КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформатика»
за спеціальністю 122 Комп’ютерні науки

на тему:

РОЗРОБКА СИСТЕМИ ПАРКС.NET CORE

Виконав студент 4-го курсу
Павло РЕДЬКО _____
(підпис)

Науковий керівник:
Асистент, кандидат фізико-математичних наук
Олексій ФЕДОРУС _____
(підпис)

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

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики
«___»_____2023 р.
протокол №____
Завідувач кафедри
Василь ТЕРЕЩЕНКО _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 40 сторінок, 7 ілюстрацій, 2 таблиці, 16 джерел посилань.

Об'єктом розробки є система для паралельних обчислень ПАРКС.NET Core. Предметом роботи є застосунок для паралельних обчислень з використанням системи ПАРКС.NET Core.

Метою роботи є реалізація оновленої системи, яка забезпечує швидше виконання обчислень порівняно зі старою системою ПАРКС.NET.

Інструменти дослідження: інтегроване середовище розробки JetBrains Rider, мова програмування C#, хостинг Google Cloud, емулятор консолі Terminal.

Результат роботи: досліджено наявну систему ПАРКС.NET, визначено недоліки та можливі виправлення, розроблено оновлену систему ПАРКС.NET Core, яка забезпечить виправлення недоліків, збільшення швидкодії та оновлюваності системи.

Розроблений програмний продукт забезпечує високу швидкодію обчислень у різних сферах застосування, таких як високонавантажені системи, наукові дослідження, обробка даних. Результати дослідження підтверджують значимість, перспективи та зручність розробленої системи ПАРКС.NET Core.

ЗМІСТ

ВСТУП

Розвиток технологій у сфері програмного забезпечення постійно вимагає оновлення і модернізації існуючих систем. Технологія ПАРКС є потужним інструментом для розробки паралельних систем з асинхронним керуванням та рекурсивними структурами. Вона знаходить застосування у різних галузях, таких як великі дата-центри, обробка великих обсягів даних та інтенсивні обчислення. Однак, з метою покращення швидкодії, безпеки, стабільності та забезпечення кросплатформності, необхідно провести оновлення системи ПАРКС до нової версії платформи .NET.

У рамках цієї дипломної роботи розглядається процес розробки системи ПАРКС (Паралельні Асинхронні Рекурсивно Керовані Системи) на платформі .NET 7 (раніше відомої як .NET Core).

Метою даної роботи є оновлення системи ПАРКС.NET для збільшення швидкодії, забезпечення високого рівня безпеки, зниження витрат ресурсів та підвищення стабільності системи.

Були поставлені такі завдання:

1. Проаналізувати існуючу системи ПАРКС.NET, визначити недоліки реалізації
2. Розробити нову систему ПАРКС.NET Core

Об’єктом є консольний застосунок для паралельних обчислень з використанням системи ПАРКС.NET Core. Для його створення розглядаються сучасні підходи та методики міграції з платформи .NET Framework на .NET, проводиться аналіз вже існуючого рішення ПАРКС.NET та розробляється нова система ПАРКС.NET Core з оптимізацією та виправленням недоліків. Для розробки використовувалися інтегроване середовище розробки JetBrains Rider для мови програмування C# та емулятор консолі Terminal.

Розроблена система може використовуватись у різних сферах, де потрібні паралельні та швидкі обчислення, такі як: аналіз даних, обчислювальна

інфраструктура, хмарні обчислення та великі проєкти з обчислювальною складністю. Вона дозволяє розподіляти обчислювальні завдання між багатьма вузлами, прискорюючи обчислення та забезпечуючи високу продуктивність обробки даних.

РОЗДІЛ 1. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

1.1 Огляд платформи .NET та її основні компоненти

Платформа .NET була розроблена компанією Microsoft і була вперше представлена у 2000 році. Вона була створена з метою забезпечити розробникам зручні та ефективні інструменти для розробки програмного забезпечення. Першою версією платформи .NET була .NET Framework 1.0, а згодом були випущені інші версії. Основними версіями .NET є:

- .NET Framework: Це платформа, що працює на операційних системах Windows і має широке застосування у веб-розробці та розробці Windows-додатків. .NET Framework почав розвиватися з 2002 року і включає багато розширень та бібліотек для розробки різних типів програм.
- .NET Core: Це відкрите та крос-платформне втілення .NET Framework, яке може працювати на різних операційних системах, включаючи мобільні операційні системи, Windows, macOS і Linux. .NET Core був представлений в 2016 році як легкий та ефективний фреймворк для розробки хмарних додатків та мікросервісів.
- .NET: Це нова ітерація платформи .NET, яка є наслідником .NET Core. .NET є крос-платформним фреймворком і надає зручний спосіб розробки програм для різних сценаріїв, включаючи веб-додатки, мобільні додатки, ігри та інше.

Всі версії .NET мають спільну базову інфраструктуру, що дозволяє розробникам використовувати різні мови програмування, такі як C#, VB.NET, F# і багато інших, для створення потужних та масштабованих додатків. Крім того, .NET надає широкий спектр бібліотек і інструментів для полегшення розробки і підтримки різних сценаріїв програмування [1].

Одним з головних компонентів платформи .NET є Common Language Runtime (CLR). CLR був вперше введений разом з .NET Framework 1.0. Він є віртуальною машиною, яка забезпечує виконання програмного коду, написаного на мовах програмування .NET. CLR виконує багато важливих функцій, таких як керування пам'яттю, безпека виконання коду, збирання сміття та інші операції, що сприяють безпеці та швидкодії виконання програм. Крім того, CLR дозволяє розробникам писати код на високорівневих мовах програмування, які компілюються в проміжний байт-код, зрозумілий CLR. Це забезпечує переносимість програм між різними платформами та мовами, що є однією з сильних сторін платформи .NET [2].

Ще одним важливим компонентом платформи .NET є бібліотеки класів .NET (NET Framework Class Library). Ці бібліотеки містять велику кількість готових до використання класів та компонентів, які спрощують процес розробки програм. Вони надають реалізацію багатьох загальноприйнятих завдань, таких як маніпулювання рядками, робота з файлами, мережеве взаємодіяння, доступ до баз даних і багато іншого. Використання бібліотек класів .NET дозволяє розробникам економити час і зусилля, оскільки вони можуть використовувати готовий функціонал замість написання власного коду для реалізації часто використовуваних завдань [1]. На рисунку 1 показано основні компоненти платформи .NET.

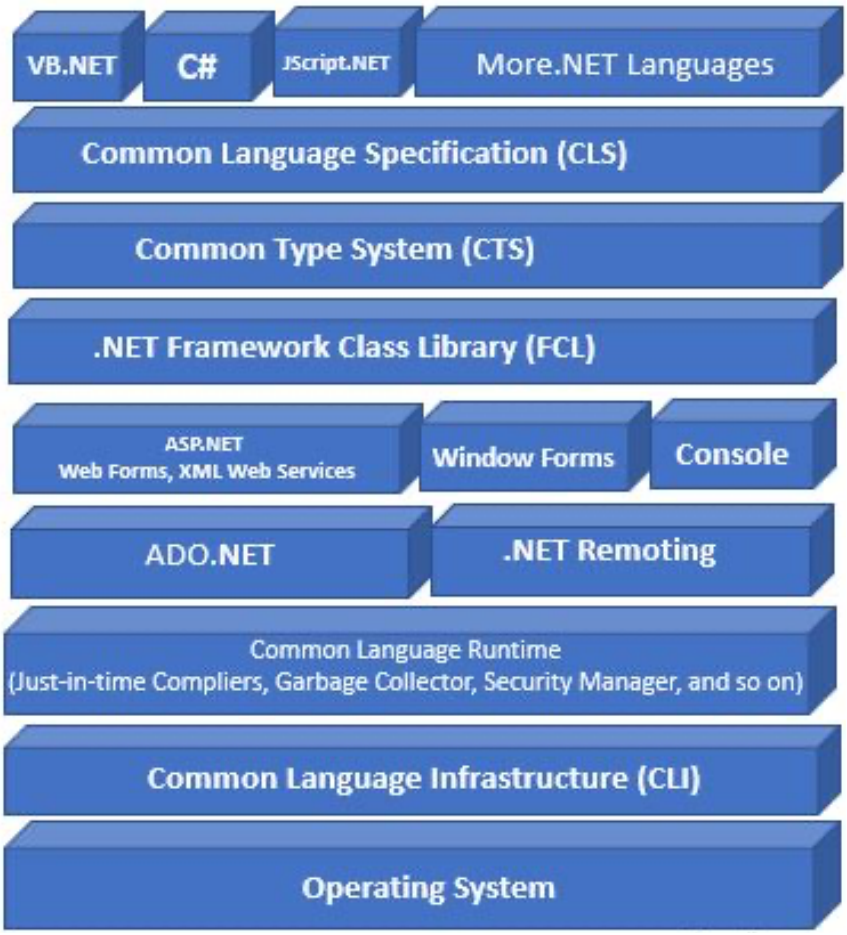

Рисунок 1 - Компоненти платформи .NET

Найпопулярнішою мовою програмування з платформи .NET є об'єктно-орієнтована мова C#. Вона була розроблена в середині 1990-х років в компанії Microsoft. Робота над мовою C# почалася у 1998 році під керівництвом Андерса Хейлсберга (Anders Hejlsberg) і була завершена у 2000 році з випуском першої версії мови. C# була створена як частина ініціативи Microsoft по розробці нової платформи для створення веб-сервісів та додатків під назвою .NET [1].

C# і Java є близькими родичами у світі програмування. Java була створена компанією Sun Microsystems як відповідь на потребу розподілених обчислень у розвитку Інтернету. Використовуючи C++ як основу, Java виправила певні небезпечні аспекти, такі як безконтрольні вказівники. Для розподілених обчислень була впроваджена концепція віртуальної машини і машинно-незалежного байт-коду. Популярність Java призвела до ліцензування її також компанією Microsoft. Проте, виник конфлікт між Sun Microsystems і Microsoft,

коли Microsoft створила власну версію Java, несумісну з концепцією машинно-незалежного середовища. Це призвело до судового процесу, під час якого Microsoft була зобов'язана припинити неліцензійне використання Java. Відповідно, Microsoft розробила власну мову C#, яка успадкувала концепції Java, включаючи віртуальну машину та байт-код, з метою створення власної платформи.

1.2 Огляд системи ПАРКС

ПАРКС-технологія програмування використовує набір програмних інструментів, які дозволяють розробляти та реалізовувати паралельні обчислення. Ця технологія базується на концепції керуючого простору.

Керуючий простір (КП) можна представити у вигляді динамічного графа, що використовується для опису логічної та комунікаційної структури задачі і відображення змін, що відбуваються в цьому просторі. Граф дозволяє визначити взаємозв'язки між різними елементами системи та відображає динамічні зміни, які відбуваються в процесі виконання обчислень [4].

Основне призначення ПАРКС - це забезпечення паралельної обробки інформації, а також підтримка розробки програмного забезпечення для паралельних обчислювальних систем.

Основні функції ПАРКС:

- Паралельне виконання: ПАРКС дозволяє виконувати алгоритми паралельно, тобто декілька алгоритмів можуть виконуватись одночасно на різних обчислювальних ресурсах. Це дозволяє покращити продуктивність і швидкодію програм.
- Асинхронність: підтримує асинхронні операції, що дозволяє виконувати різні частини програми незалежно одна від одної. Це особливо корисно, коли виконання деяких операцій може затримуватись (наприклад,

зчитування даних з мережі), і не потрібно чекати їх завершення для продовження виконання інших операцій.

- Рекурсія: підтримує рекурсивне викликання алгоритмів, що дозволяє розв'язувати завдання, що мають структуру дерева або графа, шляхом повторного виклику функцій для обробки підзадач.
- Динамічна модифікація структури керуючого простору: Одна з особливостей ПАРКС - це можливість динамічно змінювати структуру керуючого простору в процесі виконання програми. Це дає гнучкість і можливість адаптувати програму до змінних умов та вимог.
- Простота використання: ПАРКС надає високорівневі інструменти та інтерфейси програмування, що спрощують розробку паралельних програм. Завдяки цьому, розробники можуть швидко створювати ефективні паралельні програми без глибоких знань про низько рівневі деталі паралельного програмування.
- Розподілена обробка: ПАРКС підтримує розподілену обробку, що дозволяє виконувати програми на кластерах або розподілених системах. Це дозволяє розподіляти обчислювальне навантаження між різними вузлами і покращує масштабованість системи.
- Висока продуктивність: ПАРКС надає засоби для оптимізації продуктивності програм, зокрема, можливість ефективно використовувати багатоядерні процесори та розподілені ресурси. Це дозволяє досягти високої швидкодії і продуктивності паралельних програм [5].

У системі ПАРКС використовуються такі терміни як “точка”, “програмний канал” та “алгоритмічний модуль”, які важливі для розуміння концепції технології.

Точка - це один з елементів в системі, який представляє собою потік виконання. Кожна точка має свою функціональність та може виконувати алгоритмічний модуль. Програмний канал (ПК) - це зв'язок між точками, який дозволяє обмінюватися даними між ними. Різні точки можуть бути з'єднані

різними програмними каналами. Алгоритмічний модуль (АМ) - це блок коду, який виконується в окремій точці системи. Кожен алгоритмічний модуль працює паралельно з іншими модулями і може обмінюватися даними з іншими точками за допомогою програмних каналів [5].

Однією з ключових переваг системи ПАРКС є її простота у поясненні як самої концепції, так і можливостей для опису паралельних алгоритмів. Це дозволяє розробникам легко розуміти та впроваджувати паралельні обчислення, спрощуючи роботу з паралельними алгоритмами.

1.3 Основи C# та її використання в розробці ПАРКС.NET Core

Використання мови програмування C# є ключовим аспектом розробки системи ПАРКС. C# є потужною та гнучкою мовою, яка надає розробникам можливість створювати розподілені та паралельні програми з високою шкидкодією та зручністю. Вона має декілька особливостей та переваг, які роблять її ідеальним вибором для розробки систем, подібних до ПАРКС [4].

Одна з основних переваг C# полягає у її об'єктно-орієнтованому підході. C# є повністю об'єктно-орієнтованою мовою програмування, що дозволяє розробникам моделювати систему ПАРКС у вигляді об'єктів з відповідними властивостями і методами. Це спрощує розробку, управління та розширення системи ПАРКС [6].

Крім того, C# має широкий набір бібліотек та фреймворків, що полегшують розробку розподілених систем. Важливою перевагою C# є її підтримка багатопотоковості. Мова має потужні засоби для роботи з багатопотоковими програмами, що дозволяє ефективно використовувати ресурси кластерних систем та розподілених серверів. Завдяки цьому розробники можуть легко створювати та керувати багатопотоковими програмами у мові C#, що дозволяє досягти високої швидкодії в системі ПАРКС. C# також має багато вбудованих функцій та сервісів, що полегшують роботу з мережевими

операціями, серіалізацією, аутентифікацією та іншими аспектами, необхідними для розробки системи ПАРКС. Це дозволяє розробникам швидко реалізовувати необхідні функціональності та забезпечувати безперебійну роботу системи.

C# має легкий синтаксис та високу читабельність, що сприяє швидкому розробленню та підтримці коду системи ПАРКС. Це робить мову C# привабливою для команди розробників, оскільки вона дозволяє швидко писати, тестувати та підтримувати код [6].

Загалом, використання мови C# у розробці системи ПАРКС дозволяє розробникам створювати ефективні, масштабовані та розподілені системи. Багатофункціональність мови, широкий вибір бібліотек та платформа .NET роблять її потужним інструментом для розробки ПАРКС.

РОЗДІЛ 2. АНАЛІЗ ТА ПРОЄКТУВАННЯ СИСТЕМИ ПАРКС .NET

2.1 Визначення функціональних та нефункціональних вимог до системи

Для ефективного проектування системи ПАРКС на платформі .NET необхідно визначити функціональні та нефункціональні вимоги до цієї системи. Функціональні вимоги визначають, які функції та операції повинна виконувати система, в той час як нефункціональні вимоги стосуються якості, швидкодії та характеристик системи [7].

2.1.1. Функціональні вимоги

Моделювання керуючих просторів: Система повинна надавати можливість створювати керуючі простори для паралельного виконання процесів. Керуючі простори дозволяють розподіляти завдання між обчислювальними ресурсами та виконувати їх паралельно [9].

Паралельне виконання алгоритмів: Система має забезпечувати можливість паралельного виконання алгоритмів на різних обчислювальних ресурсах з метою покращення швидкодії програм.

Підтримка асинхронних операцій: Система повинна підтримувати асинхронні операції, що дозволяють виконувати різні частини програми незалежно одна від одної та покращують реактивність та швидкодію системи.

Рекурсивний виклик алгоритмів: Система має підтримувати рекурсивний виклик алгоритмів, що дозволяє розв'язувати задачі зі структурою дерева або графа шляхом повторного виклику функцій для обробки підзадач.

Динамічна модифікація структури керуючого простору: Система повинна надавати можливість динамічно змінювати структуру керуючого простору під час виконання програми з метою адаптації до змінних умов та вимог [8].

2.1.2. Нефункціональні вимоги

Швидкодія: Система повинна забезпечувати високу швидкодію виконання паралельних програм з використанням можливостей багатоядерних процесорів та розподілених ресурсів а також надавати засоби для оптимізації програм шляхом ефективного використання обчислювальних ресурсів [9].

Надійність: Система повинна бути надійною та відновлювати свою роботу після виникнення помилок або збоїв. Механізми відновлення повинні бути вбудовані для забезпечення надійності програм.

Масштабованість: Система має підтримувати масштабованість, що дозволяє розподіляти обчислювальне навантаження між різними вузлами та покращує продуктивність систем [8].

2.2 Аналіз сучасних рішень у сфері паралельних обчислень

Під час аналізу проектування системи ПАРКС на платформі .NET важливо ознайомитись з сучасними рішеннями у сфері паралельних обчислень, щоб визначити технології, методи та підходи, які можна використати для реалізації системи.

Можна виділити декілька напрямків у сфері паралельних обчислень. Один із них - акторні моделі. Акторні моделі є одним із підходів до реалізації паралельних систем, де програма складається з незалежних акторів, що обмінюються повідомленнями та виконують свої обчислення асинхронно. Один з відомих акторних фреймворків, який можна використати для розробки ПАРКС, - Akka.NET [10].

Ще одним напрямком є розподілені системи, які дозволяють розподіляти обчислювальне навантаження між різними вузлами мережі, забезпечуючи паралельне виконання задач. Наприклад, технологія Apache Hadoop надає засоби для обробки великих обсягів даних у розподіленому середовищі [11].

Також існують фреймворки та бібліотеки, які допомагають в розробці паралельних програм. Наприклад, фреймворк Parallel Extensions для .NET надає зручні засоби для паралельного програмування, включаючи можливість використання паралельних циклів, паралельних колекцій та інших конструкцій [12].

Важливим елементом систем ПАРКС є рекурсивні алгоритми, оскільки вони дозволяють розв'язувати складні задачі зі структурою дерева або графа. Один з прикладів рекурсивної моделі у контексті паралельного програмування - Parallel LINQ (PLINQ), що надає можливості паралельної обробки даних у LINQ-запитах [10].

2.3 Аналіз наявної системи ПАРКС.NET

Система ПАРКС.NET базується на клієнт-серверній архітектурі, де клієнтська програма і сервер взаємодіють для обміну даними та керування обчислювальними задачами. Клієнтська програма відповідає за взаємодію з користувачем, створення та передачу обчислювальних задач на сервер. Сервер, у свою чергу, приймає задачі від клієнтів, розподіляє їх між вузлами (нодами) та збирає результати обчислень для повернення клієнту.

Система ПАРКС.NET складається з декількох компонентів:

- Клієнтська програма - це інтерфейс для взаємодії з користувачем. Вона дозволяє користувачеві вводити параметри обчислювальних задач та відправляти їх на сервер для обробки. Клієнтська програма також отримує результати обчислень та відображає їх користувачеві.

- Сервер - центральний компонент системи, він отримує обчислювальні задачі від клієнтської програми і розподіляє їх між доступними вузлами (нодами). Сервер також відповідає за збір результатів обчислень з вузлів та їх повернення клієнтській програмі.
- Вузли (ноди) - це обчислювальні вузли, які виконують розрахунки над отриманими задачами. Вони приймають задачі від сервера, обробляють їх та повертають результати серверу.

Спосіб роботи:

1. Користувач запускає клієнтську програму і вводить параметри обчислювальної задачі.
2. Клієнтська програма відправляє задачу на сервер.
3. Сервер приймає задачу та розподіляє її між вузлами для обчислення.
4. Кожен вузол виконує обчислення над призначеною йому задачею.
5. Після завершення обчислень, вузол повертає результати серверу.
6. Сервер збирає результати від усіх вузлів та передає їх клієнтській програмі.
7. Клієнтська програма відображає результати користувачу.

Для зображення роботи програми було запущено обчислення алгоритмічного модуля на 4 точках системи. На рисунку 2 зображено процес виконання обчислень на одній з точок. На рисунку 3 зображено обчислення на сервері. На рисунку 4 зображено обчислення на алгоритмічному модулі. На цьому рисунку видно необхідну інформацію для клієнта, а саме: до яких точок системи під’єднано, час виконання та результат алгоритмічного модуля.

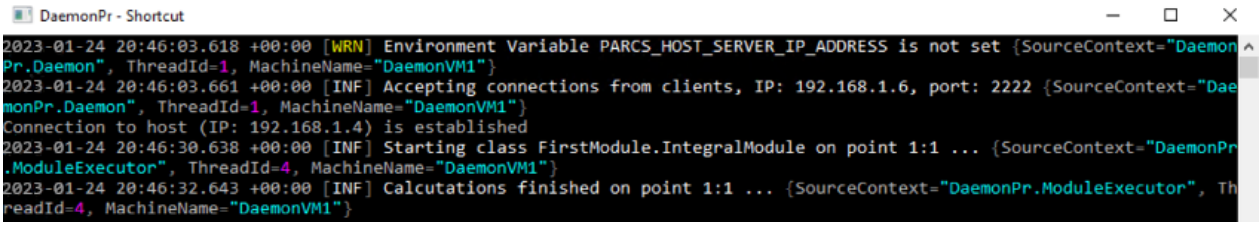

Рисунок 2 – Процес виконання обчислень на одній з точок

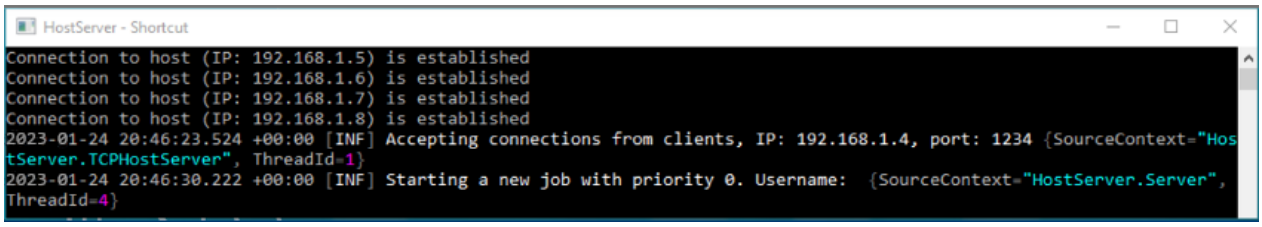

Рисунок 3 – Процес виконання обчислень на сервері

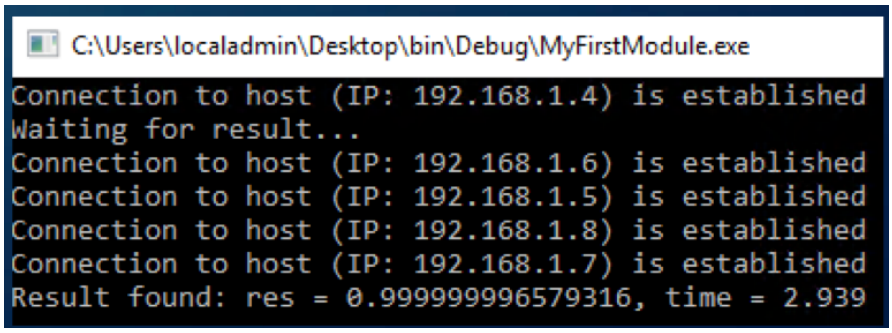

Рисунок 4 - Процес виконання обчислень на модулі

Переваги наявної системи ПАРКС.NET:

- Паралельні обчислення - система підтримує паралельні обчислення, що дозволяє ефективно використовувати ресурси багатоядерних та розподілених систем.
- Простота використання - система має зрозумілий інтерфейс для взаємодії з користувачем, що дозволяє легко створювати та відправляти обчислювальні задачі.
- Масштабованість - система може бути масштабована шляхом додавання нових вузлів для обчислення, що дозволяє розподілити навантаження та збільшити продуктивність.

Негаразди наявної системи ПАРКС.NET:

- Не використання асинхронності, а використання псевдо асинхронності: Відсутність використання асинхронних патернів (async/await) може призводити до блокування потоків та зниження продуктивності системи у випадках, коли обчислення затримуються.

- Стара версія .NET Framework - використання старої версії .NET Framework може обмежити доступність деяких функціональних можливостей та патчів безпеки, які присутні у більш нових версіях фреймворка. А також старий стиль коду необхідно оновлювати до нових стандартів для покращення розуміння розробниками.

Загалом, система ПАРКС.NET має переваги у вигляді підтримки паралельних обчислень, простоти використання та масштабованості. Однак, негаразди, такі як відсутність достатньої кількості асинхронних функцій та використання старої версії .NET Framework, можуть впливати на продуктивність та доступність деяких функцій у системі.

2.4 Проєктування архітектури та компонентів системи ПАРКС.NET

Під час проєктування майбутньої системи ПАРКС.NET, основою для розробки є наявна версія ПАРКС.NET, проте з урахуванням покращень та пристосування під .NET 7. Нова реалізація системи повинна забезпечувати паралельну обробку інформації, використовуючи оновлені функціональні можливості та переваги .NET 7.

Архітектура системи ПАРКС.NET повинна бути гнучкою та розширюваною, забезпечуючи зручну взаємодію між компонентами. Рекомендується використовувати модульну архітектуру, де кожен компонент відповідає за виконання певної функціональності та має чітко визначений інтерфейс взаємодії з іншими компонентами. Це сприятиме зрозумілості та легкості управління системою.

Одним із головних компонентів системи є точки. Кожна точка представляє потік, який виконує алгоритмічний модуль (АМ). Комунікація між точками здійснюється за допомогою програмних каналів (ПК), які можуть бути реалізовані у вигляді зв'язків між точками. Архітектура системи повинна

забезпечувати зручний спосіб встановлення та керування програмними каналами між точками.

При проектуванні системи, слід враховувати такі основні принципи:

- Масштабованість - система повинна бути здатною працювати з різними обсягами даних та кількістю точок. Потрібно передбачити механізми автоматичного масштабування та оптимізації роботи системи при збільшенні обсягів обробки.
- Ефективність - використання паралельних алгоритмів повинно сприяти покращенню продуктивності системи. Оптимізація виконання АМ та ефективна обробка комунікації між точками дозволить забезпечити швидкодію системи.
- Надійність - система повинна бути стійкою до відмов та забезпечувати відновлення роботи в разі непередбачуваних ситуацій. Впровадження механізмів контролю помилок, збереження стану та резервне копіювання даних допоможе підвищити надійність системи.
- Простота використання - інтерфейси та API системи повинні бути зрозумілими та зручними для розробників. Наявність документації, прикладів використання та підтримка сприятимуть швидкому старту роботи з системою.

З урахуванням цих переваг та недоліків, проектування майбутньої системи ПАРКС.NET на базі .NET 7 повинно спрямовуватись на створення гнучкої та ефективної архітектури, яка забезпечує масштабованість, ефективність, надійність та простоту використання.

2.5 Проектування бази даних для системи ПАРКС.NET

База даних для системи ПАРКС.NET грає важливу роль у зберіганні та управлінні даними, необхідними для оптимального функціонування системи. Вона може бути реалізована за допомогою різних типів баз даних, включаючи

реляційні (наприклад, SQL Server, MySQL, PostgreSQL) та нереляційні (наприклад, MongoDB, Redis). Структура бази даних включає різні таблиці або колекції, що дозволяють зберігати різні типи даних. Наприклад, можуть бути створені таблиці для зберігання інформації про задачі, вузли, користувачів та інші сутності, які використовуються в системі. Зв'язки між таблицями можуть бути встановлені за допомогою первинних та зовнішніх ключів для забезпечення цілісності даних.

Схема бази даних визначає структуру таблиць, поля, типи даних та зв'язки між ними. Вона може бути визначена за допомогою SQL-сценаріїв або міграційних скриптів. Схема включає опис структури таблиць, індексів, обмежень, тригерів та інших об'єктів бази даних.

У системі ПАРКС.NET можна використовувати різні моделі даних, залежно від потреб проекту. Реляційна модель може використовуватись для зберігання структурованих даних, де дані організовані в таблиці зі стовпцями і рядками. Нереляційна модель, така як документоорієнтована, може використовуватись для зберігання невизначених або змінних даних, де дані представлені у вигляді документів, колекцій ключ-значення тощо.

Доступ до даних в базі даних здійснюється за допомогою SQL-запитів, функцій чи методів, залежно від типу бази даних. SQL-запити використовуються для взаємодії з реляційними базами даних, де можна виконувати операції вибірки, вставки, оновлення та видалення даних. Нереляційні бази даних можуть мати свої власні механізми доступу до даних, зазвичай на основі запитів, команд або методів, специфічних для конкретної бази даних.

Безпека даних є важливим аспектом бази даних системи ПАРКС.NET. Механізми безпеки можуть включати аутентифікацію та авторизацію користувачів, рівні доступу до даних, шифрування та інші заходи для захисту даних від несанкціонованого доступу, модифікації та втрати.

Проектування бази даних для системи ПАРКС.NET включає в себе розробку схеми бази даних, визначення таблиць, полів і зв'язків між ними, а також вибір типів даних і визначення обмежень цілісності. Основна мета

проектування бази даних - забезпечити ефективне збереження, організацію та доступ до даних, що використовуються системою ПАРКС.NET.

При проектуванні бази даних для системи ПАРКС.NET рекомендується використовувати реляційну модель даних, яка дозволяє структуровано зберігати дані в таблицях зі зв'язками між ними. Кожна таблиця в базі даних представляє певний тип даних або сутність, пов'язану з функціональністю системи.

Наприклад, для системи ПАРКС.NET можуть бути створені такі таблиці:

- Таблиця "Користувачі" з полями, які описують особисту інформацію користувачів, такі як ім'я, прізвище, електронна пошта та пароль для аутентифікації.
- Таблиця "Завдання" з полями, які описують характеристики розподілених обчислювальних завдань, такі як ідентифікатор завдання, опис, статус виконання і результати.
- Таблиця "Точки" з полями, що містять інформацію про розподілені точки системи, таку як їхній ідентифікатор, IP-адреса, статус доступності тощо.

Для забезпечення ефективного доступу до даних можна використовувати індекси, які прискорюють пошук і фільтрацію даних. Наприклад, можна створити індекс на полі "ім'я" в таблиці "Користувачі", щоб забезпечити швидкий пошук користувачів за їхнім іменем.

Однак, при проектуванні бази даних для системи ПАРКС.NET потрібно враховувати певні фактори та виклики. Наприклад, зростаючий обсяг даних може вимагати оптимізації запитів і використання розподіленої бази даних для забезпечення масштабованості. Крім того, варто пам'ятати про безпеку даних та використання відповідних механізмів для захисту конфіденційної інформації.

Загалом, проєктування бази даних для системи ПАРКС.NET вимагає ретельного аналізу вимог до даних, визначення потреб системи та врахування найкращих практик в галузі баз даних. Правильно спроектована база даних допоможе забезпечити ефективну та надійну роботу системи ПАРКС.NET,

забезпечуючи зручний доступ до необхідної інформації та підтримку обчислювальних завдань.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ ПАРКС.NET

3.1 Архітектура системи ПАРКС.NET Core

3.1.1 Логічне розбиття на проєкти

Всі компоненти системи розбиваються на логічні проєкти в залежності від їхньої функціональності та ролі. Існує проєкт сервера, вузлів та алгоритмічного модуля. Кожен проєкт має свої вхідні та вихідні дані, а також взаємодіє з іншими проєктами для обміну даними та координації роботи. Логічне розбиття на проєкти дозволяє забезпечити модульність та складність системи, що полегшує розробку, тестування та підтримку.

3.1.2 Проєкт сервера

Проєкт сервера відповідає за керування та координацію роботи всіх компонентів системи. Він приймає запити від клієнтів, розподіляє завдання між вузлами та контролює виконання обчислень.

У системі проєкт створений під назвою HostServer. Він містить клас Server, який є основою логіки сервера ПАРКС. Цей клас є синглтоном (Singleton) і доступний через властивість Instance. Він використовує ConcurrentDictionary<int, JobInfo> для збереження інформації про завдання (задачі) і надає методи для додавання нового хоста, створення та видалення точок, початку, завершення та скасування завдань.

Проєкт також включає клас TCPHostServer, який відповідає за мережеву взаємодію з клієнтами. Він наслідує клас BackgroundService для запуску в фоновому режимі. В методі ExecuteAsync встановлюється прослуховувач TCP, приймаються підключення клієнтів і обробляються їхні запити. Цей клас також містить методи для обробки запитів клієнта, такі як створення та видалення точок, початок та завершення завдань, а також обробку відключення клієнта або хоста.

У проєкті наявні також такі важливі функції: зчитування хостів з файлу, вибір цільового хоста, отримання поточних завдань, перевірка статусу хостів та отримання найважливішого завдання.

3.1.3 Проєкт вузлів

Проєкт вузлів відповідає за виконання обчислень на конкретних вузлах. Кожен вузол має своє робоче навантаження і виконує обчислення згідно з отриманими від сервера завданнями.

Клас Daemon в цьому проєкті відповідає за реалізацію вузлів. Він успадковує клас BackgroundService, що дозволяє запускати його як фоновий сервіс. У класі Daemon є різні поля, які використовуються для зберігання TCP-прослуховувача, об'єкту блокування, словника для зберігання номерів точок і номерів завдань, а також інших допоміжних полів.

Основна функціональність класу Daemon реалізована у методі ExecuteAsync. У цьому методі відбувається налаштування та запуск TCP-прослуховувача, підключення до сервера, обробка клієнтських запитів та інші операції, необхідні для роботи вузла системи ПАРКС.

Клас Daemon також має додаткові методи, які виконують дії, такі як з'єднання з сервером, запуск прослуховувача, обробка клієнтських запитів тощо. Для роботи класу використовуються бібліотеки

Microsoft.Extensions.DependencyInjection, Microsoft.Extensions.Hosting, Serilog та ParcsCore.

3.1.4 Проєкт алгоритмічного модуля

Проєкт алгоритмічного модуля містить реалізацію паралельних алгоритмів, які використовуються для обробки даних та виконання завдань.

Клас MatrixesModule є реалізацією алгоритмічного модуля, який використовує ПАРКС.NET Core для паралельного обчислення множення матриць. Він успадковує клас MainModule з бібліотеки ParcsCore.

Головний метод класу - Run - виконує обчислення множення матриць за допомогою ПАРКС. Спочатку отримує необхідні дані, такі як імена файлів з матрицями та кількість точок, на які розбивається обчислення. Далі перевіряється правильність введених аргументів командного рядка.

Після підготовки необхідних даних, створюються точки та канали зв'язку між ними. Для кожної точки створюється канал, а потім виконується клас NewMatrixModule.MultMatrix в кожній точці.

Далі в залежності від кількості точок виконується обчислення множення матриць з використанням відповідного алгоритму розбиття матриць. Після отримання результатів, обчислені матриці об'єднуються за допомогою методів Join2, Join4 або Join8, в залежності від кількості точок. Нарешті, отриманий результат зберігається у файл.

3.1.4 Проєкт модуля ПАРКС

Проєкт ParcsCore є бібліотекою, яка надає функціональність для розподіленого обчислення в системі. Основними класами цього проєкту є:

- **Channel.cs** і **ConcurrentChannel.cs** - класи, які відповідають за комунікацію між вузлами (нодами). Вони дозволяють передавати повідомлення та дані між вузлами.
- **Point.cs** і **ConcurrentPoint.cs** - класи, які представляють вузли (точки). Вони використовуються для створення обчислювальних задач і взаємодії з іншими вузлами.
- **Constants.cs** - клас, який містить константи, використовувані у системі ПАРКС.NET Core.
- **HostInfo.cs** - клас, який представляє інформацію про хост (вузол). Він забезпечує з'єднання з хостом і отримання даних про його характеристики, такі як IP-адреса, порт, кількість процесорів тощо.
- **Job.cs** - клас, який представляє обчислювальну задачу. Він дозволяє створювати точки (вузли) для виконання обчислень та взаємодії з сервером системи ПАРКС.NET Core.
- **IModule.cs** і **IModuleOptions.cs** - інтерфейси, які використовуються для реалізації модулів (коду, який виконується на вузлах). Вони визначають методи і властивості, необхідні для виконання обчислень.
- **ParcsException.cs** - клас, який представляє виняток, пов'язаний з помилками.
- **TaskQueue.cs** - клас, який забезпечує чергу завдань для виконання на вузлах.

3.2 Конфігурація сервера та вузлів

На локальній машині було налаштовано з’єднання між сервером і вузлами системи ПАРКС.NET Core. Нижче наведено виконані кроки для налаштування цього з’єднання:

1. Встановлення IP-адрес: для кожного вузла була визначена унікальна IP-адреса на локальній машині. Це може бути зроблено через налаштування мережних параметрів або використання DHCP для автоматичного присвоєння IP-адрес.
2. Конфігурація сервера - в конфігураційному файлі сервера ПАРКС.NET Core були вказані IP-адреси кожного вузла для забезпечення правильного з’єднання, а також порти, які будуть використовуватись для комунікації з кожним вузлом.
3. Перевірка з’єднання - після налаштування IP-адрес та портів було запущено сервер ПАРКС.NET Core на локальній машині. Сервер намагався встановити з’єднання з кожним вузлом, використовуючи вказані IP-адреси та порти.

Було перевірено логи сервера для виявлення можливих проблем з підключенням до вузлів на локальній машині.

Таким чином, з’єднання між сервером і вузлами на локальній машині було успішно налаштовано, що дозволяє системі ПАРКС.NET Core спілкуватись та працювати з вузлами. На рисунку 5 показано результат з’єднання в консолі точки, де видно, що спочатку була запущена програма DaemonPr та введено локальну IP адресу цієї точки для зв’язку з сервером. Далі показано, що програма приймає з’єднання від клієнтів за IP адресою цієї точки, а вкінці показано, що з’єднання з сервером налаштовано.

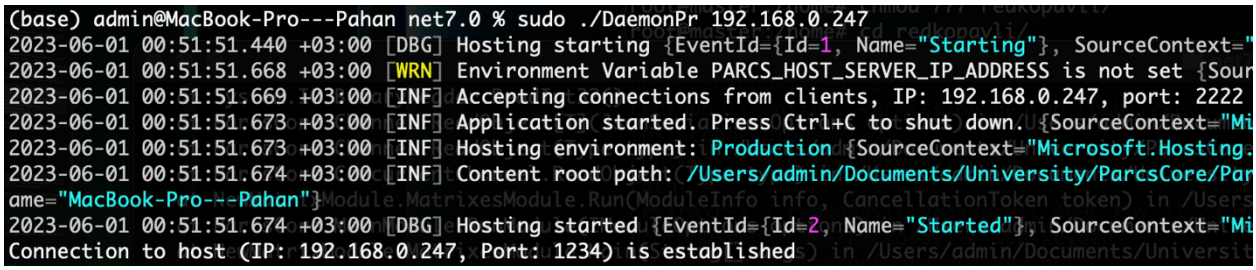

Рисунок 5 – Результат з’єднання в консолі точки

На рисунку 6 зображено консоль сервера. Спочатку була запущена програма HostServer та введено локальну IP цього сервера. Далі показано, що програма приймає з’єднання від клієнтів за IP адресою цього сервера, а вкінці показано, що з’єднання з точкою налаштовано та почалась нова робота.

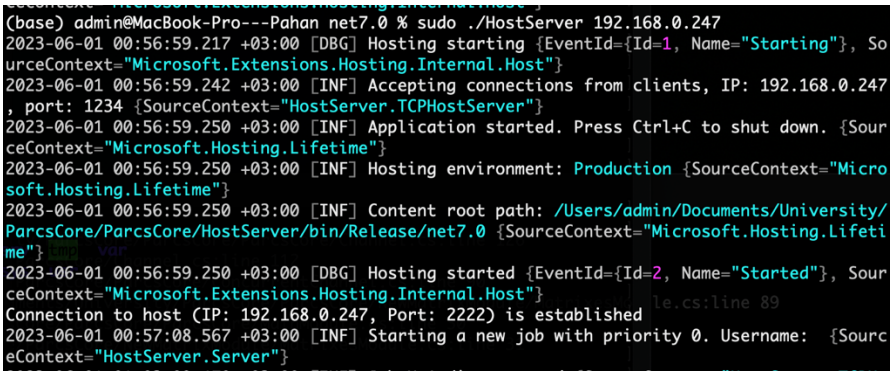

Рисунок 6 – Результат з’єднання в консолі сервера

3.3 Оптимізація та покращення коду

3.3.1 Виправлення стилю коду

У процесі оптимізації та покращення коду системи ПАРКС.NET Core було внесено такі зміни зі стилю коду:

- Форматування коду - було переглянуто всі файли з вихідним кодом і вирівняні відступи, використовуючи зручний і послідовний стиль. Було

забезпечено послідовність використання відступів, включаючи відповідну кількість пробілів або табуляцій для покращення читабельності.

- Іменування змінних та функцій - переглянуто імена змінних та функцій і внесено виправлення для забезпечення зрозумілості та послідовності. Було дотримано змінних стандартів іменування, таких як використання зрозумілих та описових назв змінних.
- Документування коду - було переглянуто коментарі та документацію коду і внесено необхідні зміни для забезпечення зрозумілості та актуальності інформації. Було додано та оновлено коментарі до функцій, щоб пояснити їхню функціональність та використання.

Таким чином, було виконано виправлення стилю коду, що призвело до поліпшення читабельності, структурованості та зрозумілості вихідного коду системи ПАРКС.NET Core.

3.3.2 Використання асинхронних функцій

Було переглянуто код та виділено блоки коду, які можуть працювати асинхронно. Було створено асинхронні методи, що дозволяють виконання довгих операцій без блокування головного потоку виконання. Використання ключового слова async дозволяє використовувати асинхронні операції, такі як асинхронні запити до бази даних або мережеві виклики, без блокування головного потоку виконання. Використання ключового слова await дозволяє головному потоку виконання чекати завершення асинхронної операції без блокування його роботи.

Розглянемо приклад: на рис. 4 вказано різницю в коді для синхронного та асинхронного методу. Метод WriteData реалізовує підтримку читання данних у _writer синхронно та очищення потоку після виконаної операції методом Flush. У новій реалізації метод WriteDataAsync використовує аналогічні два методи, проте вони працюють асинхронно.

```
public void WriteData(string data)
{
    _writer.Write(data);
    _writer.Flush();
}

public async void WriteDataAsync(string data)
{
    await _writer.WriteAsync(data);
    await _writer.FlushAsync();
}
```

Рисунок 7 - Демонстрація різниці асинхронного та синхронного методу

Таким чином, шляхом використання асинхронних функцій в коді системи ПАРКС.NET Core було досягнуто покращення продуктивності та швидкодії, особливо при роботі з довгочасними операціями, які не вимагають блокування головного потоку виконання.

3.3.3 Рефакторинг та оновлення коду

У процесі оптимізації та покращення коду системи ПАРКС.NET було виконано рефакторинг та оновлення коду для підвищення його якості, читабельності та підтримки. Рефакторинг - це процес вдосконалення внутрішньої структури коду, при якому зберігається зовнішня функціональність програми. Це означає переписування коду з метою поліпшення його якості, читабельності, ефективності та підтримки. Рефакторинг допомагає зменшити заборгованість технічного боргу, спрощує розробку, розуміння та модифікацію коду, а також підвищує його стабільність та переносимість.

Було виконано такі зміни з рефакторингу:

- Переглянуто вихідний код та видалено зайві фрагменти коду, які не використовуються або не мають необхідної функціональності. Це сприяє зниженню зайнятості пам'яті та спрощенню розуміння коду.

- Переглянуто код і розбито його на окремі модулі та функції відповідно до їх функціональності та відповідальності. Це сприяє поліпшенню читабельності коду та його модульності.
- Переглянуто алгоритми, що використовуються в системі, та внесено оптимізації для зниження обчислювального часу та збільшення швидкодії. Це може включати використання більш оптимальних алгоритмів, уникнення зайвих обчислень та вдосконалення структур даних.
- Виконано тестування оновленого коду для переконання в його працездатності та відсутності помилок. Це допомагає забезпечити непорушність функціональності системи та зберегти її стабільність.

Таким чином, за допомогою рефакторингу та оновлення коду системи ПАРКС.NET Core було досягнуто поліпшення якості, читабельності та швидкодії коду, що сприяє його непорушному функціонуванню та майбутній підтримці.

3.4 Тестування системи та підготовка віртуальних машин

3.4.1 Створення проєкту та віртуальних машин на GCP

Перш ніж почати тестування системи ПАРКС.NET Core на віртуальних машинах, потрібно створити проєкт і налаштувати віртуальні машини на платформі Google Cloud Platform (GCP). Ось кроки, які було виконано:

- Увійдено до облікового запису GCP на веб-порталі GCP.
- Створено новий проєкт, натиснувши кнопку "Створити проєкт"
- Надано проєкту підходяще ім'я і ідентифікатор проєкту (Project ID).
- Було вказано потрібні параметри віртуальної машини, такі як тип машини, регіон, образ операційної системи тощо. Для успішної роботи ПАРКС.NET Core було використано операційну систему Ubuntu Linux 20.04 та

налаштовано необхідні опції, такі як розмір диску, правила файрволу тощо. Було повторено останні кроки для створення інших екземплярів віртуальних машин

3.4.2 Підключення до віртуальних машин через SSH

Після успішного створення віртуальних машин на GCP, було підключено до них за допомогою протоколу SSH. Це дозволило виконувати команди і налаштування на віртуальних машинах. Ось кроки, які було виконано для підключення:

1. Відкрито термінал або командний рядок на локальному комп'ютері.
2. Введена наступна команда з використанням фактичної IP-адреси віртуальної машини, до якої необхідно підключитись: gcloud compute ssh master
3. Введено ім'я користувача та пароль для віртуальної машини. Ці дані можуть відрізнятися залежно від конфігурації віртуальної машини.

Після введення вірних облікових даних було підключено до віртуальної машини через SSH.

3.4.3 Встановлення .NET 7 runtime

Після успішного підключення до віртуальних машин через SSH необхідно встановити .NET 7 runtime для забезпечення правильної роботи системи ПАРКС.NET Core. Виконані кроки для встановлення:

- Оновлено пакети віртуальної машини командою: sudo apt update
- Встановлено пакети, необхідні для .NET 7 runtime, виконавши наступну команду: sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
- Додано репозиторій .NET виконавши наступні команди:

wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb

sudo dpkg -i packages-microsoft-prod.deb

sudo apt update

- Встановлено .NET 7 runtime виконавши наступну команду: sudo apt install -y dotnet-runtime-7.0
- Після завершення встановлення було перевірено версію .NET 7 runtime командою: dotnet --version та підтверджено, що встановлено .NET 7 runtime.

3.4.4 Запуск вузлів та сервера

Після успішного встановлення .NET 7 runtime можна запускати вузли та сервер системи ПАРКС.NET Core. Для запуску було виконано наступні кроки:

1. Перейдено до каталогу з встановленою системою ПАРКС.NET Core. з терміналу та виконано команду для запуску сервера: dotnet HostServer. Ця команда запустила сервер.
2. Відкрито ще 4 термінали для запуску кожного вузла та виконано на кожному команду: dotnet Daemon, яка запустила вузли.
3. Використано допоміжний файл server.txt, в якому вказано IP адресу серверу, щоб вузли знали, до якого серверу під’єднуватись.

Після запуску, вузли під’єднались до серверу та були готові до тестування та використання.

3.5 Виконання алгоритмічного модуля та розподіл завдань

3.5.1 Розподіл робіт між вузлами

Для успішного виконання алгоритмічного модуля системи ПАРКС.NET Core було здійснено розподіл завдань між вузлами. Кожен вузол отримав свою частину роботи для обробки. Нижче наведено опис виконаних кроків:

- Визначено кількість доступних вузлів, що брали участь у виконанні алгоритмічного модуля.
- Розподілено завдання між вузлами відповідно до алгоритму розподілу, який був визначений у системі ПАРКС.NET Core.
- Кожен вузол отримав своє завдання для обробки, яке включало необхідні дані та параметри.

3.5.2 Зв'язок між алгоритмічним модулем та вузлами

Після розподілу завдань між вузлами системи ПАРКС.NET Core було встановлено зв'язок між алгоритмічним модулем та вузлами для передачі даних та комунікації. Нижче наведено виконані кроки для забезпечення зв'язку:

- В алгоритмічному модулі були використані відповідні функції та бібліотеки для встановлення з'єднання з кожним вузлом.
- Кожен вузол був ідентифікований унікальним ідентифікатором або адресою, яку використовував алгоритмічний модуль для звернення до нього.
- Дані та команди були передані між алгоритмічним модулем та вузлами через встановлене з'єднання.

3.5.3 Обчислення результатів та їх складання

Після виконання завдань на кожному вузлі системи ПАРКС.NET Core було здійснено обчислення результатів та їх подальше складання. Нижче наведено опис виконаних кроків:

- Кожен вузол виконав свою частину роботи та отримав результати обчислень.
- Результати, отримані на кожному вузлі, були зібрані та збережені для подальшого аналізу та використання.
- Залежно від потреб алгоритмічного модуля, результати можуть бути об'єднані, оброблені або використані окремо.

Це описує процес виконання алгоритмічного модуля та розподілу завдань між вузлами, зв'язок між алгоритмічним модулем та вузлами, а також обчислення результатів та їх складання в системі ПАРКС.NET Core.

3.6 Тестування та порівняння зі старою системою ПАРКС.NET

3.6.1 Підготовка тестових сценаріїв

Для порівняння нової системи ПАРКС.NET Core зі старою системою були підготовлені тестові сценарії. Нижче наведено опис проведення цього етапу:

- Були визначені різні тестові сценарії, які включали різні розміри матриць та кількість вузлів (точок).
- Для кожного сценарію були підготовлені вхідні дані, які використовувалися для обчислення на обох системах.
- Було встановлено налаштування тестового середовища, включаючи запуск вузлів та налаштування зв'язку між ними.

3.6.2 Вимірювання часу виконання

Для порівняння швидкості виконання нової системи ПАРКС .NET зі старою системою були проведені вимірювання часу виконання для кожного тестового сценарію. Нижче наведено опис проведення цього етапу:

- Кожен тестовий сценарій був запущений на обох системах, використовуючи однакові вхідні дані та налаштування.
- Було виміряно час виконання для кожного сценарію на обох системах. Результати були записані для подальшого аналізу.
- Кількість вузлів (точок) та розмір матриць були варійовані, щоб оцінити вплив цих факторів на час виконання обчислень.

3.6.3 Результати порівняння та швидкодія нової системи

Після вимірювання часу виконання для різних тестових сценаріїв на обох системах були отримані результати порівняння та визначена швидкодія нової системи ПАРКС.NET Core.

Були створені таблиці порівняння, які відображають час виконання обчислень на новій та старій системах для різної кількості вузлів (точок) та розмірів матриць.

Крім того, було проведено аналіз впливу кількості вузлів (точок) та розміру матриць на швидкодію системи, що дозволило виявити оптимальні параметри для досягнення максимальної ефективності обчислень.

У таблиці 1 вказано результати обчислень використовуючи стару систему ПАРКС.NET, а в таблиці 2, відповідно, результати обчислень використовуючи нову систему ПАРКС.NET Core. Виконавши порівняння таблиць, можна відзначити зростання швидкості в середньому на 43% у порівнянні зі старою системою на великих розмірах матриць. На менших розмірах матриць різниця не настільки велика. Це може бути зумовлено тим, що покращення системи .NET 7

включає оптимізацію роботи з великими обсягами даних, а також більше використання ресурсів таких як CPU, пам’ять та кеш для ефективної обробки даних.

Таблиця 1 – Час виконання обчислень в секундах на старій системі ПАРКС.NET

Розмір матриці	**1 точка**	**2 точки**	**4 точки**	**8 точок**
100 x 100	11.73	9.54	8.63	7.08
1000 x 1000	23.76	15.89	13.42	10.71
2000 x 2000	94.51	63.27	53.58	32.84
3000 x 3000	212.34	141.88	120.10	76.06

Таблиця 2 – Час виконання обчислень в секундах на новій системі ПАРКС.NET

Розмір матриці	**1 точка**	**2 точки**	**4 точки**	**8 точок**
100 x 100	9.43	7.14	6.68	5.52
1000 x 1000	13.46	9.00	7.61	6.07
2000 x 2000	53.68	35.85	30.34	19.27
3000 x 3000	120.77	80.62	68.24	43.62

ВИСНОВКИ

У рамках даної дипломної роботи було проведено дослідження та розроблено нову систему для паралельних обчислень ПАРКС.NET Core. Було проаналізовано вже існуючі рішення і аналоги та наведено їх переваги та недоліки. Основний недолік існуючих рішень це стара версія ядра системи, що ускладнює оновлюваність та використання на нових пристроях. Було спроєктовано архітектуру нової системи, та розроблено програму, яка вдосконалює стару систему та виправляє її недоліки. Під час розробки було проведено виправлення стилю коду, використано асинхронні функції, де було необхідно та була можливість, проведено рефакторинг та оновлення. Було протестовано систему на хостингу Google Cloud Platform та отримано результати, які свідчать про високу швидкодію нової системи порівняно зі старою системою ПАРКС.NET. Експериментальні дані демонструють, що нова система забезпечує виконання обчислень приблизно на 43% швидше, що свідчить про успішне використання технології ПАРКС.NET Core та відповідність розробленої системи сучасному рівню технічних знань та технологій.

Подальше використання нової системи може бути успішним у різних галузях та сферах, де вимагаються паралельні обчислення. Застосування цієї системи сприятиме покращенню швидкодії обчислень, що є важливим в контексті високонавантажених систем, наукових досліджень, обробки даних та інших сфер.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. G. Baptista, F. Abbruzzese, “Software Architecture with C# 10 and .NET 6”, .NET, vol. 8, pp. 145-148, 2022.

2. J. Richter, “CLR via C# (Developer Reference)”, Common Language Runtime, pp. 20-24, 2012

3. Федорус О.М. ПАРКС як засіб реалізації розподілених хмарних обчислень. // Штучний інтелект, 2016, № 2, С. 70-75.

4. А.В. Анісімов, О.В. Деревянченко, А.Ю. Хавро Застосування системи ПАРКС в docker контейнерах та Google Cloud Platform, 2018, С. 53-58.

5. Microsoft. C# https://learn.microsoft.com/en-us/dotnet/csharp/ 2023

6. D. Rubinstein, M. L. Tushman, "Technological Discontinuities, Organizational Capabilities, and Strategic Commitments," Administrative Science Quarterly, vol. 40, no. 3, pp. 439-465, 1995.

7. S. P. Bradley, U. M. Fayyad, C. Reina, "Scaling Clustering Algorithms to Large Databases," Knowledge Discovery and Data Mining, pp. 9-15, 1998.

8. R. J. Blakeley, "Parallel Computing: State-of-the-Art and Future Directions," Communications of the ACM, vol. 31, no. 8, pp. 880-897, 1988.

9. S. S. Yau, "A Comparative Study of Parallel Sorting Algorithms on the Transputer System," Parallel Computing, vol. 13, no. 4, pp. 365-384, 1989.

10. J. L. Hennessy, D. A. Patterson, "Computer Architecture: A Quantitative Approach," 6th ed., Morgan Kaufmann, 2017.

11. Андрій Хавро Parcs.NET https://github.com/AndriyKhavro/Parcs.NET 2016

12. J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Communications of the ACM, vol. 51, no. 1, pp. 107-113, 2008.

13. T. C. Huang, "Fault-Tolerant Parallel Computing," Computer, vol. 21, no. 2, pp. 48-59, 1988.

14. J. L. Hennessy, D. A. Patterson, "Computer Organization and Design: The Hardware/Software Interface," 5th ed., Morgan Kaufmann, 2013.

15. R. Buyya, M. Pathan, "Grid and Cloud Computing: Concepts and Practical Applications," CRC Press, 2010.

16. A. Y. Zomaya, "Parallel and Distributed Computing Handbook," CRC Press, 2018.

ДОДАТОК А

Код програмної реалізації:

https://github.com/pashokred/ParcsCore