operator=(task_completion&&) = default;

    friend std::ostream& operator<< (std::ostream& stream, const
task_completion& tc)
36     {
        stream << tc.time_start << ',' << tc.time_end << ',' <<
tc.worker_index << ',' << get_ref(tc.task);
        return stream;
    }

41

    struct later_first
    {
        constexpr bool operator()(const task_completion& l, const
task_completion& r)
        {
46             return l.time_end > r.time_end;
        }
    };

51

    // members
    double time_start;
    double time_end;
    size_t worker_index;
56    Task task;
};

}

```

appendicies/cppcode/smpp/task_processor.hpp

```

#pragma once

#include <functional>

5 #include <smpp/processor.hpp>
#include <smpp/task.hpp>
#include <smpp/priority_queue.hpp>
#include <smpp/task_completion.hpp>

```

```

10

namespace smpp
{
    struct TaskProcessor
15    {
        typedef SimpleTask task;
        typedef task* task_ptr;
        typedef priority_queue<task_completion<task_ptr>, typename
task_completion<task_ptr>::later_first> task_queue;
        typedef std::vector<task_completion<task_ptr>> return_type;

20

        virtual return_type operator()(const std::vector<Processor>& procs,
std::vector<task>& tasks) const = 0;
    };

25    struct TaskProcessorWithTransfer : TaskProcessor
    {
        typedef TaskProcessor::task task;
        typedef TaskProcessor::task_ptr task_ptr;
        typedef TaskProcessor::task_queue task_queue;
30        typedef TaskProcessor::return_type return_type;

        TaskProcessorWithTransfer(
            double bandwidth = 1e8, // ~100 mbit/s
            double connection_setup = 0.00001 // time to setup connection
35        )
            : bandwidth(bandwidth), connection_setup(connection_setup)
        {
        }

40        double transfer_time(size_t bits_to_transfer) const
        {
            return bits_to_transfer / bandwidth + connection_setup;
        }

45        return_type operator()(const std::vector<Processor>& procs,
std::vector<task>& tasks) const override
        {
            task_queue p_queue;
            return_type processed_tasks;
50            processed_tasks.reserve(tasks.size());

            auto task_iterator = tasks.begin();

```

```

        for (size_t i = 0; i < procs.size() && task_iterator != tasks.end();
++i)
        {
55             const auto time_to_process =
                procs[i].time_to_complete(task_iterator->complexity) //main
processing time
                +
                transfer_time(task_iterator->bits_to_transfer) // time
for data transfer
                ;
60             p_queue.emplace(0.0, time_to_process, i, &(*task_iterator));
                ++task_iterator;
        }

        while (!p_queue.empty())
65         {
            auto tk = p_queue.pop();
            if (task_iterator != tasks.end())
            {
                const auto time_to_process =
70
procs[tk.worker_index].time_to_complete(task_iterator->complexity) //main
processing time
                +
                transfer_time(task_iterator->bits_to_transfer)
// time for data transfer
                ;
                p_queue.emplace(tk.time_end, tk.time_end + time_to_process,
tk.worker_index, &(*task_iterator));
75                 ++task_iterator;
            }
            processed_tasks.push_back(std::move(tk));
        }

80         return processed_tasks;
    }

    double bandwidth;
    double connection_setup;
85 };
}

```

Код симулятора на мові Java з використанням пакета CloudSim Plus

../code/cloud_simulations/src/main/java/com/simulations/matrix/BasicScheduler.java

```
package com.simulations.matrix;

import org.cloudbus.cloudsim.cloudlets.Cloudlet;
4 import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.CloudSimTags;
import org.cloudbus.cloudsim.core.events.SimEvent;
import org.cloudbus.cloudsim.datacenters.Datacenter;
import org.cloudbus.cloudsim.brokers.DatacenterBrokerAbstract;
9 import org.cloudbus.cloudsim.vms.Vm;

import java.util.*;
import java.util.function.Function;

14 public class BasicScheduler extends DatacenterBrokerAbstract {
    private final HashSet<Cloudlet> submittedCloudlets = new HashSet<Cloudlet>();
    private final List<Cloudlet> finishedCloudlets = new ArrayList<>();
    private int runningCloudlets = 0;

19    private final CloudSim sim;

    BasicScheduler(CloudSim simulation) {
        super(simulation);
        sim = simulation;

24        setDatacenterSupplier(this::selectDatacenterForWaitingVms);

        setFallbackDatacenterSupplier(this::selectFallbackDatacenterForWaitingVms);
    }

29    protected Datacenter selectDatacenterForWaitingVms() {
        return (getDatacenterList().isEmpty() ? Datacenter.NULL :
        getDatacenterList().get(0));
    }

    protected Datacenter selectFallbackDatacenterForWaitingVms() {
34        return getDatacenterList().stream()
            .filter(dc -> !getDatacenterRequestedList().contains(dc))
            .findFirst()
            .orElse(Datacenter.NULL);
    }
}
```



```

    }

39
    @Override
    public Set<Cloudlet> getCloudletCreatedList() {
        return submittedCloudlets;
    }

44
    @Override
    public <T extends Cloudlet> List<T> getCloudletFinishedList() {
        return (List<T>) new ArrayList<>(finishedCloudlets);
    }

49
    @Override
    protected void processCloudletReturn(SimEvent ev) {
        final Cloudlet c = (Cloudlet) ev.getData();
        finishedCloudlets.add(c);
54        println(String.format("%.2f: %s: %s %d finished and returned to broker.",
                                getSimulation().clock(), getName(), c.getClass().getSimpleName(),
                                c.getId()));
        —runningCloudlets;

        if(!getCloudletWaitingList().isEmpty()) {
59            sendOneCloudletToVm(c.getVm());
        }

        if (runningCloudlets > 0) {
            return;
64        }

        final Function<Vm, Double> func =
getVmDestructionDelayFunction().apply(c.getVm()) < 0 ? vm → 0.0 :
getVmDestructionDelayFunction();
        //If gets here, all running cloudlets have finished and returned to the
broker.
        if (getCloudletWaitingList().isEmpty()) {
69            println(String.format(
                                "%.2f: %s: All submitted Cloudlets finished executing.",
                                getSimulation().clock(), getName()));
            requestIdleVmsDestruction(func);
            return;
74        }

        /*There are some cloudlets waiting their VMs to be created.
        Then, destroys finished VMs and requests creation of waiting ones.
        When there is waiting Cloudlets, it always request the destruction
79        of idle VMs to possibly free resources to start waiting
        VMs. This way, the a VM destruction delay function is not set,

```

```

        defines one which always return 0 to indicate
        that in this situation, idle VMs must be destroyed immediately.
        */
84     requestIdleVmsDestruction(func);
        requestDatacenterToCreateWaitingVms();
    }

    protected boolean sendOneCloudletToVm(Vm vm) {
89         List<Cloudlet> waiting = getCloudletWaitingList();
        if (waiting.isEmpty())
            return false;
        Cloudlet cloudlet = waiting.remove(0);
        cloudlet.setVm(vm);
94         send(getVmDatacenter(vm).getId(), 0.0, CloudSimTags.CLOUDLET_SUBMIT_ACK,
cloudlet);
        ++runningCloudlets;

        return true;
    }

99     @Override
    protected void requestDatacentersToCreateWaitingCloudlets()
    {
        for (Vm vm : getVmExecList())
104         {
            sendOneCloudletToVm(vm);
        }
    }
}

```

../code/cloud_simulations/src/main/java/com/simulations/matrix/CloudletsTableBuilderE

```

package com.simulations.matrix;

2
import org.cloudbus.cloudsim.cloudlets.Cloudlet;
import org.cloudsimplus.builders.tables.CloudletsTableBuilder;
import org.cloudsimplus.builders.tables.TableBuilder;

7 import java.util.List;

public class CloudletsTableBuilderExtended extends CloudletsTableBuilder
{
    public CloudletsTableBuilderExtended(final List<? extends Cloudlet> list){
12         super(list);

        TableBuilder table = super.getTable();
    }
}

```

```

        super.addColumn(table.addColumn("User", "ID"),
17         cloudlet -> ((CloudletUser)cloudlet).getUser_id()
        );
    }
}

```

../code/cloud_simulations/src/main/java/com/simulations/matrix/CloudletUser.java

```

package com.simulations.matrix;

import org.cloudbus.cloudsim.cloudlets.CloudletSimple;

5 public class CloudletUser extends CloudletSimple
{
    private final int user_id;

    CloudletUser(final int userid, final long cloudletLength, final int pesNumber)
10    {
        super(cloudletLength, pesNumber);
        user_id = userid;
    }

15    CloudletUser(final int userid, final int id, final long cloudletLength,
    final long pesNumber)
    {
        super(id, cloudletLength, pesNumber);
        user_id = userid;
20    }

    public int getUser_id()
    {
25        return user_id;
    }
}

```

../code/cloud_simulations/src/main/java/com/simulations/matrix/DataCenterSimpleFixed

```

package com.simulations.matrix;

3 import org.cloudbus.cloudsim.allocationpolicies.VmAllocationPolicy;
import org.cloudbus.cloudsim.cloudlets.Cloudlet;
import org.cloudbus.cloudsim.cloudlets.CloudletExecution;
import org.cloudbus.cloudsim.core.CloudSimTags;
import org.cloudbus.cloudsim.core.Simulation;
8 import org.cloudbus.cloudsim.datacenters.DatacenterCharacteristics;
import org.cloudbus.cloudsim.datacenters.DatacenterSimple;

```

```

import org.cloudbus.cloudsim.hosts.Host;
import org.cloudbus.cloudsim.schedulers.cloudlet.CloudletScheduler;
import org.cloudbus.cloudsim.vms.Vm;

13
import java.util.List;
import java.util.stream.Collectors;

public class DataCenterSimpleFixed extends DatacenterSimple{
18
    private double UpdatePeriodThreshold = 0.001;
    private static double UpdatePeriodThresholdAddition = 0.0001;
    private boolean can_schedule_after_period = true;

23
    DataCenterSimpleFixed(Simulation simulation,
                          DatacenterCharacteristics characteristics,
                          VmAllocationPolicy vmAllocationPolicy)
    {
        super(simulation, characteristics, vmAllocationPolicy);
28
    }

    public void setUpdatePeriodThreshold(double value)
    {
        UpdatePeriodThreshold = value;
33
    }

    public double getUpdatePeriodThreshold()
    {
        return UpdatePeriodThreshold;
38
    }

    @Override
    protected boolean isTimeToUpdateCloudletsProcessing()
    {
43
        return getSimulation().clock() < 0.111 || getSimulation().clock() >=
        getLastProcessTime() + UpdatePeriodThreshold;
    }

    @Override
    protected double updateCloudletProcessing() {
48
        if (!isTimeToUpdateCloudletsProcessing())
        {
            // Schedule VM update processing
            // This fixes a bug when many ( bug is visible when all finished )
            before UpdatePeriodThreshold happens
            if(can_schedule_after_period)
53
            {

```

```

        schedule(getId(), UpdatePeriodThreshold +
UpdatePeriodThresholdAddition,
CloudSimTags.VM_UPDATE_CLOUDLET_PROCESSING_EVENT);
        can_schedule_after_period = false;
    }
    return Double.MAX_VALUE;
58     }
    can_schedule_after_period = true;

    double nextSimulationTime = updateHostsProcessing();
    if (nextSimulationTime != Double.MAX_VALUE) {
63         nextSimulationTime =
getCloudletProcessingUpdateInterval(nextSimulationTime);
        schedule(getId(),
            nextSimulationTime,
            CloudSimTags.VM_UPDATE_CLOUDLET_PROCESSING_EVENT);
    }
68     setLastProcessTime(getSimulation().clock());
    return nextSimulationTime;
}

@Override
73     protected double updateHostsProcessing() {
        double nextSimulationTime = Double.MAX_VALUE;
        for (final Host host : getHostList()) {
            final double time = host.updateProcessing(getSimulation().clock());
            nextSimulationTime = Math.min(time, nextSimulationTime);
78         }

        // Guarantees a minimal interval before scheduling the event
        final double minTimeBetweenEvents = UpdatePeriodThreshold +
UpdatePeriodThresholdAddition;
        nextSimulationTime = Math.max(nextSimulationTime, minTimeBetweenEvents);
83

        if (nextSimulationTime == Double.MAX_VALUE) {
            return nextSimulationTime;
        }

88         return nextSimulationTime;
    }

@Override
    public void checkCloudletsCompletionForGivenVm(Vm vm) {
93         CloudletScheduler cl_scheduler = vm.getCloudletScheduler();
        List<CloudletExecution> finishedList =
cl_scheduler.getCloudletFinishedList();
        if(finishedList.size() > 0) {

```

```

        Cloudlet cl = finishedList.get(finishedList.size() - 1).getCloudlet();
        if(!cl_scheduler.isCloudletReturned(cl))
98         {
            this.sendNow(cl.getBroker().getId(), 20, cl);
            cl_scheduler.addCloudletToReturnedList(cl);
        }
    }
103 }
}

```

../code/cloud_simulations/src/main/java/com/simulations/matrix/MatrixMultiplicationSim

```

1 package com.simulations.matrix;

import org.apache.commons.cli.*;
import org.cloudbus.cloudsim.schedulers.vm.VmSchedulerSpaceShared;
import org.cloudbus.cloudsim.util.Log;
6 import org.cloudbus.cloudsim.allocationpolicies.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.brokers.DatacenterBroker;
import org.cloudbus.cloudsim.cloudlets.Cloudlet;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.Simulation;
11 import org.cloudbus.cloudsim.datacenters.Datacenter;
import org.cloudbus.cloudsim.datacenters.DatacenterCharacteristics;
import org.cloudbus.cloudsim.datacenters.DatacenterCharacteristicsSimple;
import org.cloudbus.cloudsim.hosts.Host;
import org.cloudbus.cloudsim.hosts.HostSimple;
16 import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.ResourceProvisioner;
import org.cloudbus.cloudsim.provisioners.ResourceProvisionerSimple;
import org.cloudbus.cloudsim.resources.Pe;
import org.cloudbus.cloudsim.resources.PeSimple;
21 import org.cloudbus.cloudsim.schedulers.cloudlet.CloudletSchedulerSpaceShared;
import org.cloudbus.cloudsim.schedulers.vm.VmScheduler;
import org.cloudbus.cloudsim.utilizationmodels.UtilizationModel;
import org.cloudbus.cloudsim.utilizationmodels.UtilizationModelFull;
import org.cloudbus.cloudsim.vms.Vm;
26 import org.cloudbus.cloudsim.vms.VmSimple;

import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Constructor;
31 import java.lang.reflect.InvocationTargetException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.concurrent.ExecutorService;

```

```

36 import java.util.concurrent.TimeUnit;
    import java.util.function.Function;

    import static java.util.concurrent.Executors.newFixedThreadPool;

41
    // Theoretical values:
    // machine can do ~ 10000000000 additions per second
    // so make it a nominal value

46 public class MatrixMultiplicationSimulation {
    private static final Map<String, Class> scheduler_mapper = new HashMap<>();
    static
    {
        scheduler_mapper.put("minmin", SimpleSchedulers.MinMinScheduler.class);
51 scheduler_mapper.put("minmax", SimpleSchedulers.MinMaxScheduler.class);
        scheduler_mapper.put("maxmin", SimpleSchedulers.MaxMinScheduler.class);
        scheduler_mapper.put("maxmax", SimpleSchedulers.MaxMaxScheduler.class);
        scheduler_mapper.put("all_mt", null);
    }

56
    // Nominal MIPS (i7-6700K 1 core)
    private static final long NOMINAL_MIPS = 100000000000L;

    // Matrix multiplication complexity
61 private static final double multiplicationComplexityMultiplier = 1.;

    private static final int HOSTS = 1;
    private static final int HOST_PES = 100;
    private static final long HUGE_VALUE = 1000000000000000000L;
66 private static final long HOST_P_MIPS = NOMINAL_MIPS*1000;

    //members
    private final long problem_size;
    private final long slice1;
71 private final long slice2;

    private static class RunResult
    {
        private final double time_first;
76 private final double time_second;

        RunResult(double first, double second)
        {
            time_first = first;
81 time_second = second;
        }
    }

```

```

        double getFirst()
        {
86             return time_first;
        }

        double getSecond()
        {
91             return time_second;
        }
    }

    public MatrixMultiplicationSimulation(long n, long slice_size1, long
slice_size2) {
96         problem_size = n;
        slice1 = slice_size1;
        slice2 = slice_size2;
    }

101    long problemSize()
    {
        return problem_size;
    }

106    long getSlise1()
    {
        return slice1;
    }

111    long getSlise2()
    {
        return slice2;
    }

116    // Calculate complexity of (m,n) dot (n,k).
    // In general —  $O(m \cdot n \cdot k)$ 
    // Returns MIPS estimation assuming double+double = 1 MI
    private static double CalculateOverallComplexity(long m, long n, long k,
double mulComplexityMultiplication)
    {
121        //return m*k*(n*mulComplexityMultiplication + n-1);
        return m*k*n*mulComplexityMultiplication;
    }

    private static List<Vm> createVmsWithMIPS(List<Long> MIPS) {
126        final List<Vm> list = new ArrayList<>(MIPS.size());
        for (int i = 0; i < MIPS.size(); i++) {

```



```

        Vm vm = new VmSimple(i, MIPS.get(i), 1)
            .setRam(512).setBw(1000)
            .setCloudletScheduler(new CloudletSchedulerSpaceShared());
131         //.setCloudletScheduler(new CloudletSchedulerTimeShared());
        list.add(vm);
    }
    return list;
}

136
private static Datacenter createDatacenter(Simulation sim) {
    final List<Host> hostList = new ArrayList<>(HOSTS);
    for(int i = 0; i < HOSTS; i++) {
        Host host = createHost();
141        hostList.add(host);
    }
    DatacenterCharacteristics characteristics = new
DatacenterCharacteristicsSimple(hostList);
    final Datacenter dc = new DataCenterSimpleFixed(sim, characteristics, new
VmAllocationPolicySimple());
    return dc;
146 }

private static Host createHost() {
    List<Pe> peList = new ArrayList<>(HOST_PES);
    //List of Host's CPUs (Processing Elements, PEs)
151 for (int i = 0; i < HOST_PES; i++) {
        peList.add(new PeSimple(HOST_P_MIPS, new PeProvisionerSimple()));
    }

    final long ram = 1000000; //in Megabytes
156 final long bandwidth = 10000000; //in Megabits/s
    final long storage = HUGE_VALUE; //in Megabytes
    ResourceProvisioner ramProvisioner = new ResourceProvisionerSimple();
    ResourceProvisioner bwProvisioner = new ResourceProvisionerSimple();
    VmScheduler vmScheduler = new VmSchedulerSpaceShared();
161 Host host = new HostSimple(ram, bandwidth, storage, peList);

    host.setRamProvisioner(ramProvisioner).setBwProvisioner(bwProvisioner).setVmScheduler(
        return host;
    }

166 private static List<Cloudlet> createCloudlets(int userid, long n, long n1,
int nextid)
{
    if(n1==0)
        return new ArrayList<>();
    long n_slices = n/n1;

```

```

171     boolean partial_slice = n%n1 != 0;
        long n2 = n - n_slices*n1;

        long n_cloudlets = n_slices*n_slices;
        if(partial_slice)
176     {
            n_cloudlets+= 2*n_slices + 1;
        }

        final List<Cloudlet> list = new ArrayList<>((int)n_cloudlets);
181     UtilizationModel utilization = new UtilizationModelFull();

        long full_complexity =
        (long) (CalculateOverallComplexity(n1,n,n1,multiplicationComplexityMultiplier));
        long p_complexity =
        (long) (CalculateOverallComplexity(n1,n,n2,multiplicationComplexityMultiplier));
        long pp_complexity =
        (long) (CalculateOverallComplexity(n2,n,n2,multiplicationComplexityMultiplier));
186

        for (int i = 0; i < n_slices; i++)
        {
            for (int j = 0; j < n_slices; j++)
            {
191                list.add(new CloudletUser(userid, ++nextid, full_complexity, 1)
                    .setUtilizationModel(utilization));
            }
        }
        if(partial_slice)
196     {
            for (int i = 0; i < n_slices; i++)
            {
                list.add(new CloudletUser(userid, ++nextid, p_complexity, 1)
                    .setUtilizationModel(utilization));
201                list.add(new CloudletUser(userid, ++nextid, p_complexity, 1)
                    .setUtilizationModel(utilization));
            }
            list.add(new CloudletUser(userid, ++nextid, pp_complexity, 1)
                .setUtilizationModel(utilization));
206        }

        return list;
    }

211     public static RunResult simulateProblem(Class brokerClass, List<Long>
MIPSCapacities,

                                                long size, long slice1, long slice2,
        boolean debug_info)

```

```

        throws NoSuchMethodException, InstantiationException,
        IllegalAccessException, IllegalArgumentException,
InvocationTargetException
    {
216         MatrixMultiplicationSimulation MMS = new
MatrixMultiplicationSimulation(size, slice1, slice2);

        double start, end;

        CloudSim simulation = new CloudSim(-1, new GregorianCalendar(), false,
0.0001);
221

        Datacenter datacenter = createDatacenter(simulation);

        Constructor constructor = brokerClass.getConstructor(CloudSim.class);
        DatacenterBroker broker =
        (DatacenterBroker) constructor.newInstance(simulation);
226

        List<Vm> vmList = createVmsWithMIPS(MIPSCapacities);
        broker.submitVmList(vmList);

        start = System.nanoTime();
231        List<Cloudlet> cloudlets = new ArrayList<>();
        cloudlets.addAll(createCloudlets(0, MMS.problemSize(), MMS.getSlice1(),
0));
        cloudlets.addAll(createCloudlets(1, MMS.problemSize(), MMS.getSlice2(),
cloudlets.get(cloudlets.size() - 1).getId()));
        end = System.nanoTime();

236        // DEBUG INFO
        if(debug_info)
        {
            System.out.println("Cloudlet_creation_duration");
            System.out.println(Double.toString((end - start) / 1e9));
241            System.out.println(String.format("Submitted_%d_cloudlets",
cloudlets.size()));
        }

        broker.submitCloudletList(cloudlets);

246        start = System.nanoTime();
        simulation.start();
        end = System.nanoTime();

        List<Cloudlet> received = broker.getCloudletFinishedList();
251

        if(debug_info)

```

```

    {
        System.out.println(String.format("Received_%d_cloudlets",
received.size()));
    }
256    if(received.size() != cloudlets.size()) {
        throw new RuntimeException(
            String.format(
                "Cloudlet_processing_corruption.\n" +
                "Simulation_case:\n" +
261                "Scheduler:_%s_Problem_size:_%d.Slice_size1:_%d.Slice_size2:_%d.",
                brokerClass.getName(), MMS.problemSize(),
MMS.getSlise1(), MMS.getSlise2()
            )
        );
    }
266    received.sort((o1, o2) ->
(Double.compare(o2.getFinishTime(), o1.getFinishTime())));

    double times[] = new double[2];

    CloudletUser last_cloudlet = (CloudletUser)received.get(0);
271

    int reverse_user = 0;
    if(last_cloudlet.getUser_id()==0)
    {
        reverse_user = 1;
276        times[0] = last_cloudlet.getFinishTime();
    }
    else
    {
        reverse_user = 0;
281        times[1] = last_cloudlet.getFinishTime();
    }
    for(Cloudlet cl : received)
    {
        CloudletUser cl_user = (CloudletUser)cl;
286        if(cl_user.getUser_id() == reverse_user)
        {
            times[reverse_user] = cl.getFinishTime();
            break;
        }
291    }

    double last_arrival_time = Math.max(times[0], times[1]);

```

```

        double last_arrival_time2 = received.stream().mapToDouble(cl ->
cl.getFinishTime()).max().getAsDouble();
296         if(last_arrival_time != last_arrival_time2)
            throw new RuntimeException("Arrival_time_corruption");

        // DEBUG INFO
301         if(debug_info)
        {
            System.out.println(String.format("Simulation_duration_%f_seconds",
(end - start) / 1e9));
            System.out.println(String.format("Simulation_internal_duration_%f_
seconds", last_arrival_time));

306            System.out.println(String.format("Received_%d_cloudlets",
received.size()));
            System.out.println("received_cloudlets_table_sorted_from_last_
received_to_first");
            new CloudletsTableBuilderExtended(received).build();
        }

311         return new RunResult(times[0], times[1]);
    }

    static void simulate(Class brokerClass, List<Long> MipsCapacities,
                        long problem_size, Iterable<Long> slices1,
Iterable<Long> slices2,
316                        String resulting_filename,
                        boolean debug_info)
    {
        try {
            double start = System.nanoTime();

321            FileWriter writer = new FileWriter(resulting_filename, false);
            DateFormat dateFormat = new SimpleDateFormat("yyyy.MM.dd_HH:mm:ss");
            Date date = new Date();
            writer.write(String.format("Starting_time:_%s\n",
dateFormat.format(date)));

326            writer.write(String.format("Using_scheduler_%s\n",
brokerClass.getName()));
            writer.write("First_player,");
            writer.write("Second_player,");
            writer.write("TimeFirst(in_seconds)");
            writer.write("TimeSecond(in_seconds)\n");

331            writer.flush();

            for (long i : slices1) {

```

```

        for (long j : slices2) {
            RunResult time = simulateProblem(brokerClass, MipsCapacities,
problem_size, i, j, debug_info);
336            writer.write(String.format("%d,%d,%f,%f\n", i, j,
time.getFirst(), time.getSecond()));
            writer.flush();
        }
    }

341    double end = System.nanoTime();
    System.out.println("Time_for_all_simulations(in_seconds)_");
    System.out.println(Double.toString((end - start) / 1e9));

    writer.close();
346    }
    catch (Exception e)
    {
        System.out.println(String.format("Exception_in_simulate:_%s.\n_
Reason:_%s._Stack_trace:_%s", e.getMessage(), e.getCause(),
e.getStackTrace()));
    }
351    }

    static void simulate_full(Class brokerClass, List<Long> MipsCapacities,
        long problem_size, long start_slice, long max_slice,
        String resulting_filename,
356        boolean debug_info)
    {
        List<Long> slices = new ArrayList<>((int) (max_slice-start_slice));
        for(long i = start_slice; i < max_slice; ++i)
            slices.add(i);
361        simulate(brokerClass, MipsCapacities, problem_size, slices, slices,
resulting_filename, debug_info);
    }

    // maybe do random slice list uniformly distributed?
    static void simulate_urandom(Class brokerClass, List<Long> MipsCapacities,
366        long problem_size, long start_slice, long
max_slice, long count,
        String resulting_filename,
        boolean debug_info)
    {
        List<Long> slices = new ArrayList<>((int) (max_slice-start_slice));
371        for(long i = start_slice; i < max_slice; ++i)
            slices.add(i);
        simulate(brokerClass, MipsCapacities, problem_size, slices, slices,
resulting_filename, debug_info);
    }

```

```

    }

376     static void simulate_step(Class brokerClass, List<Long> MipsCapacities,
                                long problem_size, long start_slice, long
max_slice, long step,
                                String resulting_filename, boolean single,
                                boolean debug_info)
    {
381         List<Long> slices = new ArrayList<>((int) (max_slice-start_slice));
        for(long i = start_slice; i < max_slice; i+=step)
            slices.add(i);
        List<Long> slices2;
        if(single)
386         {
            slices2 = new ArrayList<>();
            slices2.add(0L);
        }
        else
391         slices2 = slices;
        simulate(brokerClass, MipsCapacities, problem_size, slices, slices2,
resulting_filename, debug_info);
    }

    public static void main(String[] args)
396         throws IOException, NoSuchMethodException, InstantiationException,
        IllegalAccessException, IllegalArgumentException,
InvocationTargetException
    {
        Options options = new Options();
        options.addOption(
401            Option.builder()
                .longOpt("debug")
                .hasArg(false)
                .required(false)
                .desc("print_debug_information")
406                .build()
        );
        options.addOption(
            Option.builder()
                .longOpt("single")
411                .hasArg(false)
                .required(false)
                .desc("single_player")
                .build()
        );
416        options.addOption(
            Option.builder("o")

```

```

        .longOpt("output")
        .hasArg(true)
        .required(false)
421     .desc("output_file")
        .build()
    );
    options.addOption(
        Option.builder()
426         .longOpt("print_logs")
        .hasArg(false)
        .required(false)
        .desc("enable_logging")
        .build()
431    );
    options.addOption(
        Option.builder("n")
        .longOpt("problem_size")
        .hasArg(true)
436         .required(false)
        .desc("problem_size_(n,n).dot(n,n)")
        .build()
    );
    options.addOption(
441         Option.builder()
        .longOpt("step_size")
        .hasArg(true)
        .required(false)
        .desc("problem_size_increment_stepsizes")
446         .build()
    );
    options.addOption(
        Option.builder()
        .longOpt("start_slice")
451         .hasArg(true)
        .required(false)
        .desc("starting_slice_size")
        .build()
    );
456    options.addOption(
        Option.builder()
        .longOpt("max_slice")
        .hasArg(true)
        .required(false)
461         .desc("max_slice_size")
        .build()
    );

```



```

options.addOption(
466     Option.builder()
        .longOpt("scheduler")
        .hasArg(true)
        .required(false)
        .desc("scheduler_type:_minmin,_minmax,_maxmin,_
maxmax")
471     .build()
);
options.addOption(
    Option.builder()
        .longOpt("mips")
476     .hasArgs()
        .required()
        .valueSeparator(',')
        .desc(String.format("comma_separated_mips_capacities_
as_multiplication_of_nominal_(%d)", NOMINAL_MIPS))
        .build()
481 );
options.addOption(
    Option.builder()
        .longOpt("mips")
        .hasArgs()
486     .required()
        .valueSeparator(',')
        .desc(String.format("comma_separated_mips_capacities_
as_multiplication_of_nominal_(%d)", NOMINAL_MIPS))
        .build()
);
491 CommandLineParser parser = new DefaultParser();
try
{
    CommandLine line = parser.parse( options, args );

496     final boolean print_debug = line.hasOption("debug") ? true :
false;

    if(!line.hasOption("print_logs"))
        Log.disable();
    final String output_filename = line.getOptionValue("output",
"results.txt");

501     Class brokerClass = null;
    String broker_class_str=line.getOptionValue("scheduler",
"minmax");
    brokerClass =
scheduler_mapper.getDefault(broker_class_str.toLowerCase(), SimpleSchedulers.MaxMinSch

```

```

String[] mipss_str = line.getOptionValues("mips");
506 List<Long> MipsCapacities = new ArrayList<>();
    for(String mips_str : mipss_str)

MipsCapacities.add((long) (NOMINAL_MIPS*Double.parseDouble(mips_str)));

    boolean single = line.hasOption("single");
511

    final long n = Long.parseLong(line.getOptionValue("problem_size",
"1000"));
    final long step = Long.parseLong(line.getOptionValue("step_size",
"5"));

    final long start_slice;
    if(line.hasOption("start_slice")){
516         start_slice =
Long.parseLong(line.getOptionValue("start_slice"));
    }
    else {
        start_slice = Math.max(n/200, 50);
    }
521 final long max_slice;
    if(line.hasOption("max_slice")){
        max_slice = Long.parseLong(line.getOptionValue("max_slice"));
    }
    else {
526         max_slice = 3*n/4;
    }

    switch (broker_class_str)
    {
531         case "all_mt": {
            ExecutorService es = newFixedThreadPool(4);
            for (Class current_scheduler : scheduler_mapper.values())
            {
                if(current_scheduler!=null) {
                    DateFormat dateFormat = new
SimpleDateFormat("yyyy_MM_dd__HH_mm_ss");
536                    Date date = new Date();
                    String filename = String.format("%s.%s.txt",
current_scheduler.getSimpleName(), dateFormat.format(date));
                    es.execute(() -> simulate_step(current_scheduler,
MipsCapacities, n, start_slice, max_slice, step, filename, single,
print_debug));
                }
            }
541            es.shutdown();
            try {

```

```

        es.awaitTermination(Long.MAX_VALUE, TimeUnit.SECONDS);
    } catch (InterruptedException e) {
        e.printStackTrace();
546    }
        break;
    }
    case "all_st": {
        for (Class current_scheduler : scheduler_mapper.values())
        {
551            if(current_scheduler!=null) {
                DateFormat dateFormat = new
SimpleDateFormat("yyyy_MM_dd__HH_mm_ss");
                Date date = new Date();
                String filename = String.format("%s.%s.txt",
current_scheduler.getSimpleName(), dateFormat.format(date));
                simulate_step(current_scheduler, MipsCapacities,
n, start_slice, max_slice, step, filename, single, print_debug);
556            }
        }
        break;
    }
    default:
561        simulate_step(brokerClass, MipsCapacities, n,
start_slice, max_slice, step, output_filename, single, print_debug);

        }
    }
    catch( ParseException exp )
566    {
        System.err.println( "Parsing failed. Reason: " +
exp.getMessage() );
        HelpFormatter formatter = new HelpFormatter();
        formatter.printHelp( "matrixsim", options );
    }
571 }
}

```

../code/cloud_simulations/src/main/java/com/simulations/matrix/SimpleSchedulers.java

```
package com.simulations.matrix;
```

```
3 import org.cloudbus.cloudsim.core.CloudSim;
```

```
public class SimpleSchedulers
```

```
{
```

```
    public static class MaxMaxScheduler extends BasicScheduler
```

```
8    {
```

```

    public MaxMaxScheduler(CloudSim simulation)
    {
        super(simulation);
        setVmComparator((vm1,vm2)->Double.compare(vm2.getMips(),
vm1.getMips()));
13        setCloudletComparator((c1,c2)->Long.compare(c2.getLength(),
c1.getLength()));
    }
}

public static class MinMinScheduler extends BasicScheduler
18 {
    public MinMinScheduler(CloudSim simulation)
    {
        super(simulation);
        setVmComparator((vm1,vm2)->Double.compare(vm1.getMips(),
vm2.getMips()));
23        setCloudletComparator((c1,c2)->Long.compare(c1.getLength(),
c2.getLength()));
    }
}

public static class MinMaxScheduler extends BasicScheduler
28 {
    public MinMaxScheduler(CloudSim simulation)
    {
        super(simulation);
        setVmComparator((vm1,vm2)->Double.compare(vm2.getMips(),
vm1.getMips()));
33        setCloudletComparator((c1,c2)->Long.compare(c1.getLength(),
c2.getLength()));
    }
}

public static class MaxMinScheduler extends BasicScheduler
38 {
    public MaxMinScheduler(CloudSim simulation)
    {
        super(simulation);
        setVmComparator((vm1,vm2)->Double.compare(vm1.getMips(),
vm2.getMips()));
43        setCloudletComparator((c1,c2)->Long.compare(c2.getLength(),
c1.getLength()));
    }
}
}

```

Додаткові скрипти для обробки результатів

../scripts/plot.py

```
import os
from os import path
import sys

4
import numpy as np
from plotly import offline as pltly
from plotly import graph_objs as pltly_obj

9 def plot_results(fpath, fname, subset = 0.01):
    fname_noex = fname[:fname.rfind(".")]

    full_data = np.loadtxt(path.join(fpath, fname), skiprows=3, delimiter=',')

14    idx = np.random.choice([True,False], size=len(full_data), p=[subset,
        1-subset])

    data = full_data[idx]

    x, y, time1, time2= data[:,0], data[:,1], data[:,2], data[:,3]

19
    trace1 = pltly_obj.Scatter3d(
        x=x,
        y=y,
        z=time1,
24        mode='markers',
        marker={'size':2, 'color':time1, 'colorscale':'Jet'},
        stream={'maxpoints':20}
    )

29    trace2 = pltly_obj.Scatter3d(
        x=x,
        y=y,
        z=time2,
        mode='markers',
34        marker={'size':2, 'color':time2, 'colorscale':'Viridis'},
        stream={'maxpoints':20}
    )

    pltly.plot([trace1,trace2], filename=fname_noex+".html", auto_open=False)

39
```

```

if __name__ == "__main__":
    subset = 0.01
    if len(sys.argv) > 1:
        subset = float(sys.argv[1])
44    files = os.listdir("./")
    target_files = filter(lambda x: path.isfile(path.join("./",x)) and
x.endswith(".txt"), files)
    for tf in target_files:
        plot_results("./", tf, subset)

```

../scripts/plot_single.py

```

import os
2 from os import path
import sys

import numpy as np
from plotly import offline as pltly
7 from plotly import graph_objs as pltly_obj

def plot_results(fpath, fname, stop_point = 100):
    fname_noex = fname[:fname.rfind(".")]

12    data = np.loadtxt(path.join(fpath, fname), skiprows=3, delimiter=',')

    x, y = data[:stop_point,0], data[:stop_point,2]

    trace = pltly_obj.Scatter(
17        x = x,
        y = y
    )

    pltly.plot([trace], filename=fname_noex+".html", auto_open=False)
22

if __name__ == "__main__":
    stop_point = -1
    if len(sys.argv)>1:
        stop_point = int(sys.argv[1])
27    files = os.listdir("./")
    target_files = filter(lambda x: path.isfile(path.join("./",x)) and
x.endswith(".txt"), files)
    for tf in target_files:
        plot_results("./", tf, stop_point)

```

../scripts/plot_single_cpp.py

```

import os
from os import path

```

```

import sys

4
import numpy as np
from plotly import offline as pltly
from plotly import graph_objs as pltly_obj

9 def plot_results(fpath, fname):
    fname_noex = fname[:fname.rfind(".")]

    full_data = np.loadtxt(path.join(fpath, fname), skiprows=1, delimiter=',')

14    data = full_data

    x, time= data[:,0], data[:,1]

    trace1 = pltly_obj.Scatter(
19         x=x,
        y=time
    )

    pltly.plot([trace1], filename=fname_noex+".html", auto_open=False)

24
if __name__ == "__main__":
    files = os.listdir("./")
    target_files = filter(lambda x: path.isfile(path.join("./",x)) and
x.endswith(".txt"), files)
    for tf in target_files:
29        plot_results("./", tf)

```

../scripts/plot_single_cpp_together.py

```

import os
from os import path
import sys

5 import numpy as np
from plotly import offline as pltly
from plotly import graph_objs as pltly_obj

def plot_results(fpath, fname):
10    fname_noex = fname[:fname.rfind(".")]
    full_data = np.loadtxt(path.join(fpath, fname), skiprows=1, delimiter=',')

    data = full_data

15    x, time= data[:,0], data[:,1]

```

```

        trace1 = pltly_obj.Scatter(
            x=x,
            y=time
20         )

        pltly.plot([trace1], filename=fname_noex+".html", auto_open=False)

25 def plot_results(datas, names):
    traces = []
    for d, nm in zip(datas, names):
        name = nm[:nm.rfind(".")]
        trace = pltly_obj.Scatter(x=d[:,0],y=d[:,1], name = name)
30     traces.append(trace)
    pltly.plot(traces, filename="result"+"html", auto_open=False)

    if __name__ == "__main__":
35         files = os.listdir("./")
        target_files = list(filter(lambda x: path.isfile(path.join("./",x)) and
                                x.endswith(".txt"), files))
        datas = []
        for tf in target_files:
            full_data = np.loadtxt(path.join("./", tf), skiprows=1, delimiter=',')
40         datas.append(full_data)
        plot_results(datas, target_files)

```

../scripts/plot_surf.py

```

import os
from os import path
3 import sys

import numpy as np
from plotly import offline as pltly
from plotly import graph_objs as pltly_obj
8

def plot_results(fpath, fname, subset = 1):
    fname_noex = fname[:fname.rfind(".")]

    full_data = np.loadtxt(path.join(fpath, fname), skiprows=1, delimiter=',')
13

    idx = np.random.choice([True,False], size=len(full_data), p=[subset,
        1-subset])

    data = full_data#[idx]

```



```

18     x = data[:,0]
        mn = int(np.min(x))
        mx = int(np.max(x))

        x = np.array(range(mn, mx+1))
23     y = np.array(range(mn, mx+1))
        z1 = data[:,2].reshape((mx - mn + 1,-1))
        z2 = data[:,3].reshape((mx - mn + 1,-1))

        trace1 = pltly_obj.Surface(
28             x=x,
                y=y,
                z=z1
            )

33     trace2 = pltly_obj.Surface(
            x=x,
            y=y,
            z=z2
        )

38     pltly.plot([trace1], filename=fname_noex+".html", auto_open=False)

if __name__ == "__main__":
    subset = 0.01
43     if len(sys.argv) > 1:
        subset = float(sys.argv[1])
        files = os.listdir("./")
        target_files = filter(lambda x: path.isfile(path.join("./",x)) and
            x.endswith(".txt"), files)
        for tf in target_files:
48             plot_results("./", tf, subset)

```

../scripts/plot_surf_matplotlib.py

```

1 import os
    from os import path
    import sys

    import numpy as np
6 from scipy.interpolate import griddata

    from matplotlib import cm
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import Axes3D
11

```

```

def plot_results(fpath, fname, subset = 1):
    fname_noex = fname[:fname.rfind(".")]

16     full_data = np.loadtxt(path.join(fpath, fname), skiprows=1, delimiter=',')

    #idx = np.random.choice([True,False], size=len(full_data), p=[subset,
    1-subset])

    data = full_data

21     x,y,z1,z2 = data[:, 0],data[:, 1],data[:, 2],data[:, 3]
    X,Y = np.meshgrid(x,y)
    Z1 = griddata((x,y), z1, (X,Y), method='cubic')
    Z2 = griddata((x,y), z2, (X,Y), method='cubic')

26

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, Z2, linewidth=0, antialiased=False)

31     plt.show()


36 if __name__ == "__main__":
    subset = 0.01
    if len(sys.argv) > 1:
        subset = float(sys.argv[1])
    files = os.listdir("./")
41     target_files = filter(lambda x: path.isfile(path.join("./",x)) and
    x.endswith(".txt"), files)
    for tf in target_files:
        plot_results("./", tf, subset)

```

../scripts/plot_surf_together_wireframe.py

```

1 import os
  from os import path
  import sys

  import colorlover as cl

6
  import numpy as np
  from scipy.interpolate import griddata

  from plotly import offline as pltly
11 from plotly import graph_objs as pltly_obj

```

```

colors = ['#FF0000', '#00FF00', '#0000FF', '#FFFF00', '#00FFFF', '#FF00FF']

def plot_results(datas, fnames, subset = 1):
16
    lines = []
    counter = 0
    for (full_data, fname) in zip(datas, fnames):
        fname_noex = fname[:fname.rfind(".")]
21
        #idx = np.random.choice([True, False], size=len(full_data), p=[subset,
        1-subset])

        data = full_data
        x, y, z1, z2 = data[:, 0], data[:, 1], data[:, 2], data[:, 3]
26
        xi = np.linspace(min(x), max(x), num=int(max(x) - min(x)))
        yi = np.linspace(min(y), max(y), num=int(max(y) - min(y)))
        X, Y = np.meshgrid(xi, yi)
        Z = griddata((x, y), z1, (X, Y), method='cubic')
31
        line_marker = dict(color=colors[counter], width=2)
        lines.append(pltly_obj.Scatter3d(x=[0], y=[0], z=[0], name =
        fname_noex+"wf", legendgroup=fname_noex+"wf", showlegend=True,
        visible='legendonly', mode='marker', marker={'size':0.01}))
        for i, j, k in zip(X, Y, Z):
            lines.append(pltly_obj.Scatter3d(x=i, y=j, z=k,
            legendgroup=fname_noex+"wf", showlegend=False, mode='lines',
            line=line_marker))
36
        counter = counter + 1

    layout = pltly_obj.Layout(title = '□', showlegend=True)
    fig = dict(data=lines, layout = layout)
    pltly.plot(fig, filename="result.html", auto_open=False)
41
if __name__ == "__main__":
    subset = 0.01
    if len(sys.argv) > 1:
        subset = float(sys.argv[1])
46
    files = os.listdir("./")
    target_files = list(filter(lambda x: path.isfile(path.join("./", x)) and
    x.endswith(".txt"), files))

    datas = [np.loadtxt(path.join('./', fname), skiprows=1, delimiter=',') for
    fname in target_files]
    plot_results(datas, target_files, subset)

```

../scripts/utils.py

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
4 import nash

def divisorGenerator(n):
    large_divisors = []
    for i in range(1, int(np.sqrt(n) + 1)):
9         if n % i == 0:
            yield i
            if i*i != n:
                large_divisors.append(n / i)
    for divisor in reversed(large_divisors):
14         yield divisor

def divisors(n):
    return np.array(list(divisorGenerator(n)), dtype=np.int)

19 def argmin(array):
    return np.unravel_index(np.argmin(array, axis=None), array.shape)

def read_file(file_name):
24     file = open(file_name, "r")
    general_info = file.readline()
    slices = file.readline()
    slices = list(map(lambda x: int(x), slices[slices.find("=")+1:].split("-")))
    mips = file.readline()
29     file.readline()
    data = np.loadtxt(file, delimiter=",")
    general_info = {g[:g.find("=")]:float(g[g.find("=")+1:]) for g in
    general_info.split("|||")}

    n = len(slices)
34     data1 = data[:,2]
    data2 = data[:,3]
    df1 = pd.DataFrame(data1.reshape((n,n)), index=slices, columns=slices)
    df2 = pd.DataFrame(data2.reshape((n,n)), index=slices, columns=slices)

39     return general_info, np.array(slices), df1, df2

def descend(matrix, a_0, b_0):
    n_v = matrix.shape[0]
44     n_h = matrix.shape[1]
```

```

p = np.array([a_0,b_0])
p_history = [p]

while True:
49     fl = max(p[0]-1,0)
        fr = min(p[0]+2,n_h)
        sl = max(p[1]-1,0)
        sr = min(p[1]+2,n_v)
        block = matrix[fl:fr, sl:sr]
54     next_offset = argmin(block)
        p_next = p + next_offset
        if p[0] != 0:
            p_next[0] = p_next[0] - 1
        if p[1] != 0:
59     p_next[1] = p_next[1] - 1
        p = p_next
        p_history.append(p)
        if np.all(p_history[-1] == p_history[-2]):
            break
64     return np.array(p_history[:-1])

gen, slices, data_f, data_s =
    read_file("./sim_min_min_n_2500_m_1.000e+10_bw_1.000e+08_p_0.000e+00_rd_10.txt")

def run_descend_print(matrix, a_0, b_0, slices):
69     idx = descend(matrix, a_0, b_0)
        print(idx)
        print("start_{}_{}_".format(slices[idx[0]]))
        print("end_{}_{}_".format(slices[idx[-1]]))
        print("time_{}_{}_".format(matrix[idx[0][0], idx[0][1]]))
74     print("time_{}_{}_".format(matrix[idx[-1][0], idx[-1][1]]))

```