Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів обробки даних у високонавантажених системах
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗзм-22

Свиридова В.В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Назаров О. С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Свиридовій Віолетті Віталіївні
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження методів обробки даних у високонавантажених системах»

2. Термін подання студентом роботи до екзаменаційної комісії 21.06.2024

3. Вихідні дані до роботи ОС Ubuntu 16.04, середовище розробки Intellij IDEA, локальний сервер, мобільний додаток під керуванням платформи Flutter

4. Перелік питань, що потрібно опрацювати в роботі
асинхронне програмування, біг дата, високонавантажені системи, ефективність системи, mapreduce, обробка даних, паралелізм, реактивне програмування, apache cassandra, apache spark.

КАЛЕНДАРНИЙ ПЛАН

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Видача завдання	29.04.2024	виконано
2	Аналіз предметної галузі	01.05.2024	виконано
3	Постановка задачі	02.05.2024	виконано
4	Експериментальні дослідження	02.05 – 20.05.24	виконано
5	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.05 – 22.05.24	виконано
6	Написання та оформлення статті та тез доповіді	20.05 – 23.05.24	виконано
7	Підготовка пояснювальної записки	01.05 – 26.05.24	виконано
8	Підготовка презентації та доповіді	26.05 – 2.05.24	виконано
9	Нормоконтроль	07.06 – 11.06.24	виконано
10	Рецензування	11.06 – 14.06.24	виконано
11	Занесення диплома в електронний архів	15.06.2024	виконано
12	Попередній захист	15.06.2024	виконано
13	Допуск до захисту у зав. кафедри	18.06.2024	виконано

Дата видачі завдання «29» квітня 2024 р.

Студент _____ Свиридова В.В.

(підпис)

Керівник роботи_____ Назаров О.С.

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить 65 сторінок, 26 рисунків, 10 джерел.

АСИНХРОННЕ ПРОГРАМУВАННЯ, БІГ ДАТА, ВИСОКОНАВАНТАЖЕНІ СИСТЕМИ, ЕФЕКТИВНІСТЬ СИСТЕМИ, MAPREDUCE, ОБРОБКА ДАНИХ, ПАРАЛЕЛІЗМ, РЕАКТИВНЕ ПРОГРАМУВАННЯ, APACHE CASSANDRA, APACHE SPARK.

Об'єктом дослідження даної работи є різноманітні методи та стратегії обробки даних, зокрема в контексті високонавантажених систем. Дослідження охоплює аналіз та порівняння передових методів утилізації середовищ виконання з підтримкою паралелізму, зокрема методу асинхронного та реактивного програмування.

Основною метою є визначення оптимального підходу для обробки великих обсягів даних в системах високого навантаження, зокрема в контексті систем пошуку інформації.

Метод рішення передбачає вивчення та оцінку інших підходів, зокрема тих, що входять до галузі Big Data, таких як MapReduce, Apache Spark, та Apache Cassandra. Визначення переваг та недоліків цих методів допоможе здійснити збалансований вибір оптимального інструментарію для реалізації системи обробки даних у високонавантажених умовах.

ASYNCHRONOUS PROGRAMMING, BIG DATE, HIGHLY LOADED SYSTEMS, SYSTEM EFFICIENCY, MAPREDUCE, DATA PROCESSING, PARALLELISM, REACTIVE PROGRAMMING, APACHE CASSANDRA, APACHE SPARK.

The object of research of this course is various methods and strategies of data processing, in particular in the context of highly loaded systems. The study covers the analysis and comparison of advanced methods of utilization of execution environments

with support for parallelism, in particular the method of asynchronous and reactive programming.

The main goal is to determine the optimal approach for processing large volumes of data in high-load systems, in particular in the context of information retrieval systems.

The solution method involves studying and evaluating other approaches, including those in the Big Data field, such as MapReduce, Apache Spark, and Apache Cassandra. Determining the advantages and disadvantages of these methods will help to make a balanced choice of optimal tools for implementing a data processing system in highly loaded conditions.

Я, Свиридова Віолетта Віталіївна, студент(ка) гр. ІПЗмз-22, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів обробки даних у високонавантажених системах», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 8
1 Аналіз предметної галузі 9
1.1 Аналіз предметної області 9
1.2 Актуальні підходи та методи для створення високонавантаженої системи пошуку 12
1.3 Постановка задачі дослідження 14
2 Аналіз та застосування big data 16
2.1 Поняття big data 16
2.2 Генезіс big data 16
2.3 Особливості big data 17
2.3.1 Перша характеристика 3v моделі 18
2.3.2 Друга характеристика 3v моделі 20
2.3.3 Третя характеристика 3v моделі 21
2.4 Інструменти та методи big data 23
3 Аналіз та вибір середовища для виконання програмного коду 28
3.1 Аналіз та огляд CLR 28
3.2 Аналіз та огляд JVM 30
3.3 Порівняння CLR та JVM та обґрунтування подальшого вибору 30
3.4 Докладний аналіз JVM та її складових з наступним визначенням способів контекстної оптимізації 33
4 Порівняння методів та конфігурацій у високонавантажених системах для пошуку інформації з практичної точки зору 39
4.1 Опис оточуючого середовища системи 39
4.2 Виконання та оцінка результатів використання стандартної моделі для прийому та обробки запитів 39
4.3 Підготовка, впровадження та аналіз результатів реактивної моделі отримання та обробки запитів 44
4.4 Порівняння отриманих результатів 48
Висновки 52

Перелік джерел посилання 53
Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 55
Додаток А Звіт результатів перевірки на унікальність тексту в базі хнуре..... 56
Додаток Б Слайди презентації 57
Додаток В Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам дсту 3008: 2015..... 65

ВСТУП

Вперше здійснена подача інформації у великих обсягах створила потребу у високоефективних системах обробки даних, які здатні працювати з великими об'ємами інформації при високому навантаженні. У світі сучасних технологій високонавантажені системи стають все більш поширеними, адже компанії та організації прагнуть оптимізувати роботу своїх сервісів та забезпечити ефективне управління даними.

Метою даної роботи є дослідження різних підходів до обробки даних у високонавантажених системах. Зростання обсягу інформації та необхідність її швидкої та ефективної обробки вимагають ретельного вивчення інструментів та методів, призначених для роботи з великими потоками даних. Дослідження цієї теми важливо для розуміння та впровадження найбільш оптимальних стратегій обробки даних у високонавантажених умовах.

У цьому дослідженні буде розглянуто різноманітні підходи до обробки даних у високонавантажених системах, зокрема враховано технології обробки стріму, паралельну обробку, а також використання інноваційних інструментів, спрямованих на оптимізацію роботи з великими обсягами інформації. Передбачається проведення аналізу переваг та недоліків різних підходів з метою визначення їхньої ефективності у конкретних умовах високого навантаження.

Це дослідження спрямована на вивчення сучасних тенденцій у сфері обробки даних та надає можливість глибше розуміти виклики, з якими стикаються високонавантажені системи, а також знаходити оптимальні рішення для їхньої ефективної роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної області

Типовий веб-проект включає три основні компоненти, які разом утворюють ефективну архітектуру програмного забезпечення. Перший компонент - це клієнтський додаток, що може бути представлений веб-додатком для браузера або мобільним додатком. Він виступає інтерфейсом для користувача та забезпечує зручний доступ до функціоналу системи.

Другий компонент – це веб-сервер, який може бути представлений одним чи декількома серверами з подальшою їхньою реплікацією. Веб-сервер відповідає за центральну реалізацію бізнес-логіки системи, використовуючи фреймворк загального або специфічного призначення, наприклад, фреймворки для електронної комерції, такі як SAP Hybris або Magento.

Третій компонент – це сервер даних, який, за потреби, може бути реплікованим SQL базою даних. Цей сервер відповідає за зберігання та управління даними, необхідними для функціонування системи.

Такий підхід представляє собою типову трирівневу архітектуру програмного забезпечення, яка зустрічається в більшості веб-проектів (рис. 1.1).

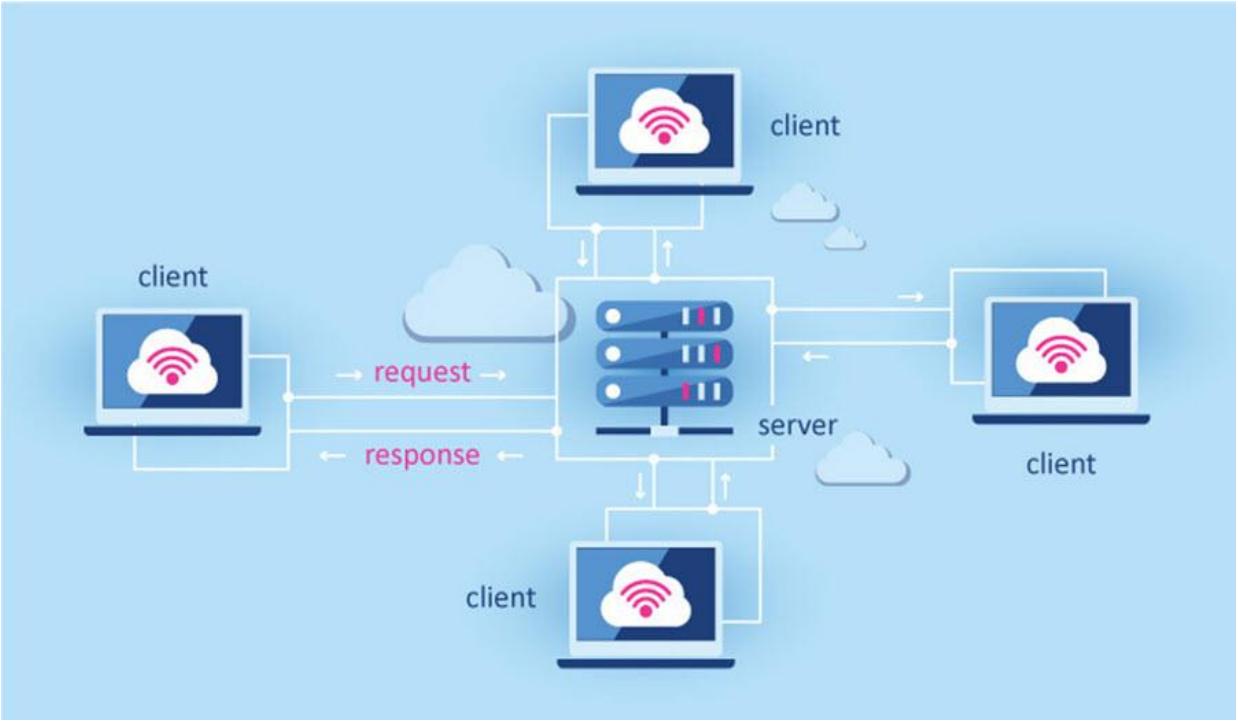

Рисунок 1.1 – Трирівнева архітектура (за даними [1])

Ця архітектура визнана досить ефективною і займає більше 92% від усіх веб-проектів. Її популярність пояснюється не лише розвитком та підтримкою веб-співтовариства та open-source проектів, але й легкістю пошуку та найму необхідних спеціалістів, а також навчання інженерів із різними профілями для підтримки проектів.

У багатьох випадках, особливо в тих, де системи стикаються з важкими обчисленнями чи обробкою значної кількості даних, використання підходу, при якому для кожного з’єднання з сервером створюється відповідний потік виконання, стає об'єктом ретельного розгляду.

Хоча цей метод може бути ефективним для великої кількості паралельних з’єднань (N), що призводить до створення в системі відповідної кількості потоків (M, де $M \geq N$), проте у випадках, коли виникають завдання, пов'язані з важкими обчисленнями або обробкою великого обсягу даних, цей підхід може винести сумнів щодо його ефективності та рентабельності.

Операційні системи мають обмеження на кількість можливих потоків, що можуть бути створені, і сами процесори обмежені в кількості паралельних потоків (зазвичай до 24). Інше можливе рішення - реплікація серверів API та СУБД, хоча збільшує вартість підтримки системи за алгебраїчною прогресією, не завдає прискорення обчислень або обробки даних відповідно до величини збільшення.

Крім того, існують ризики, пов'язані з некоректним розподілом обчислень між різними інстансами, що вимагає конфігурації реверс-проксі, оркестратора сервісів, а також оптимізації та конфігурації самого реплікованого чи розподіленого серверу СУБД. В результаті витрати на підтримку та невеликий приріст ефективності з масштабуванням системи можуть призвести до невеликої вигоди від такого підходу.

Для вирішення вищезазначених проблем і підвищення рентабельності системи можна використовувати нові моделі паралельної обробки, які дозволяють ефективніше використовувати ресурси та оптимізувати роботу системи. Серед таких моделей можна виділити асинхронну обробку запитів та даних, модель "реактор", реактивну модель, модель, засновану на акторах, та інші.

Ці нові підходи дозволяють значно поліпшити ефективність системи відносно її функціональності та завдань. У порівнянні з традиційним блокуючим підходом, де кожен потік виконання призупиняє свою роботу у випадку, наприклад, очікування відповіді від бази даних, зазначені моделі дозволяють частково або повністю уникнути зупинок у роботі потоків виконання.

Одним із прикладів застосування такого підходу є використання асинхронної моделі обробки запитів з використанням асинхронних бібліотек вводу-виводу. Це значно зменшує навантаження на сервер (див. рис. 1.2) і сприяє ефективнішому використанню ресурсів системи.

Рисунок 1.2 – Цикл обробки подій (за даними [2])

Такий підхід може бути більш рентабельним та адаптованим до потреб системи, забезпечуючи покращену продуктивність та оптимізацію її функціоналу.

В умовах, коли обчислення стають важким завданням, а саме вимагають значної тривалості чи інтенсивності щодо використаної процесорної потужності, ефективною альтернативою є створення кластерів для обчислень. Це представляє собою додатковий фізичний рівень (tier) в рамках вже описаної архітектури. У цьому випадку рівень бізнес-логіки, який раніше був представлений одним фізичним серверним рівнем, розділяється на різні фізичні рівні для серверів, що відповідають завданням вводу-виводу та обчислень.

Цей підхід відкриває можливості для розробки спеціалізованих інструментів, які спрямовані на підвищення ефективності систем, створених за такою архітектурою.

Проекти з підвищеними вимогами до часу обчислень, розміру даних та інших параметрів обирають методи, архітектурні підходи та інструменти, які відповідають їхнім потребам у рентабельності. Такі методології та підходи, як реактивне програмування, кластерні обчислення, розподілені бази даних, NoSQL-довкілля, а також специфічні конфігурації програмних середовищ (наприклад, вибір конкретних реалізацій Garbage Collection або самої JVM у JVM-довкіллях), є більш рентабельними при вирішенні таких завдань. Водночас вони можуть бути менш рентабельними чи ефективними при вирішенні завдань більш типових категорій.

1.2 Актуальні підходи та методи для створення високонавантаженої системи пошуку

У контексті задачі пошуку спеціалізованої інформації, критичними аспектами є:

- кількість потоків виконання: визначає, скільки запитів система може обробляти одночасно. Оптимальне використання потоків може покращити швидкість відповіді на запити;
- ефективність використання оперативної пам'яті: важлива для оптимізації роботи системи, оскільки великі об'єми даних можуть призвести до великого споживання пам'яті і вплинути на продуктивність;
- кількість запитів, опрацьовуваних в єдиний момент часу: цей аспект визначає, наскільки ефективно система обробляє багато запитів одночасно, що є ключовим для високонавантажених середовищ;
- швидкість обробки даних: включає в себе аналіз результатів на предмет релевантності, фільтрацію, індексацію та інші операції, які визначають якість та швидкість відповідей на запити.

Пошукова система, яка має доступ до величезної кількості файлів різних об'ємів, повинна бути здатна ефективно масштабуватися та розподіляти навантаження горизонтально. Забезпечення підтримки горизонтального масштабування важливо, оскільки обсяг даних та запитів може швидко зростати, і система повинна бути готова до збільшення ресурсів для ефективної роботи.

Швидкість обробки запитів в пошуковій системі залежить від кількох ключових факторів:

- процесорна потужність: визначає, наскільки швидко система може виконувати обчислення. Потужний процесор може значно поліпшити продуктивність;
- кількість ядер процесора: багатоядерний процесор може паралельно обробляти декілька завдань, що важливо для високонавантажених систем та багатопоточних додатків;
- кількість рівнів кешу процесора та їх обсяг: кеш-пам'ять забезпечує швидкий доступ до даних, що може суттєво підвищити швидкість обробки;
- кількість потоків, що працюють водночас: можливість обробки кількох потоків одночасно може поліпшити паралельні обчислення та реакцію на багато запитів;
- обсяг оперативної пам'яті: достатній обсяг RAM дозволяє утримувати в пам'яті великі об'єми даних, що важливо для ефективної обробки запитів;
- частота як процесору, так і чипів ОЗУ: визначає швидкість, з якою пристрої можуть виконувати обчислення та доступ до пам'яті.

Кількість одночасно опрацьованих запитів також залежить від загальної потужності та завантаженості системи в конкретний момент часу. Фізично допустима кількість підключень на сервері часто визначається архітектурою системи та її здатністю обробляти паралельні запити, що може впливати на ефективність обробки великої кількості запитів одночасно.

Ефективність використання обчислювальних ресурсів залежить від ряду чинників, включаючи низькорівневу реалізацію мови програмування, програмне

середовище, цільову платформу, конфігурацію платформи, обрані програмні алгоритми та саму реалізацію програмного додатку. Зазначено, що останній фактор має значно більший вплив, ніж інші. Популярні мови програмування та платформи можуть показувати різні результати в бенчмарках з обчислень та системи вводу-виводу. Однак існують приклади високонавантажених систем, які, незважаючи на менш ефективні платформи, є ефективними вирішенням завдань.

Важливим фактором є конфігурація платформи, а не лише її вибір. Важливо враховувати, що платформи зі схожим набором можливостей можуть вести себе по-різному, наприклад, однопотокове середовище JavaScript або Dart порівняно з багатопоточним середовищем, таким як JVM або Erlang VM.

Кількість одночасно оброблюваних запитів, ефективність апаратного забезпечення та швидкість обробки даних залежать від горизонтального масштабування, вибору платформи та використання методів та підходів під час реалізації, а також конфігурації платформи. Реактивне програмування представляє підхід до ефективного використання апаратних ресурсів для обробки постійних джерел даних. Наприклад, вибір JVM, мови програмування, що компілюється у Java байткод, компілятора, серверного ПО та інших параметрів платформи впливає на ефективність системи.

Категорії проблем, такі як швидкість операцій вводу-виводу, обробки даних та управління дисковим простором, є предметом досліджень в галузі Big Data, що може надати методи та інструменти для вирішення вказаних в контексті проблем.

1.3 Постановка задачі дослідження

Дослідження спрямоване на вивчення пошукової системи на етапі реалізації. Основною метою є розгляд методів, інструментів та програмних компонентів середовища програмного забезпечення, спрямованих на вирішення проблем, що є характерними для високонавантажених систем. Серед цих проблем важливі аспекти, такі як оптимізація об’єму використовуваної пам’яті, ефективність утилізації паралельного (конкурентного) середовища, швидкість обробки даних та ефективність управління дисковими даними.

Вибір теми дослідження обумовлений тим, що зазначений клас проблем стає все більш поширеним у розробці веб-систем, але існуючі традиційні методи не завжди ефективні для їх вирішення. Актуальність дослідження обумовлена постійним зростанням цих проблем у сучасному програмуванні.

З метою вирішення поставлених завдань необхідно провести аналіз основних категорій методів та інструментів обробки даних у високонавантажених середовищах. Доцільно розглянути методи та інструменти, що використовуються в галузі Big Data, а також нові методи та моделі щодо утилізації паралельного середовища.

Після детального аналізу категорій методів та інструментів слід вибрати найбільш підходящі серед них. Далі важливо проаналізувати обрані методи та інструменти, а після цього визначити ті, які будуть порівняні зі стандартною моделлю роботи веб-серверів.

Для отримання висновків щодо відносної ефективності вибраних методів має бути проведений ряд експериментів в однакових умовах програмного середовища та системи. Результати експериментів повинні бути формалізовані і порівняні між собою для отримання об'єктивних висновків.

2 АНАЛІЗ ТА ЗАСТОСУВАННЯ BIG DATA

2.1 Поняття Big Data

Термін "Big Data" вказує на галузь, що зазвичай асоціюється з отриманням, обробкою та управлінням даними, обсяг яких виходить за межі умовного середнього значення. Визначається, що термін стосується наборів даних, розмір яких перевищує можливості звичайних баз даних для внесення, зберігання, управління та аналізу інформації. Світові репозиторії даних продовжують зростати з кожною хвилиною.

Проте, термін "Big Data" включає не лише обробку великих за середнім значенням обсягів даних, але й створення та реплікацію величезних обсягів даних в форматі, що відрізняється від традиційних форматів обробки даних, наприклад, веб-журнали, відеозаписи, текстові документи, машинний код, геопросторові дані тощо. Це може призводити до відсутності відповідних інструментів для вирішення поставлених завдань, таких як установка зв'язків між частинами даних і аналіз для отримання корисної інформації. Динамічна природа даних, особливо в високонавантажених системах, додає складності, оскільки традиційні методи і інструменти не завжди можуть відповідати такому темпу. Таким чином, галузь "Big Data" спрямована на роботу з великими обсягами даних різного формату та складу, що мають динамічну природу і використовуються з різних джерел з метою збільшення ефективності, створення нових продуктів, поліпшення існуючих та підвищення конкурентоспроможності.

2.2 Генезіс Big Data

У кіберпросторі існує безліч джерел, обробка яких може перевертати звичайний підхід і засоби для роботи з цими даними. Такі джерела включають в себе неперервний потік даних з різноманітних сенсорів, потоки інформації з різних веб-сервісів (новинні стрічки, повідомлення та події в соціальних мережах), потоки геолокаційних даних від GPS-модулів, а також централізовані сховища даних, пов'язаних із конкретною сферою (наприклад, реєстр фізичних осіб, зареєстрованих як ФОП), тощо.

Глобальний охоплення та поширення таких технологій стали ключовим фактором для впровадження концепції Big Data у практично всі сфери людського життя, де наукова діяльність, комерційна діяльність і державне управління визначаються як основні пріоритети. З плином часу характер даних значно змінився, відходячи від своєї початкової статичної та застарілої природи, де вони втрачали цінність після досягнення певної цільової події (наприклад, вартості квитка на концерт). Замість цього вони стали сировиною для виявлення нових взаємозв'язків та формування нових гіпотез, які можуть бути ефективними з економічної точки зору.

Це вказує на те, що їх можна використовувати для вирішення суміжних чи навіть абсолютно не пов'язаних завдань, де вони виступають сировиною для аналітичних процесів іншого типу, роду чи категорії, спрямованих вже на розв'язання завдань відповідної сфери.

Галузь внесла найбільший внесок у розвиток так званого прогнозування, де ці прогнози будуються на основі значно розширеного обсягу даних, що зазвичай перевищує традиційно прийняті обсяги. Один із типових прикладів цього підходу – прогнозування курсів валют, передбачення товарів, які часто супроводжують один одного при покупці, або навіть прогноз потенційних злочинів на основі аналізу виявлених патернів поведінки.

Все це служить основою для такої галузі комп'ютерних наук, як штучний інтелект. При розробці подібних прогнозів використовуються технології та інструменти, засновані на використанні різноманітних математичних прийомів, які операціонують на великих обсягах даних. Надійність та ефективність таких систем ґрунтуються на наданні, а також регулярному оновленні, різноманітних даних, які необхідні для калібрування алгоритмів і створення прогнозів.

2.3 Особливості Big Data

BigData зазвичай характеризується основними параметрами, а саме: обсягом, швидкістю і різноманіттям (див. рис. 2.1).

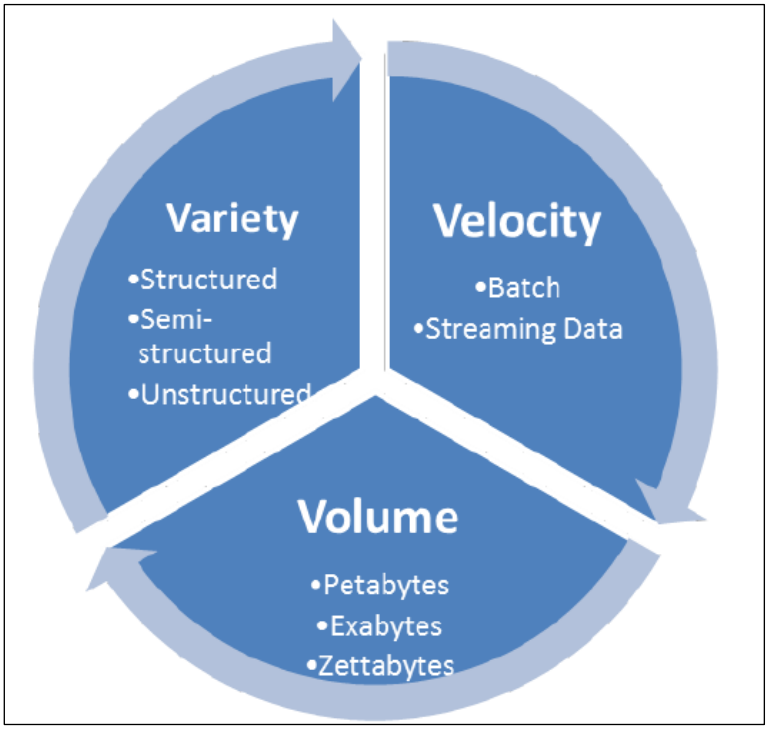

Рисунок 2.1 – Big data 3v модель (за даними[3])

2.3.1 Перша характеристика 3v моделі

Перша характеристика – це обсяг, який визначає кількість та розмір сформованих і збережених даних. Ці значення визначають значимість і можливе використання даних, а також визначають, чи може репозиторій вважатися BigData.

Обсяг часто є основною асоціацією з BigData, оскільки може бути дуже великим. Мова йде про розмір і/або кількість даних, що можуть перевищувати можливості навіть спеціалізованих систем.

У сучасному світі спостерігається експоненційний ріст обсягу даних, оскільки дані стають все більше не тільки текстовими. Інформацію тепер часто зберігають у формі відео, музики та великих зображень у соціальних мережах. Системи зберігання для підприємств часто використовують терабайти та петабайти. З цим зростанням розмірів баз даних додатки і архітектура, призначені для їхньої підтримки, також повинні адаптуватися та переглядати вже існуючі рішення. Таким чином, великі обсяги даних справедливо вважаються частиною BigData.

Наприклад, у зв'язку з мережею Facebook необхідно зберігати фотографії, які відправляють користувачі. Важливо враховувати, що кількість користувачів Facebook перевищує чисельність населення Китаю. Кожен з цих користувачів може відправити необмежену кількість зображень. У загальному обсягу, Facebook зберігає приблизно 250 мільярдів зображень.

У контексті BigData характеристика обсягу визначає розмір подібних даних. З розвитком галузі обсяги даних в спеціалізованих системах продовжать зростати.

Якщо зосередитися на галузі Інтернету речей, де прогрес в галузі BigData є ключовим фактором для власного розвитку, можна розглянути приклад зі збору різних показників за допомогою сенсорів або лічильників, що підключені до єдиної мережі. Усі отримані даними від сенсорів будуть продовжувати зростати, поступово наповнюючи репозиторій новою інформацією.

Питання в цьому випадку стосується темпу надходження даних і їх кінцевого обсягу. За оцінками Gartner, Cisco і Intel, кількість підключених пристроїв у середньому в середовищі Інтернету речей (IoT) буде коливатися від 20 до 200.

Давайте розглянемо проблему обсягу даних, які генерує один пристрій. Нехай це буде датчик температури. З рівнем деталізації в одну хвилину (один вимір в хвилину), обсяг становитиме 525 950 точок даних на рік. Якщо ці датчики використовуються для контролю за температурою в спеціалізованій споруді (наприклад, машинобудівельний завод), то може бути використано близько тисячі датчиків – кількість залежить від розміру самої споруди.

Характеристика обсягу, як одна з основних характеристик BigData, застосовна не лише в спеціалізованих галузях, таких як IoT. Як об'єкт аналізу можна зосередити увагу на застосунках для смартфонів, зокрема - додатках для нотаток. З великою кількістю виробників і постачальників додатків, що використовують дані в хмарному середовищі, користувачі можуть отримувати доступ до своїх нотаток на різних пристроях. Оскільки багато додатків використовують модель Freemium, де безкоштовна версія використовується як

привабливий засіб для преміум-версії, постачальники додатків на основі SaaS, як правило, мають значну кількість даних для зберігання.

2.3.2 Друга характеристика 3v моделі

Другою ключовою характеристикою галузі є швидкість генерації даних, що визначає, наскільки оперативно дані надходять в цільову систему. Розширення обсягу даних та стрімкий розвиток соціальних мереж суттєво змінили спосіб, яким суспільство сприймає і використовує інформацію.

Раніше приймалося, що дані від вчора є найбільш актуальними. Проте зараз люди в соціальних мережах активно реагують, щоб швидко повідомити про останні події. Оновлення в соціальних мережах стають застарілими практично миттєво, і користувачі зосереджують увагу на свіжих повідомленнях. Рух даних тепер відбувається практично в реальному часі, а вікно актуальності даних скоротилося до долі секунди. Швидкість оновлення даних визначається як одна з ключових характеристик BigData.

При аналізі системи з високим навантаженням, такої як Facebook, користувачі щодня завантажують понад 900 мільйонів зображень, загальний обсяг яких дорівнює близько 328,5 мільярда за рік. Це приблизно на третину більше, ніж загальний обсяг зображень, які були збережені за весь 2017 рік (250 мільярдів).

Ще однією поширеною практикою є отримання та аналіз інформації про резонанс, викликаний маркетинговою кампанією бренду. Один з прикладів цього підходу – отримання потоку даних з релевантних постів в Twitter, які потім піддаються "аналізу настроїв".

Аналіз мережевих пакетів з метою вживання відповідних заходів кібербезпеки служить живим прикладом характеристики "швидкість". Глобальна мережа пересилає величезні обсяги даних кожну секунду, і частину цього потоку необхідно фільтрувати через брандмауери корпоративної мережі. З огляду на зростання кількості кібератак, кіберзлочинності та кібершпигунства, небажаний

трафік може ховатися в загальному потоці даних, які проходять через мережевий екран.

Щоб уникнути витоку конфіденційної інформації, цей потік даних має бути досліджений і проаналізований на предмет аномалій та патернів поведінки, які можуть вказувати на потенційну загрозу. У зв'язку з ростом застосування шифрування для захисту даних, виявлення шкідливих пакетів в зашифрованому потоці стає викликом.

У сфері Інтернету речей (IoT) зі зростанням кількості підключених датчиків збільшується і загальний потік даних. Датчики є постійними джерелами інформації, яку вони передають через мережу майже в режимі реального часу. Це зумовлює постійне збільшення кількості даних.

Щодо бази даних пошукової системи, вона постійно оновлюється та поповнюється новими файлами. Швидкість оновлення даних у пошуковій системі робить її суттєвою для категоризації як системи, пов'язаної з BigData.

2.3.3 Третя характеристика 3v моделі

Третьою ключовою характеристикою BigData є різноманітність даних, яка визначається різноманітними форматами, в яких вони можуть бути збережені. Дані можуть представляти собою файл бази даних, документ Excel або CSV, або навіть простий текстовий файл. У деяких випадках дані можуть мати нетрадиційні формати, такі як відео, SMS, PDF або спеціальні формати, створені для вирішення конкретних завдань.

Організаціям необхідно забезпечити структуру та інфраструктуру для зберігання і обробки таких різноманітних даних. У реальному світі дані зазвичай зберігаються в різних форматах, і вирішення цієї проблеми відбувається завдяки розвитку галузі BigData, де приділяється особлива увага розробці технік, технологій і інструментів для ефективної обробки таких різноманітних даних.

Різноманітність даних визначає тип і характер інформації, що надає можливість фахівцям з обробки даних ефективно отримувати і використовувати цю різноманітну вихідну інформацію.

У контексті пошукової системи, різноманітність даних залежить від вимог до самої системи та її реалізації. Більшість популярних пошукових систем мають можливість виконувати пошук за заданими лексемами у різних типах файлів, таких як PDF, онлайн-книги, CSV файли, а не лише на веб-сторінках. Таким чином, пошукову систему можна вважати відповідною для BigData проекту з цього погляду.

Ще однією ключовою характеристикою BigData є достовірність (veracity). Достовірність в контексті BigData відноситься до можливих спотворень, шуму і відхилень в даних. Це питання, чи дані, які зберігаються та збираються, є значущими для аналізу конкретної проблеми. Для багатьох фахівців, достовірність у сфері аналізу даних є однією з найбільших проблем, навіть порівняно з іншими характеристиками BigData, такими як обсяг, швидкість і різноманітність.

При визначенні стратегії щодо BigData важливо, щоб команда та партнери працювали над забезпеченням чистоти даних і впроваджували процеси, які запобігають накопиченню "брудних" даних в системах, які розробляються і підтримуються.

Достовірність має велике значення, оскільки дані відіграють ключову роль у прийнятті рішень та розробці стратегій в різних галузях, таких як роздрібна торгівля, охорона здоров'я, виробничі підприємства, компанії-розробники програмного забезпечення і т.д. Однак, якщо дані ненадійні або неточні, вони можуть бути вважені недійсними, що призводить до серйозних проблем, таких як спотворення інформації та прийняття помилкових рішень.

Щодо пошукових систем, термін "достовірність" може не мати прямого відображення, оскільки вони спрямовані на пошук релевантних файлів згідно з критеріями, визначеними користувачем. В контексті пошукової системи, яка забезпечує доступ до різноманітних даних, питання достовірності даних може бути визначено користувачем при оцінці релевантності та надійності знайдених результатів. Таким чином, важливо враховувати вимоги до достовірності при роботі з даними в будь-якому контексті, включаючи пошукові системи.

2.4 Інструменти та методи Big Data

Ефективна робота з великими обсягами даних передбачає дотримання ключових принципів, серед яких варто виділити:

- горизонтальна масштабованість: система повинна бути здатною ефективно розширюватися при додаванні нових вузлів;
- відмовостійкість: з урахуванням горизонтальної масштабованості, де може бути багато машин у кластері, важливо передбачити можливі відмови та забезпечити, щоб система продовжувала працювати навіть при виході з ладу частини компонентів;
- локальність даних: у розподілених системах, де дані розподілені по різних вузлах, важливо зберігати та обробляти дані на одному вузлі, щоб уникнути зайвих витрат на передачу даних між серверами.

Сучасні інструменти для роботи з великими обсягами даних, такі як MapReduce, враховують ці принципи та працюють відповідно до них. MapReduce, розроблений Google, є моделлю розподіленої обробки даних, яка дозволяє ефективно обробляти великі обсяги даних на комп'ютерних кластерах (див.рис.2.2).

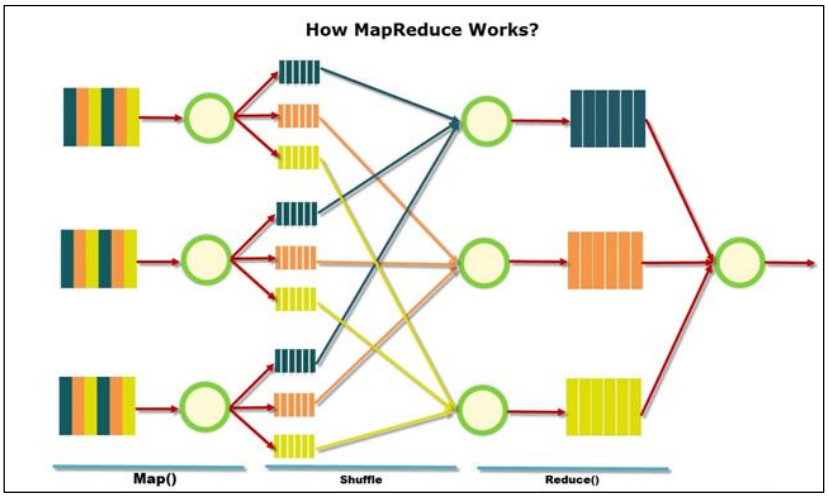

Рисунок 2.2 – Принцип роботи MapReduce (за даними [4])

MapReduce – це модель обробки даних, яка базується на тому, що дані організовані у вигляді записів та проходять через три основні стадії.

Стадія "Map": На цьому етапі дані піддаються попередній обробці за допомогою функції map(), яку визначає користувач. Функція map() застосовується до кожного вхідного запису і виконує попередню обробку та фільтрацію даних. Результатом цієї стадії є набір пар "ключ-значення", де кожен ключ може мати від нуля до N відповідних значень.

Стадія "Shuffle": На цьому етапі результати функції map() групуються за ідентифікатором ключа. Ці групи, або "кошики", служать вхідними даними для наступного етапу reduce.

Стадія "Reduce": Функція reduce, визначена користувачем, обчислює підсумковий результат для кожної групи даних. Кількість значень, що повертає функція reduce(), визначає остаточний результат завдання MapReduce. Цей процес забезпечує ефективну обробку великих обсягів даних, розподілених у вигляді записів.

Apache Hadoop є ключовим інструментом для роботи з BigData, розвиваючись від початкового фокусу на зберігання даних та виконання MapReduce-задач до обширного стеку технологій, пов'язаних із обробкою великих обсягів даних. Розроблений як проект Apache Software Foundation, Hadoop представляє собою набір утиліт, бібліотек і фреймворк для розробки та виконання розподілених програм на кластерах з сотень і тисяч вузлів (див. рис. 2.3).

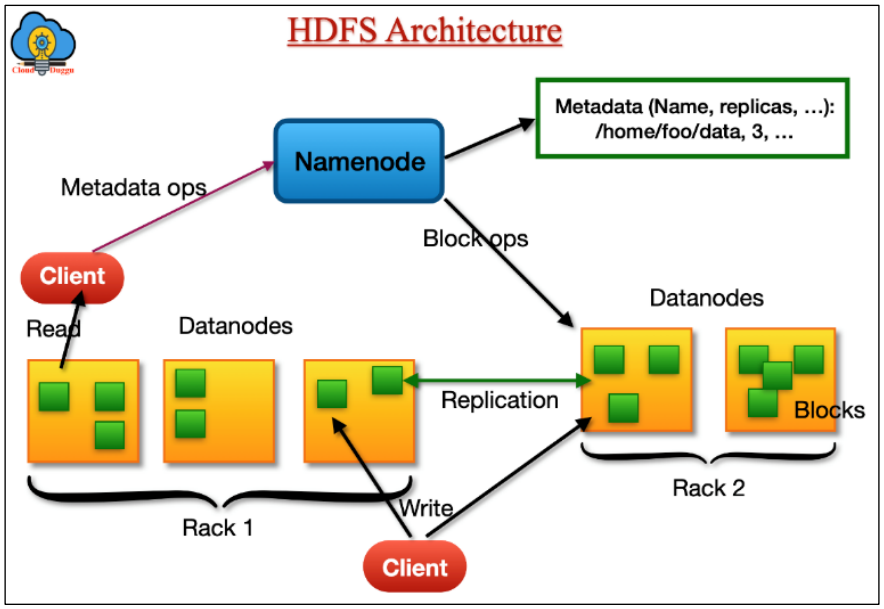

Рисунок 2.3 – Архітектура Apache Hadoop (за даними [5])

Початкова мета Hadoop була забезпечення горизонтального масштабування кластера, додаванням економічних вузлів без необхідності використання потужних серверів та дорогих систем зберігання даних. Цей підхід дозволяє розміщувати великі кластери, налагоджуючи ефективність і економічність систем. Існують великі кластери Hadoop у компаніях, таких як Yahoo (більше 4 тисяч вузлів із сумарною місткістю зберігання 15 петабайт), Facebook (приблизно 2 тисячі вузлів на 21 петабайт) і EBay (700 вузлів на 16 петабайт), що свідчить про успішність та ефективність таких систем.

Apache Hadoop складається з ключових компонентів, таких як Hadoop Distributed File System (HDFS) та Yet Another Resource Negotiator (YARN).

Hadoop Distributed File System (HDFS) — це файлова система, створена для зберігання великих файлів, які розподіляються на блоки між вузлами обчислювального кластера. Усі блоки в HDFS, крім останнього блоку файлу, мають однаковий розмір і можуть бути розміщені на декількох вузлах. Розмір блоку і кількість його реплікацій (кількість вузлів, на яких повинен бути розміщений кожен блок) визначаються на рівні файлу. Механізм реплікації гарантує стійкість розподіленої системи до відмов окремих вузлів.

Hadoop YARN (Yet Another Resource Negotiator) – це модуль, який був впроваджений у версії 2.0 (2013 р.), відповідальний за управління ресурсами кластера та планування завдань (див. рис. 2.4).

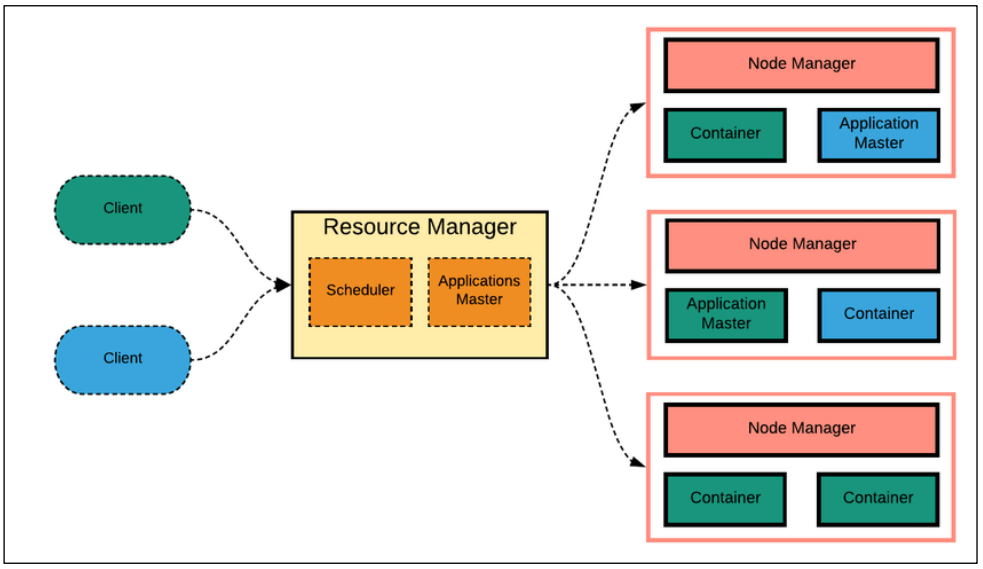

Рисунок 2.4 – Архітектура Hadoop YARN (за даними [6])

У порівнянні з попередніми версіями, де ця функція була інтегрована в модуль MapReduce, YARN функціонує як самостійний демон - Resource Manager. Він абстагує всі обчислювальні ресурси кластера і керує їх виділенням для різних розподілених обчислень. Працюючи під керуванням YARN, можна виконувати як MapReduce-програми, так і інші розподілені додатки, які підтримують відповідні програмні інтерфейси. YARN надає можливість паралельного виконання різних завдань у межах кластера та їх ізоляцію згідно з принципами мультіарендності.

Hadoop успішно вирішує завдання керування обчислювальними кластерами та горизонтального масштабування системи. Однак, через те, що його реалізація MapReduce базується на операціях, які виконуються над дисковим сховищем, це призводить до значних витрат часу на виконання. Для ефективної обробки даних виникла необхідність у більш продуктивному інструменті, такому як Spark.

Apache Spark – це фреймворк з відкритим вихідним кодом, призначений для розподіленої обробки неструктурованих і слабо структурованих даних. Він є частиною екосистеми проектів Hadoop. На відміну від традиційного обробника ядра Hadoop, Spark використовує спеціалізовані примітиви для рекурентної обробки в оперативній пам'яті, що суттєво підвищує продуктивність для певних типів завдань.

Apache Spark може взаємодіяти з середовищем, керованим Hadoop, і виступати інструментом для обробки та аналізу даних.

Ще одним ефективним та популярним інструментом для роботи з неструктурованими даними є Apache Cassandra. Це розподілена система управління базами даних, належить до класу NoSQL-систем і призначена для створення високомасштабованих та надійних сховищ великих обсягів даних у вигляді хешу. Модель зберігання даних на основі сімейства стовпців відрізняє її від інших систем, дозволяючи організувати зберігання хеша з декількома рівнями укладення. Apache Cassandra є стійкою до відмов системою управління базами даних, забезпечуючи практично лінійну масштабованість та автоматичну реплікацію даних при виході вузла з ладу.

Вищеописані інструменти представляють собою набір найбільш популярних та широко використовуваних рішень для впровадження проектів, пов'язаних із зберіганням, управлінням та обробкою об'ємних наборів як структурованих, так і неструктурованих даних. Усі вони спрямовані на підтримку горизонтального масштабування, що означає підвищення ефективності за рахунок збільшення кількості вузлів у системі в пропорційному відношенні.

Ці інструменти можуть бути застосовані для ефективного вирішення завдань при розробці пошукової системи. Наприклад, розподілена файлова система Hadoop використовується для зберігання файлів різної структури та формату, Apache Spark вирішує завдання аналізу даних для визначення релевантності та подальшого вибору файлів у відповідь на запити користувачів пошукової системи, а Apache Cassandra служить сховищем даних, що відповідають формату записів у базі даних.

Важливо відзначити, що впровадження пошукової системи за допомогою спеціалізованих інструментів Big Data передбачає зміну архітектури проекту, додаючи кластерний рівень для обробки даних. Такий підхід не гарантує вирішення проблем з високою завантаженістю системи, а лише спрямований на управління та обробку об'ємних даних, які система вже може обробляти. У разі, якщо ефективність системи за цільовими характеристиками не відповідає очікуванням, слід розглянути альтернативи щодо проектування та реалізації системи, які дозволяють уникнути залучення інструментів на рівні програмного коду та зменшити витрати порівняно з витратами на впровадження Big Data рішення.

3 АНАЛІЗ ТА ВИБІР СЕРЕДОВИЩА ДЛЯ ВИКОНАННЯ ПРОГРАМНОГО КОДУ

Під час розробки та впровадження додатку, одним із ключових аспектів є обране середовище виконання програмного коду. У випадку створення високонавантажених систем особливу вагу мають такі фактори:

- здатність ефективно працювати з паралельним виконанням через потоки або асинхронну модель програмування;
- оптимізація використання пам'яті;
- швидкість виконання операцій.

Серед найпопулярніших середовищ виконання, які мають прийнятне навантаження на пам'ять та швидкість роботи, але відзначаються високим рівнем підтримки паралельного програмування, можна виділити CLR (Common Language Runtime) та JVM (Java Virtual Machine).

3.1 Аналіз та огляд CLR

.NET Framework забезпечує Common Language Runtime (CLR) як середовище виконання програм, яке відповідає за виконання програмного коду та надає інструменти для спрощення процесу розробки. Компілятори та інші інструменти розкривають можливості CLR та дозволяють створювати код, який працює оптимально в цьому середовищі виконання. Код, що розробляється за допомогою мови програмування, орієнтованої на CLR, відомий як керований код. Він має переваги, такі як інтеграція між мовами, обробка виключень між мовами, покращена безпека, підтримка версій та розгортання, спрощена модель взаємодії компонентів та послуги налагодження та профілювання.

Щоб надати середовищу виконання можливість керувати кодом, компілятори мови повинні постачати метадані, що описують типи, члени та посилання у вашому коді. Ці метадані зберігаються разом з кодом; кожен портативний виконуваний файл (PE) CLR, що завантажується, містить метадані. CLR використовує ці дані для пошуку та завантаження класів, створення

екземплярів у пам'яті, викликів методів, генерації власного коду, забезпечення безпеки та встановлення меж контексту виконання. CLR автоматично керує обробкою макету об'єкта та управляє посиланнями на об'єкти, випускаючи їх, коли вони більше не використовуються (сборка сміття). Об'єкти, керовані таким чином, називаються керованими даними. Сбірка сміття допомагає уникнути витоків пам'яті (memory leaks) та деяких інших поширених помилок програмування.

CLR сприяє простоті розробки компонентів та програм, де об'єкти можуть взаємодіяти між собою незалежно від мови програмування. Це можливо завдяки тому, що компілятори різних мов та інструменти, спрямовані на виконавче середовище, використовують спільну систему типів, яка визначена для даного середовища. Вони також дотримуються правил виконання для визначення нових типів, а також для створення, використання, зберігання та прив'язки до них.

Частину їх метаданих становлять дані про компоненти та ресурси, з якими вони взаємодіють (залежності). Під час виконання ці дані використовуються для забезпечення належних версій всіх необхідних елементів для компонентів або додатків, щоб уникнути можливих проблем через невідповідності. Інформація про реєстрацію та стан більше не знаходиться у реєстрі, що полегшує їх встановлення та підтримку. Натомість інформація про визначені типи та їх залежності зберігається разом з кодом у вигляді метаданих, що спрощує завдання реплікації та видалення компонентів.

Переваги мови програмування CLR включають:

- підвищення продуктивності;
- можливість легкого використання компонентів, розроблених на інших мовах;
- розширення типів, які надаються бібліотекою класів;
- мовні конструкції, такі як успадкування, інтерфейси та перевантаження для об'єктно-орієнтованого програмування;
- підтримка явних безкоштовних потоків, що дозволяє створювати багатопотокові, масштабовані програми;

– обробка виключень у структурованому вигляді;
– підтримка спеціальних атрибутів;
– автоматичне управління пам'яттю (gc).

3.2 Аналіз та огляд JVM

Важливим елементом мови програмування Java є її віртуальна машина, яка забезпечує кросплатформенність виконання коду Java (і інших мов, сумісних з JVM), ефективне управління пам'яттю та захист від шкідливих програм. Ця абстрактна обчислювальна машина працює на основі набору інструкцій та користується різними областями пам'яті, а використання мови програмування з використанням віртуальної машини є досить поширеним явищем. На початку свого розвитку JVM була сильно залежною від мови програмування Java, яка виступала як високорівнева мова програмування, що дозволяла розробнику використовувати різноманітні можливості, які підтримувала сама JVM.

JVM представляє собою абстрактну машину з багатьма реалізаціями, тому перш за все важливо зосередитися на інтерфейсі цієї машини. Вона забезпечує інтерфейс та можливості, які дуже схожі на CLR. Одні з найважливіших можливостей для сучасних проблем - це ефективна підтримка паралелізму, включаючи низькорівневі конструкції, які часто необхідні в багатопоточному середовищі, такі як compareAndSet. Важливо також знати, що JVM, як середовище виконання, легко конфігурується. Тому якщо яка-небудь реалізація віртуальної машини не підходить для вирішення певної задачі, завжди можна знайти іншу, яка має потрібні можливості або оптимізовані характеристики. Наприклад, оптимізація може відбуватися в контексті певних характеристик, таких як час виконання або швидкість зміни контексту активного потоку.

3.3 Порівняння CLR та JVM та обґрунтування подальшого вибору

CLR і JVM є віртуальними машинами, які обидва виконують операції, незалежні від платформи, тобто компілюють вихідний код. Фактично, ці віртуальні машини виступають посередниками між вихідним кодом розробника та

машинним кодом системи, що дозволяє забезпечити більшу універсальність використання з різними типами процесорів. На зображенні нижче показано, що функції високого рівня у двох майже однакові (рис. 3.1).

Рисунок 3.1 – Порівняння архітектур CLR та JVM на високому рівні

Окрім їх основної функціональності, обидва також мають методи для збирання сміття, забезпечення безпеки на рівні виконання та обробки винятків. І остання спільна риса, яку варто відзначити, полягає в тому, що обидва використовують операції на основі стека, що є найпоширенішим підходом до зберігання та отримання операндів та їх результатів.

Але, звісно, для кожної подібності, яку мають ці віртуальні машини, можна знайти різницю в їхніх реалізаціях.

Одна з ключових відмінностей між CLR та JVM полягає у їх початковому призначенні: JVM була спеціально створена для роботи з Java, тоді як CLR розроблялася як нейтральна до мови платформа. Початково CLR була обмежена лише до ОС Windows, в той час як JVM була нейтральна до ОС. Однак сьогодні ці обмеження змінилися: CoreCLR працює на Linux та Mac, і багато інших мов було адаптовано для роботи з JVM (таких як Groovy, Scala, Kotlin, JRuby, Clojure, тощо). Це призводить до того, що відмінності між CLR та JVM часто відображають відмінності між мовами, які вони підтримують. Деякі з найважливіших відмінностей між мовами (наприклад, між C# та Java) реалізовані на рівні віртуальної машини. Загалом, обидві віртуальні машини мали початкові обмеження стосовно мов або операційних систем, але з часом ці обмеження були подолані, і з точки зору цих аспектів обидва рішення вважаються рівноцінними.

На рівні віртуальних машин видно ще одну значну відмінність: обидва вони використовують компіляцію JIT (Just-in-Time), але спосіб її виклику різний. У випадку з CLR весь код MSIL компілюється в машинний код в момент виклику під час виконання, тоді як JVM використовує спеціальний механізм, відомий як HotSpot, для JIT-компіляції Java-байт-коду в машинний код. Особливість HotSpot полягає в тому, що він компілює та оптимізує "гарячі" частини коду, які використовуються найчастіше. Кожен з цих підходів має свої компроміси в ефективності: CLR збирає весь машинний код одноразово, що може покращити час виконання у певних сценаріях, але компілятор HotSpot може з часом підвищити ефективність за рахунок додаткової оптимізації, якщо певні частини коду використовуються інтенсивно. Важливо також зазначити, що деякі інші реалізації JVM підтримують AOT (Ahead-of-Time) компіляцію, якщо це є більш відповідним в певних обставинах. У підсумку, можна сказати, що JVM надає більше можливостей для оптимізацій, ніж CLR.

У 2003 році Джеремі Зінгер з Кембриджського університету провів дослідження, що порівнювало роботу CLR і JVM. Результати вказують на те, що їх продуктивність була практично однаковою. Виявлено, що компілятор Java генерує менше байт-коду, що може вплинути на продуктивність VM. Проте, коли розглядали запущений код, час виконання був приблизно однаковий, з CLR та JVM, які працювали швидше, в середньому, на половину. Це дослідження підкреслює, що у протистоянні між Java та користувачами .NET важливу роль відіграють саме віртуальні машини, а не безпосередньо їхні можливості щодо ефективності додатків. Це також має сенс під час аналізу використання тієї або іншої віртуальної машини. У корпоративних компаніях розробники оперують не рідними мовами програмування, а з використанням CLR або JVM. Це особливо актуально великим компаніям з численними відділами, які працюють над різноманітними проектами та додатками.

Основні схожості між CLR та JVM включають:

– обидві є віртуальними машинами;

– обидві підтримують збирання сміття;

– вони обидві використовують операції на основі стека;
– обидві мають рівень виконання безпеки;
– обидві мають засоби для обробки винятків.
– обидві є незалежними від операційної системи;
– обидві є незалежними від мов програмування.

Основні відмінності між CLR та JVM включають:

– CLR використовує компілятор JIT, тоді як JVM використовує спеціалізований компілятор JIT під назвою Java HotSpot; деякі реалізації JVM також включають різні можливості компіляції, включаючи AOT;
– CLR має інструкції для корутин, замикань та роботи з вказівниками, що відсутні у JVM;
– JVM має більш надійні засоби для вирішення помилок та моніторингу.

Це свідчить про те, що, незважаючи на мінімальні різниці між цими віртуальними машинами, JVM має ширший простір для можливих оптимізацій завдяки великому розмаїттю існуючих реалізацій.

3.4 Докладний аналіз JVM та її складових з наступним визначенням способів контекстної оптимізації

Віртуальна машина Java – це абстрактна машина, що не залежить від платформи та забезпечує середовище виконання для байт-коду Java. Вона відповідає за те, щоб байт-код Java перетворювався у машинночитану мову. Основний метод, використаний у програмі Java, насправді обробляється самою віртуальною машиною Java (рис. 3.2).

Вплив на процес виконання віртуальної машини Java зумовлюється трьома основними компонентами: підсистемою завантажувача класів (ClassLoader), областю даних процесу виконання (runtime data area) та двигуном виконання (рис. 3.3) [9]. Функціонал динамічного завантаження класів у Java керується підсистемою ClassLoader, яка завантажує, посилає та ініціалізує файли класів під час виконання, а не на етапі компіляції.

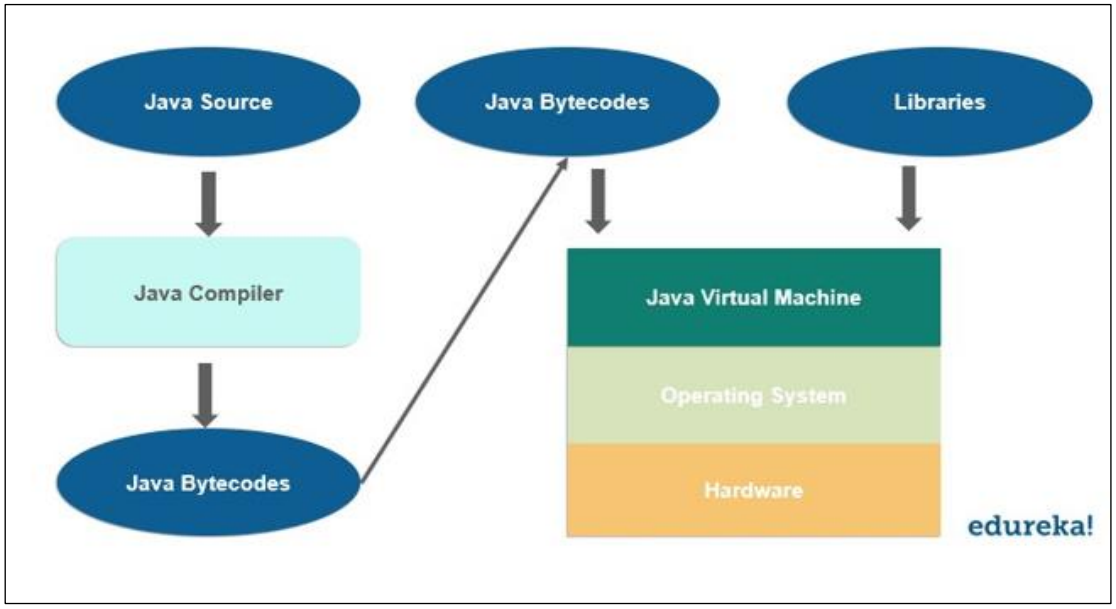

Рисунок 3.2 – Виконання коду на JVM через компіляцію та інтерпретацію

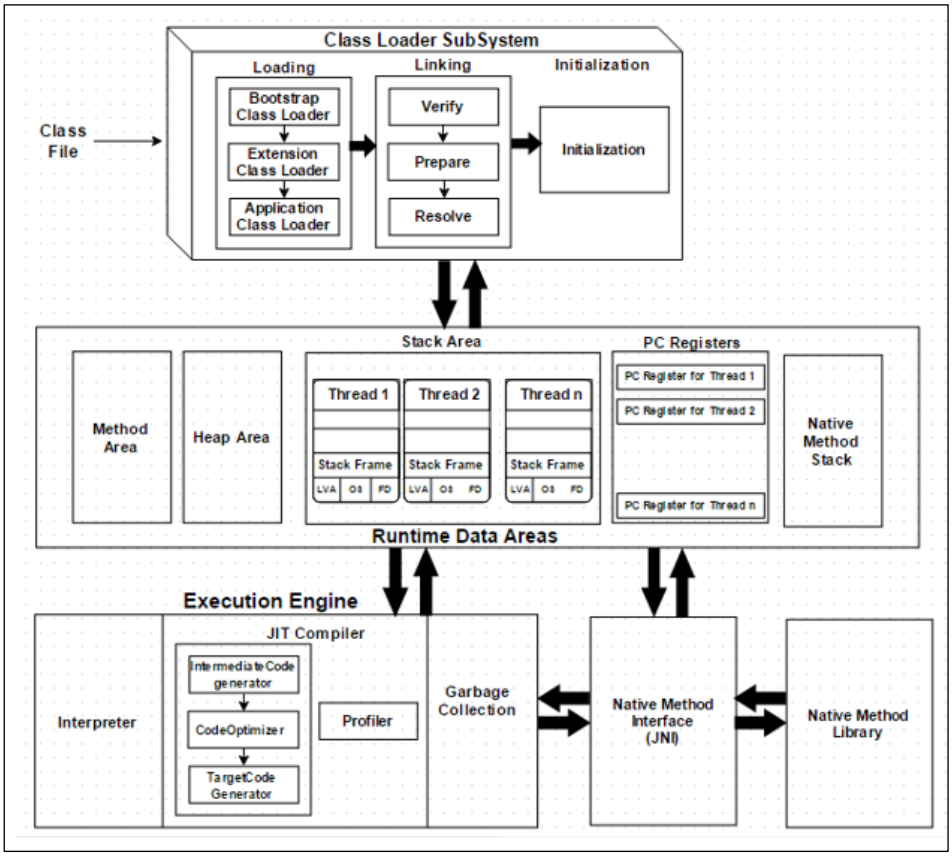

Рисунок 3.3 – Архітектурний підхід JVM

Під час завантаження класів використовуються три ClassLoader-а: BootStrap ClassLoader, Extension ClassLoader та Application ClassLoader. BootStrap ClassLoader завантажує класи з системної директорії класів, зокрема з rt.jar, та має

найвищий пріоритет. Extension ClassLoader завантажує класи з папки ext у jre\lib, а Application ClassLoader відповідає за завантаження класів з classpath рівня програми. Ці ClassLoader-и дотримуються ієрархії делегування під час завантаження файлів класу.

Під час процесу зв'язування виокремлюються три основні етапи: перевірка, підготовка та резолюція. Під час першого етапу, перевірник байт-коду аналізує правильність згенерованого байтового коду. Якщо перевірка не пройде успішно, виникає помилка перевірки (VerifyError). На етапі підготовки відбувається розподіл пам'яті для всіх статичних змінних, які отримують значення за замовчуванням. Під час резолюції всі символьні посилання на пам'ять замінюються оригінальними посиланнями з області методу.

Останнім етапом завантаження класів є ініціалізація. На цьому етапі всі статичні змінні отримують початкові значення, а статичний блок кожного завантаженого класу виконується.

Під час виконання програми пам'ять розділена на 5 основних компонентів: область методів, купу, стек, регістри комп’ютера та стеки нативних методів.

В області методів зберігаються всі дані на рівні класу, включаючи статичні змінні. Це єдина область методів у JVM і є загальним ресурсом.

У купі зберігаються всі об'єкти, а також відповідні змінні екземплярів та масиви. У JVM існує лише одна купа, і лише один екземпляр купи ініціалізується. Оскільки області Method і Heap діляться пам'яттю між кількома потоками, дані, збережені в них, не є потокобезпечними.

Для кожного потоку створюється власна область стеку виконання, де кожен виклик методу представлений як один Stack Frame у пам'яті стеку. Всі локальні змінні також зберігаються в цій області. Оскільки область стеку не є загальним ресурсом, вона забезпечує безпеку для потоків. Рамка стеку розділена на три частини: локальний масив змінних, який асоційований з методом і зберігає значення локальних змінних; стек операндів, що використовується для проміжних операцій; та дані кадру, де зберігається інформація, специфічна для методу, така як блоки обробки винятків.

Реєстри будуть використовуватися для збереження адреси поточної виконуваної інструкції для кожного потоку. Після виконання інструкції регістр Program Counter (ПК) буде оновлено для наступної інструкції.

Кожен потік матиме свій власний стек Native Method, де буде зберігатися інформація про викликані методи.

Під час виконання байт-коду він буде поступово виконуватися двигуном виконання, який читатиме і виконуватиме його по частинах.

Інтерпретатор швидше інтерпретує байт-код, але виконує його повільно. Однак його недолік полягає в тому, що при кожному виклику методу потрібно проводити повторне тлумачення.

Компілятор JIT виправляє недоліки перекладача, який використовується движком при виконанні для перетворення байтового коду. Якщо движок виявляє повторні фрагменти коду, він використовує компілятор JIT для оптимізації продуктивності системи, збираючи байт-код і перетворюючи його у нативний код, який використовується для подальших викликів методів.

Генератор проміжного коду виробляє інтермедіатний код. Оптимізатор коду забезпечує оптимізацію вищезгаданого інтермедіатного коду. Генератор цільового коду відповідає за генерацію машинного коду або власного коду.

Профілер – це спеціальний компонент, що відповідає за виявлення "гарячих точок", тобто методів, які викликаються декілька разів або ні.

Сміттєзбірник збирає та видаляє непотрібні об'єкти. Активацію сміттєзбірника можна викликати за допомогою методу System.gc(), але його виконання не гарантується. Сміттєзбірник JVM автоматично видаляє об'єкти, які були створені.

Компоненти JVM, які можна легко змінити, включають саму реалізацію JVM, що виконує всі компоненти, окрім управління пам'яттю, і сміттєзбірник, що тісно пов'язаний з реалізацією віртуальної машини через специфічний процес алокації об'єктів. Більшість активних віртуальних машин можуть підтримувати кілька реалізацій сміттєзбірника. Вони відрізняються за декількома факторами, які потребують більш детального аналізу.

Система збору сміття працює в два етапи: спочатку відбувається "позначення", а потім "вбирання". Під час етапу "позначення" збирач сміття визначає, які частини пам'яті використовуються, а які - ні. На етапі "вбирання" збирач сміття видаляє об'єкти, які були ідентифіковані під час етапу "позначення" [10].

Переваги:

– відсутність необхідності вручному розподілу пам'яті або управлінні місцем розташування, оскільки gc автоматично обробляє невикористаний простір пам'яті;
– відсутність накладних витрат на обробку вказівника;
– автоматичне управління витоком пам'яті (gc не може гарантувати повне усунення усіх протікань пам'яті, але відповідає за значну їх частину).

Недоліки:

– посилання на об'єкти потребують відстеження jvm, що призводить до додаткового навантаження на процесор, особливо коли виконуються запити з інтенсивним використанням пам'яті;
– програмісти не мають контролю над плануванням часу процесора для звільнення непотрібних об'єктів;
– використання деяких реалізацій gc може призвести до непередбачуваних зупинок програми;
– автоматизоване управління пам'яттю не досягає такої ефективності, як ручний розподіл пам'яті.
– jvm має чотири типи реалізації gc: серійний, паралельний, cms, g1.

Серійний збирач сміття є найбільш простим у реалізації, оскільки він в основному працює з одним потоком виконання. Ця реалізація сміттєзбірника заморожує всі потоки додатків під час свого запуску. Тому використання його у багатопотокових програмах, наприклад, серверному середовищі, не є найкращим варіантом. Цей збирач сміття підходить для більшості програм, що не мають високих вимог до пауз та працюють на машинах у клієнтському середовищі.

Паралельний збирач сміття, який використовується за замовчуванням в JVM, працює паралельно з декількома потоками для управління пам'яттю купи. На відміну від послідовного збирача, він забирає сміття безперервно, але при цьому може заморожувати роботу інших потоків програми під час очищення пам'яті. Розробник може встановлювати максимальну кількість потоків, які використовуються для збирання сміття та для контролю часу призупинення, пропускної здатності та розміру купи.

CMS використовує кілька потоків для збирання сміття, спрямованих на додатки, які віддають перевагу мінімізації перерв у роботі. Вони можуть дозволити спільне використання ресурсів процесора зі збирачем сміття під час роботи програми. Це означає, що програми, що використовують CMS, можуть мати меншу реактивність в середньому, але при цьому продовжують працювати під час збирання сміття. Важливо зазначити, що, оскільки CMS забезпечує одночасне збирання сміття, виклик System.gc() може призвести до припинення паралельного режиму, якщо він викликається під час активної роботи паралельного процесу. Якщо більше 98% часу витрачається на збирання сміття CMS, а менше 2% на вільне використання купи, то виникає помилка OutOfMemoryError.

G1 (Garbage First) – це засіб для оптимізації роботи програм, які працюють на мультипроцесорних системах з великим обсягом пам'яті. Вперше введений у JDK7 Update 4 та подальших версіях, він замінив колектор CMS, тому що є більш ефективним. На відміну від інших колекторів, G1 розбиває пам'ять на купи однакового розміру, кожна з яких має свій віртуальний діапазон. Під час збирання сміття G1 проводить одночасну глобальну фазу маркування, щоб визначити активні об'єкти в усій пам'яті. Після цієї фази G1 визначає порожні регіони та спочатку збирає сміття в цих областях, що забезпечує значну кількість вільного простору. Тому цей метод збору сміття отримав назву "garbage-first".

4 ПОРІВНЯННЯ МЕТОДІВ ТА КОНФІГУРАЦІЙ У ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ ДЛЯ ПОШУКУ ІНФОРМАЦІЇ З ПРАКТИЧНОЇ ТОЧКИ ЗОРУ

4.1 Опис оточуючого середовища системи

Була використана HotSpot JVM для виконання програми. JDK мав версію 1.8.0_222. Основною мовою програмування був Kotlin, яка є сумісною з JVM. Для реалізації як моделі "потік на запит", так і реактивної моделі використовувався Spring Framework п’ятої версії, реалізований двома незалежними додатками. Tomcat версії 9.0.27 виступав у ролі контейнера сервлетів для моделі "потік на запит", а Netty Web Server для реактивної моделі. Для обох контейнерів використовувалася конфігурація Spring Boot за замовчуванням.

Сервер було глобально розгорнуто та доступно через Інтернет. Це означає, що використовувався віртуальний сервер через програмне забезпечення маршрутизатора з подальшою переадресацією на локальний сервер. Декілька мобільних пристроїв були клієнтами цього сервера та виконували різні запити за допомогою графічного інтерфейсу, що був розроблений для цього додатку. Для аналізу роботи веб-сервісу був використаний профайлер VisualVM, який є складовою JDK.

Для інформаційної складової програмного забезпечення використовувалася система управління базами даних MySQL. База даних була заповнена даними з датасету про пошукові запити з ключовими словами, пов'язаними з квитками на авіарейси та перельотами, за допомогою спеціально створеного конвертера з файлів у форматі CSV. Загалом у базі даних міститься 96000 записів.

4.2 Виконання та оцінка результатів використання стандартної моделі для прийому та обробки запитів

У рамках наступного набору експериментів було використано модель обробки запитів "потік на запит", яка реалізована у фреймворку Spring MVC, а в якості контейнера сервлетів виступає Tomcat. На рисунку 4.1 показано початковий стан системи під час запуску веб-сервісу з виконанням одного запиту

для того, щоб фреймворк міг ініціалізувати всі необхідні компоненти, які будуть використані під час експерименту. На рисунку 4.2 показано стан потоків системи. Для першого експерименту був використаний паралельний збірник сміття за допомогою додаткового параметра "-XX:+UseParallelGC".

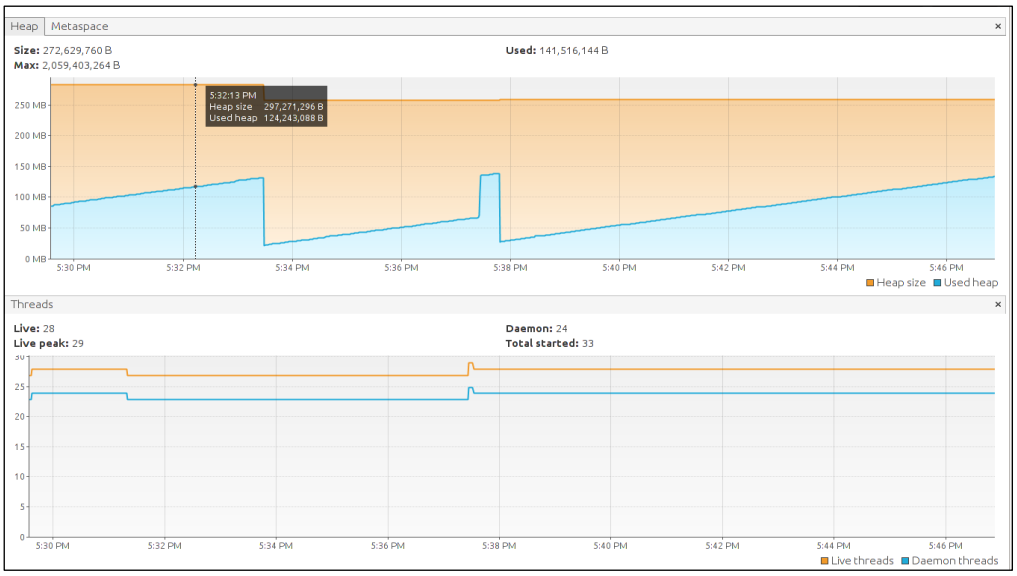

Рисунок 4.1 – Стан веб-сервера з набором потоків після налаштування зі сховищем сміття Parallel

На рисунку 4.2 видно, що в Tomcat було створено пул потоків з початковою кількістю 10 виконавчих потоків, готових обробляти вхідні запити.

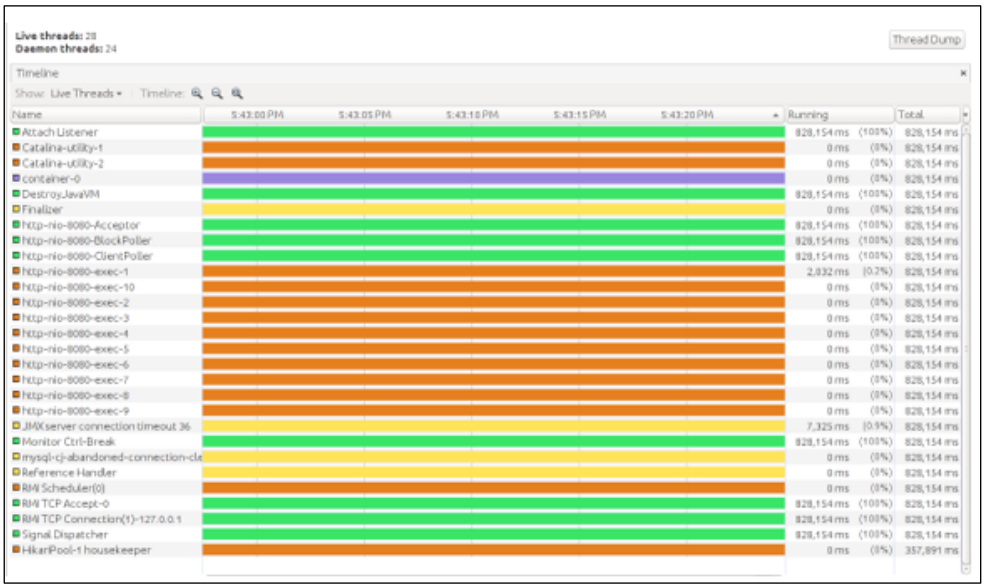

Рисунок 4.2 – Стан потоків виконання веб-сервісу за допомогою паралельного пулу потоків та збірника сміття

Слід відзначити, що для Tomcat використовується адаптер вводу-виводу без блокування (це вказують назви потоків з префіксом "nio" – non-blocking input-output). У ході експерименту з двох смартфонів було відправлено по одній тисячі запитів до розгорнутого веб-сервісу. На рисунку 4.3 зображено стан системи після виконання двох тисяч запитів, по тисячі на кожен смартфон.

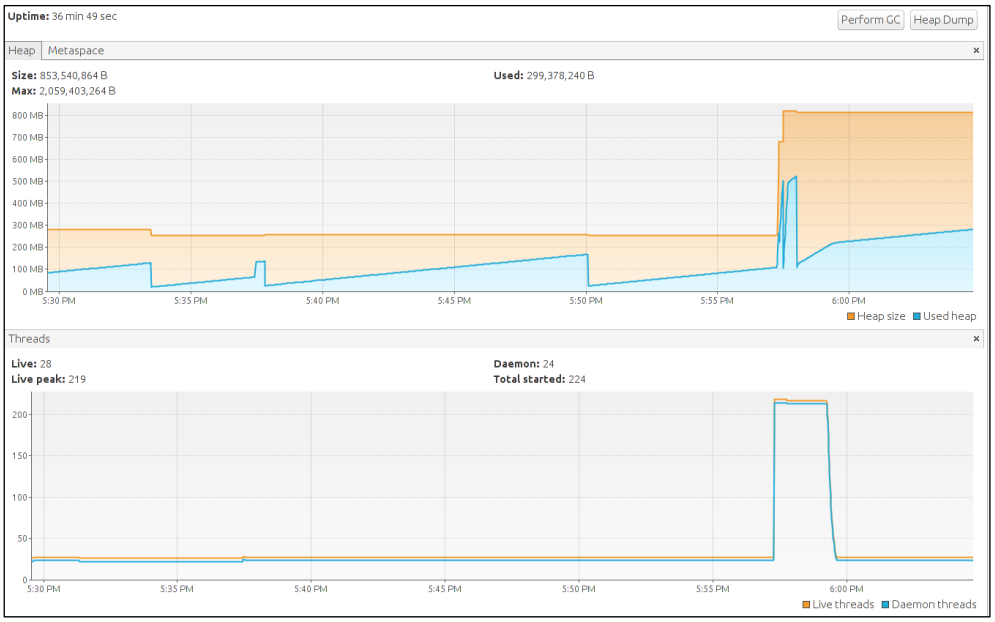

Рисунок 4.3 – Результати після обробки двох тисяч запитів одночасно з системою пулу потоків і паралельним збирачем сміття

Щодо використання пам'яті, у віртуальної машини було виділено близько 820 мегабайт, проте максимально використаний обсяг склав 526 мегабайт. Паралельний збірник сміття демонструє значну швидкість виконання, що залежить від кількості використовуваних потоків. Можна встановити, що збір не використованих об'єктів відбувається, коли пам'ять заповнюється на 60 відсотків.

Стосовно стану виконавчих потоків, варто зауважити, що реалізація контейнера сервлетів Tomcat демонструє високу ефективність. Замість того, щоб блокувати потоки, вони перебувають у стані "паркування", що означає, що замість того, щоб залишатися активними, але заблокованими, вони переходять у неактивний режим. Потоки у цьому стані використовують значно менше системних ресурсів. Важливо зазначити, що максимальний обсяг пулу потоків за конфігурацією за замовчуванням становить 200, отже, максимальна кількість

потоків у системі складає 220, з них 20 потоків є системними. Якщо система не отримує достатньо запитів для того, щоб максимальна кількість потоків працювала одночасно, контейнер сервлетів починає вивільняти виконавчі потоки. Саме через це кількість активних потоків швидко зменшується до мінімуму, що становить десять, після виконання всіх запитів. На рисунку 4.4 можна побачити графічне відображення стану потоків.

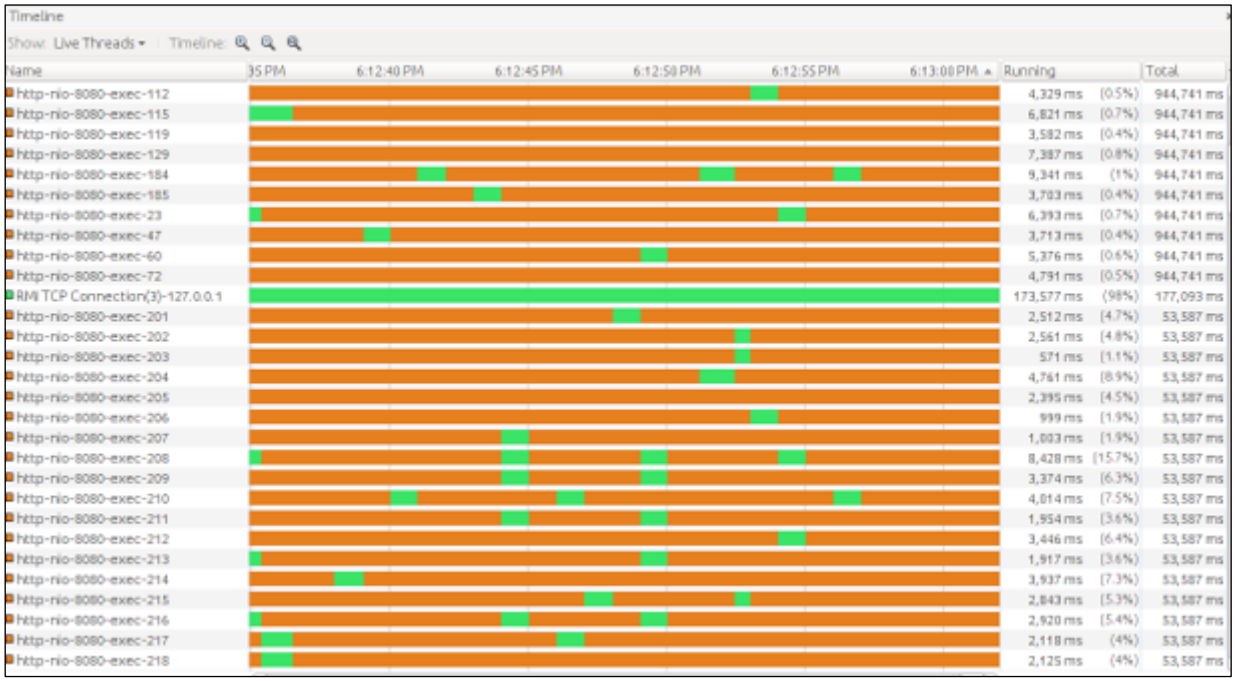

Рисунок 4.4 – Результативність потоків у системі з пулом потоків під час виконання двох тисяч одночасних запитів з використанням паралельного збирання сміття

Інший варіант реалізації включає альтернативний підхід до збирання сміття – G1. На діаграмі 4.5 показано стан віртуальної машини після запуску веб-сервісу. З контексту стану пам’яті можна зрозуміти, що збирання сміття G1 відбувається значно частіше, ніж у випадку Parallel, та відбувається у стабільному темпі. Порівняно з недетермінованим збиранням сміття Parallel, яке відбувається не в сталому режимі та змінюється залежно від об’єму використаної пам’яті віртуальною машиною, G1 виконує цю операцію детерміновано, керуючись фіксованим порогом використання пам’яті та у регулярному режимі – приблизно кожні 6 хвилин.

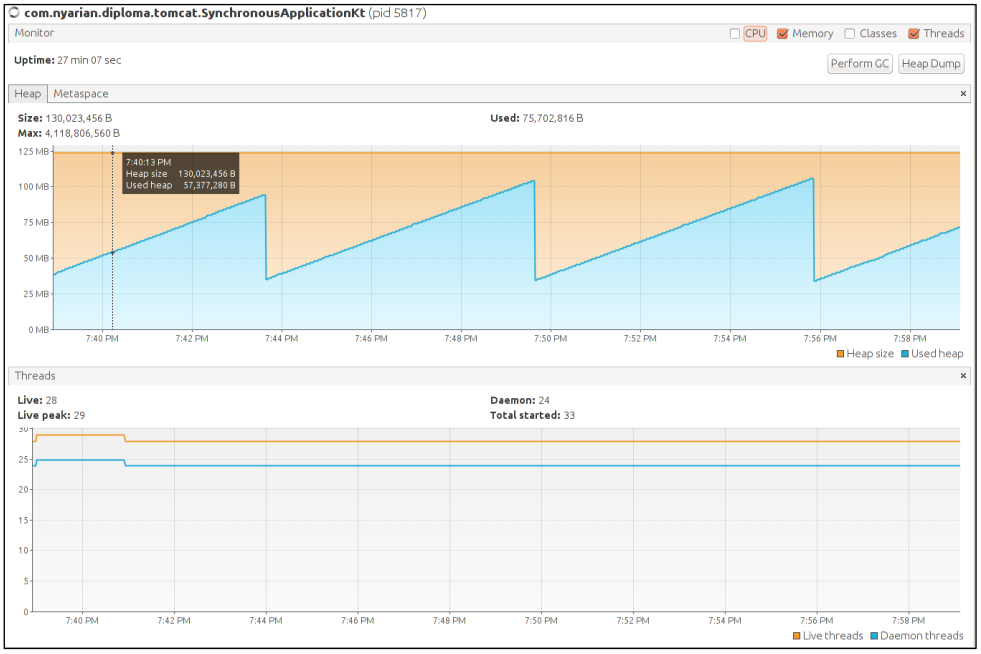

Рисунок 4.5 – Результати після запуску веб-сервера з використанням пулу потоків та ініціалізації зі збірником сміття G1

На рисунку 4.6 зображений загальний стан використання пам’яті та потоків виконання після двох тисяч запитів.

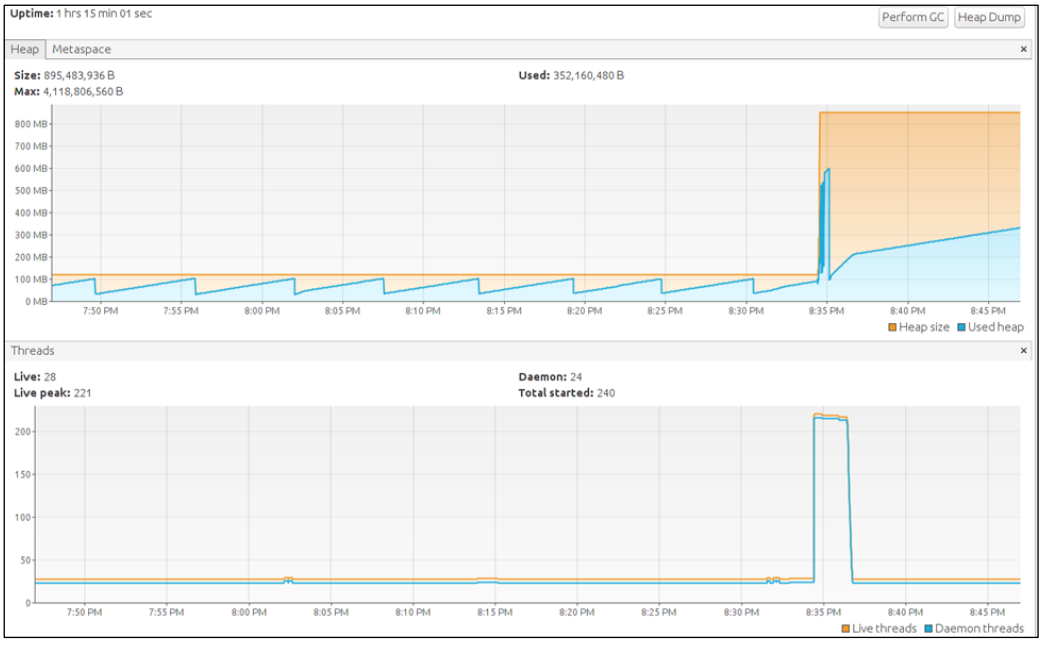

Рис. 4.6 – Загальний стан веб-сервісу з пулом потоків після обробки двох тисяч запитів з G1

Обсяг пам’яті, що був виділений для віртуальної машини, становить 854 мегабайти; максимально використаний обсяг пам’яті становить 604 мегабайти. З

цього можна постановити, що щодо споживання пам’яті збірник сміття Parallel дав кращий (менший) результат ніж G1.

4.3 Підготовка, впровадження та аналіз результатів реактивної моделі отримання та обробки запитів

У рамках наступної серії експериментів використовувалась реактивна модель обробки запитів, яка реалізована за допомогою фреймворка Spring Reactor, і в якості контейнера сервлетів виступав Netty. На рисунку 4.7 показано стан системи під час запуску веб-сервісу з виконанням одного запиту, щоб фреймворк мав можливість ініціалізувати всі необхідні компоненти для подальших експериментів. На рисунку 4.8 представлений стан потоків системи. У першому експерименті був використаний паралельний збірник сміття за допомогою додаткового флагу "-XX:+UseParallelGC".

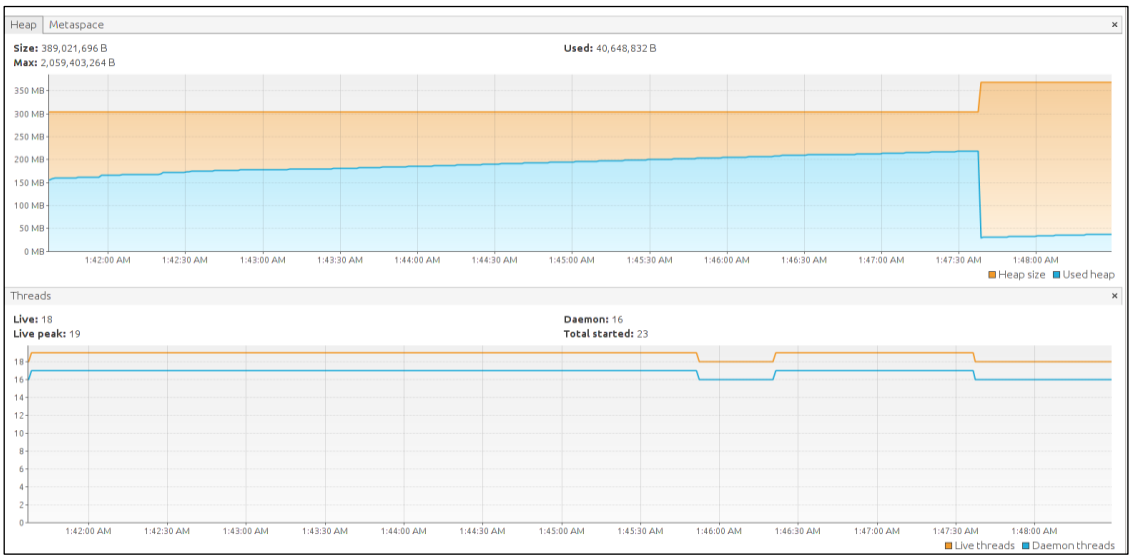

Рисунок 4.7 – Стан реактивного веб-сервера після ініціалізації паралельним збирачем сміття

На рисунку 4.8 видно, що у сервісі було створено сім потоків для регулярного перевіряння сокетів на наявність нових запитів. У цьому випадку, на відміну від попереднього варіанту, де потоки знаходились у стані "parked" до надходження нових запитів, потоки в даній реалізації постійно активні та постійно перевіряють сокети.

Рисунок 4.8 – Режим потоків виконання реактивного веб-сервісу, що використовує збірник сміття Parallel

Це призводить до певних накладних витрат на підтримку веб-сервісу в його пасивному стані, коли він не отримує запитів. Однак загальна кількість створених потоків менше, ніж у попередній реалізації, навіть на дев'ять, що є значним показником. В результаті цього можна зазначити, що для систем з низьким навантаженням використання пулу потоків та синхронної реалізації є більш ефективним з точки зору споживаних ресурсів. Це через те, що для підтримки неактивних потоків потрібно мінімально використовувати системні ресурси.

Для порівняння результатів реалізації моделі "потік на з'єднання" необхідно створити середовище, аналогічне тому, що використовувалося під час попередніх експериментів. Це означає виконання двох тисяч паралельних запитів до сервера. Стан системи після обробки запитів представлений на рисунку 4.9, а стан потоків виконання – на рисунку 4.10.

Можна сформулювати, що розмір зарезервованої купи у новій реалізації становить лише відсоток від попередньої. Обсяг купи, що був зайнятий під час обробки запитів, зменшився на 49%. Кількість потоків виконання скоротилася до 21, що є лише 10% від кількості у попередній реалізації. Варто також відзначити, що сміття було зібрано у два рази менше, ніж у попередньому випадку.

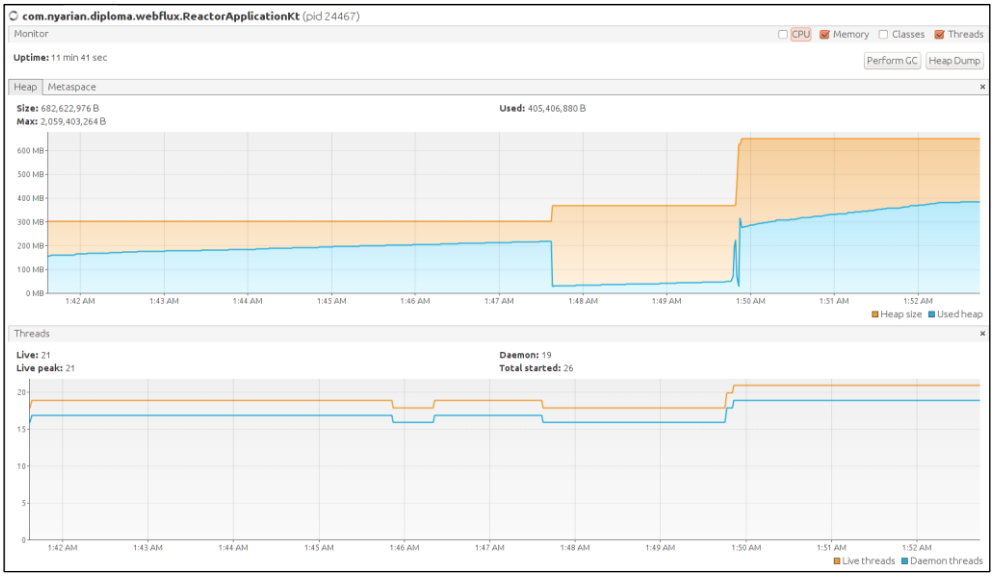

Рисунок 4.9 – Стан системи реактивного веб-сервісу зі збірником сміття Parallel після обробки 2000 запитів

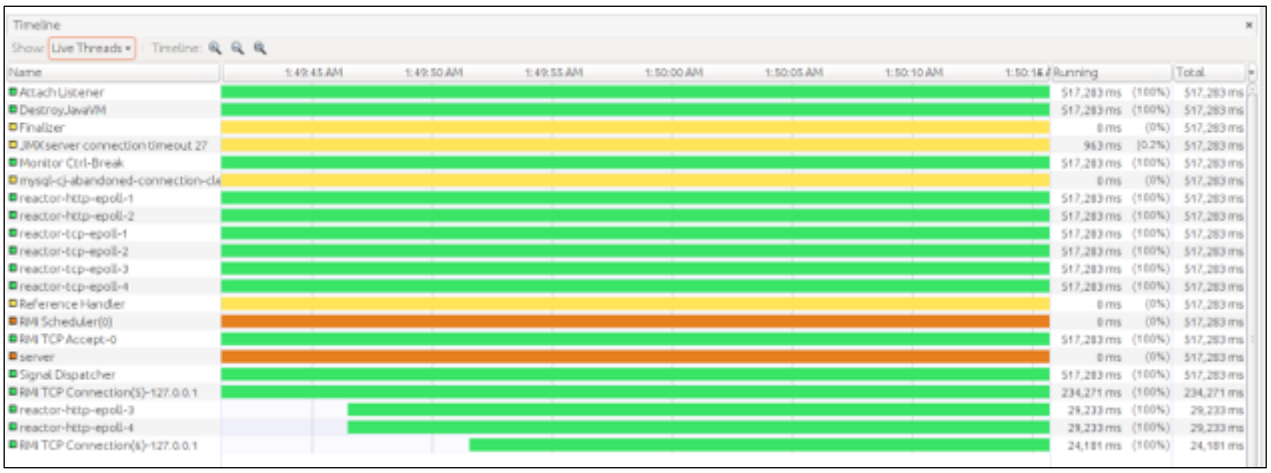

Рисунок 4.10 – Стан потоків виконання реактивного веб-сервісу із паралельним збірником сміття під час обробки 2000 запитів

Щодо обробки потоків, можна зазначити, що у поточній реалізації створено лише три допоміжні потоки, тоді як у попередній версії їх було 190 і можна було створювати ще більше, якщо пул потоків не був обмеженим. Для другого експерименту будемо використовувати сбірник сміття G1. Після ініціалізації система має такий вигляд, як зображено на рисунку 4.11. Зауважимо, що в поточній реалізації розмір зарезервованої пам'яті становить 40% від зарезервованої пам'яті зі сбірником сміття Parallel. Максимальний обсяг

використаної пам'яті складає 94 мегабайти порівняно з 220 мегабайтами у попередній реалізації, що становить 43% від обсягу пам'яті у попередній версії.

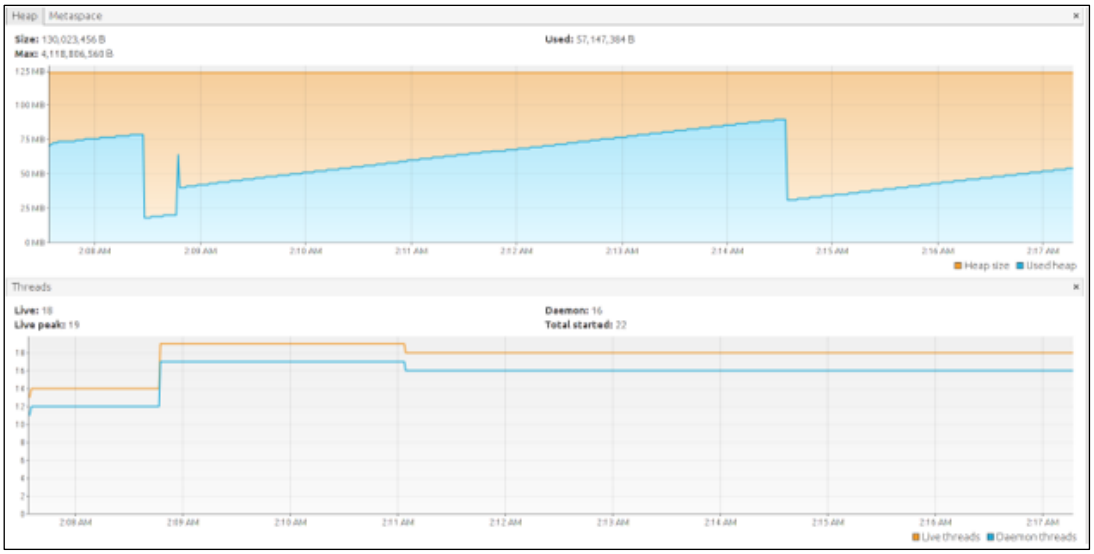

Рисунок 4.11 – Реактивний веб-сервер, після ініціалізації, переходить до роботи з сміттєзбірником G1

Результати експерименту для поточної реалізації можна побачити на рисунку 4.12. Обсяг купи, який було зарезервовано, склав 828 мільйонів байт, що на 146 мільйонів більше, ніж у попередній версії. Це означає, що результат гірший на 18%. Максимальний обсяг використаного простору склав 525 мільйонів байт, що порівняно з 320 мільйонами у попередній версії є на 205 мільйонів більшим. Це означає погіршення результату на 39%. Однак сміттєзбірка відбулася лише один раз, на відміну від попередніх випадків, коли проводилася двічі. Це свідчить про те, що, хоча цей підхід потребує більше пам'яті, він спричинює системі на 50% менше періодів "зупинки світу". Обидва підходи мають свої переваги та недоліки, проте в разі високонавантажених систем стратегія G1 є більш вигідною. У реальних умовах навантаження рідко різко змінюється, і виділення пам'яті для купи є складним процесом як для процесу, так і для операційної системи. Крім того, менше збірок сміття в загальному випадку є більш привабливою характеристикою, ніж великі коливання навантаження на пам'ять.

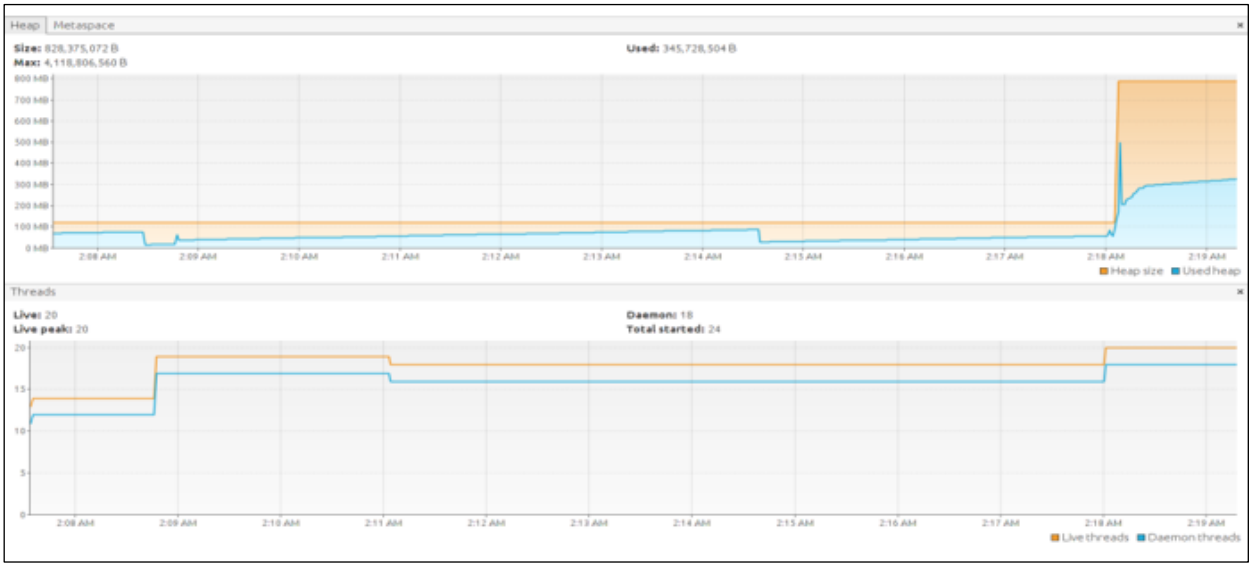

Рисунок 4.12 – Стан реактивного веб-сервера після 2000 запитів з використанням збирача сміття G1

4.4 Порівняння отриманих результатів

У ході експериментів було встановлено, що для моделі "потік на запит", обмеженої обсягом пулу потоків, який обробляє запити, використання сміттєзбирача Parallel є більш ефективним рішенням. У випадку реактивної реалізації, було визначено, що G1 є більш вигідним рішенням для високонавантажених систем. Щодо зарезервованого обсягу пам'яті, реактивна система перевершує альтернативу на 3% - 852 мільйони байт проти 828 мільйонів (див. рис. 4.13).

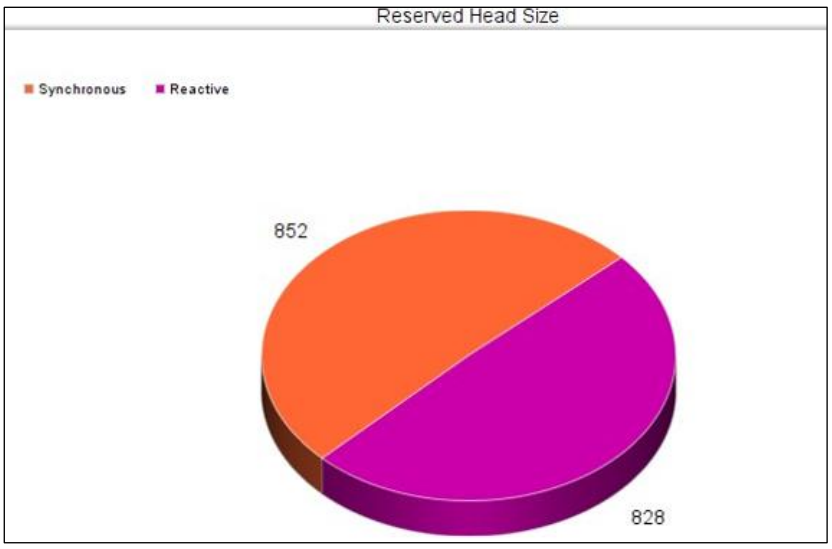

Рисунок 4.13 – Порівняння зарезервованого обсягу купи

Що стосується кількості обсягу купи, то відхилення в продуктивності системи становить менше 0,005% (рис. 4.14).

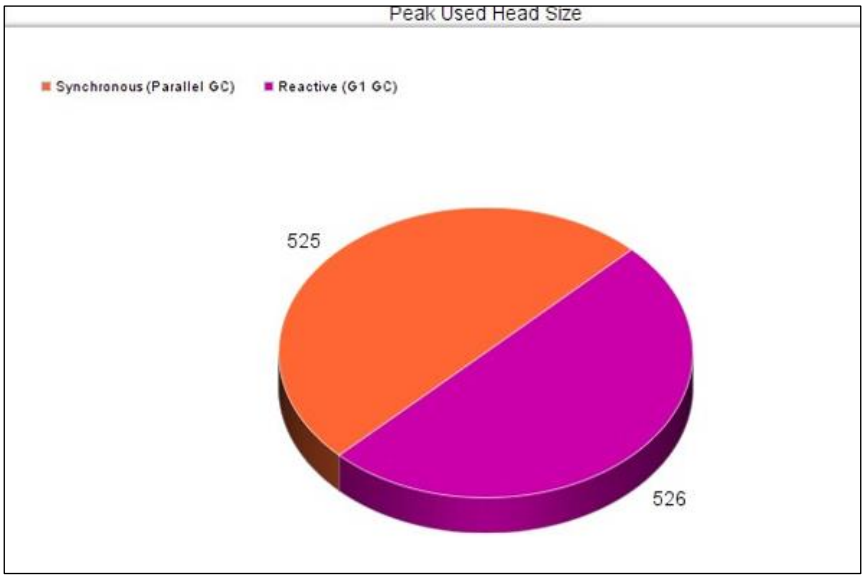

Рисунок 4.14 – Порівняння найбільш ефективно використаного обсягу купи

Щодо частоти збору сміття, синхронна система здійснила 2 цикли порівняно з одним циклом в реактивній системі (див. рис. 4.15). З цього можна зробити висновок, що навантаження на пам'ять у реактивній системі значно менше, ніж у синхронній, оскільки купа була заповнена до максимуму двічі у випадку першої виконавчої середовища. Це означає, що щодо навантаження на пам'ять реактивна система має значно кращий результат.

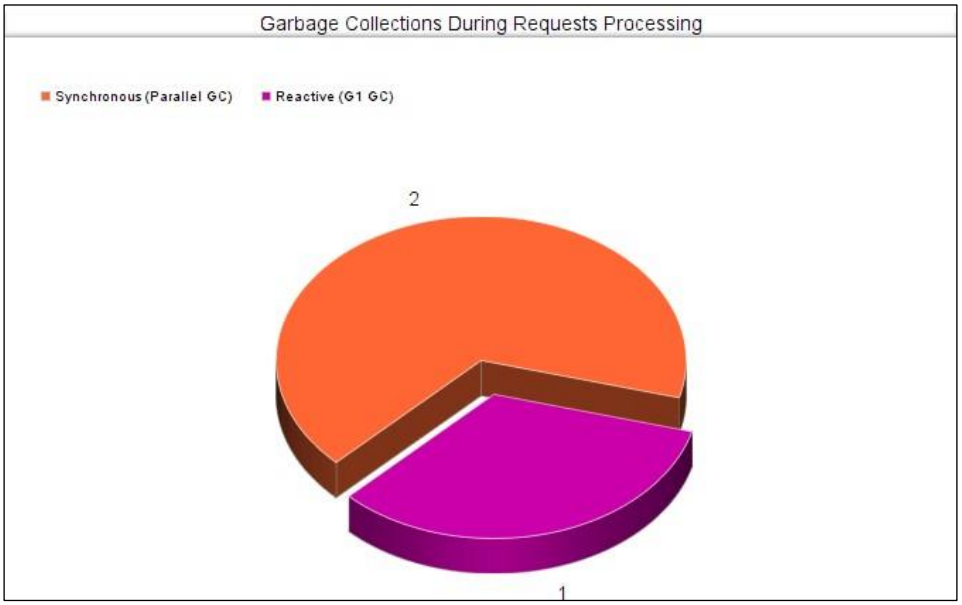

Рисунок 4.15 – Перевірка кількості збірок сміття під час обробки запитів

Щодо продуктивності потоків виконання, реактивна система переважає синхронну на 90% – 21 потік проти 220, де 220 є встановленим пороговим значенням (рис. 4.16). Швидкість, з якою система надає результат клієнту, трохи вища у випадку реактивної системи, що свідчить про менш ефективне управління потоками в синхронній системі порівняно з реактивною.

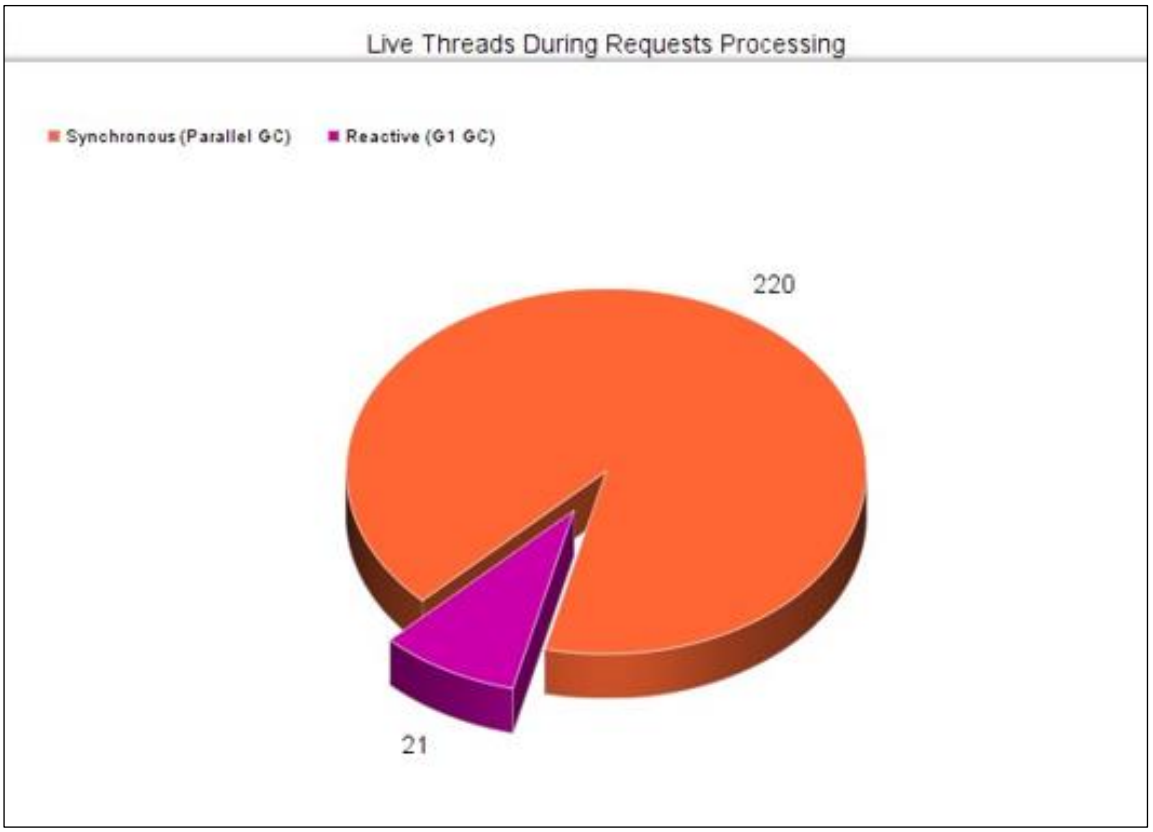

Рисунок 4.16 – Порівняння активних потоків в системі

Щодо часу виконання, синхронна реалізація відповідала на дві тисячі запитів за 20 секунд і 777 мілісекунд. У контексті реактивної реалізації, дві тисячі запитів були оброблені за 23 секунди і 284 мілісекунди. Порівняння наведене на рисунку 4.17.

Це дає змогу зробити висновок, що реактивна система працює значно ефективніше, ніж синхронна реалізація у даному контексті та при такому навантаженні на систему. Також важливо зауважити, що для реактивної системи реалізація збирача сміття за замовчуванням виявилася більш ефективною, ніж попередня реалізація.

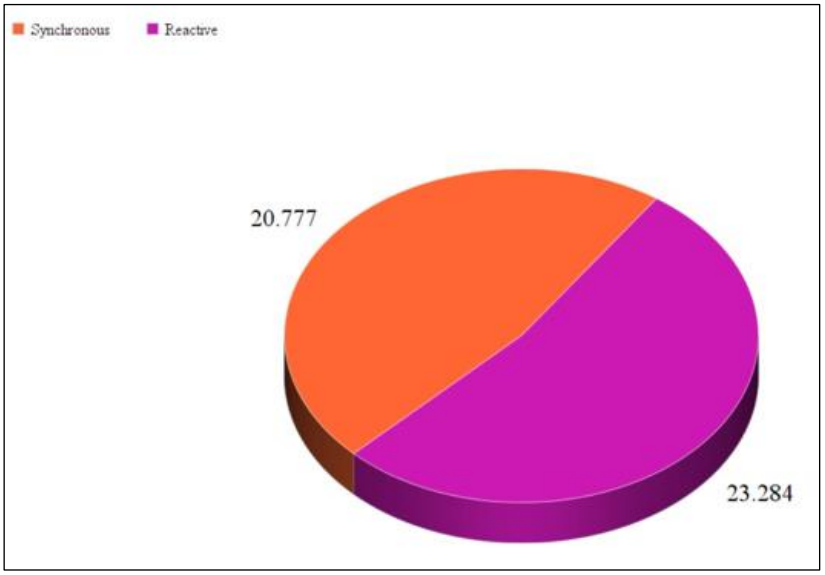

Рисунок 4.17 – Оцінка часу обробки двох тисяч запитів

Однак, якщо головним аспектом є швидкість, синхронна модель показала значно кращі результати, тож з погляду швидкості виконання вона виявляється ефективнішою, ніж реактивна.

ВИСНОВКИ

Основною темою дослідження була проблема обробки даних у високонавантажених системах, зокрема, у системі пошуку інформації. Робота акцентувала увагу на актуальності цього питання та пропонувала можливі шляхи вирішення.

Основні підходи до вирішення даної проблеми включають передові методи утилізації середовища виконання, що підтримують паралелізм. Зокрема, розглядалися метод асинхронного програмування та його нащадок – метод реактивного програмування, а також методи та інструменти, пов'язані з обробкою великих обсягів даних (Big Data). Обраним для вирішення проблеми став метод реактивного програмування, який забезпечує ефективну утилізацію системних ресурсів та потоків виконання для обробки потоку подій, що може бути потенційно нескінченим (у відношенні до розробленої системи – запитів).

У практичній частині експериментів метою було порівняння ефективності стандартного підходу до проектування та реалізації веб-сервісів, а саме моделі "потік на запит" з обмеженим пулом потоків для обробки запитів, з ефективністю системи, що використовує реактивне програмування. Після цього проводилася оцінка ефективності альтернативних конфігурацій середовищ виконання HotSpot JVM, зокрема альтернативних реалізацій компонента збірника сміття.

Під час експериментів було виявлено, що в контексті даного середовища реактивна модель є значно ефективнішим підходом для проектування та впровадження потенційно високонавантаженого веб-сервісу, особливо з урахуванням управління потоками виконання та процесами збирання сміття.

Однак з точки зору швидкості обробки запитів, синхронна модель показала на 15% кращі результати. Таким чином, у системах, де швидкість має переважний статус, та навантаження співпадає з оточенням, що активно використовувалося під час експериментів, синхронний підхід до обробки запитів та даних може вважатися більш ефективним рішенням.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Клієнт-Серверна Архітектура URL - https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 02.12.2023).

2. Exploring the Inner Workings of the Event Loop URL - https://www.devtip.co/exploring-the-inner-workings-of-the-node-js-event-loop/ (дата звернення: 02.12.2023).

3. IBM Big Data characteristics – 3V. Adopted from (Zikopoulos and Eaton 2011) URL - https://www.researchgate.net/figure/BM-Big-Data-characteristics-3V-Adopted-from-Zikopoulos-and-Eaton-2011_fig1_258247680 (дата звернення: 02.12.2023).

4. Introduction to Hadoop MapReduce – What is MapReduce & How it works URL - https://prwatech.in/blog/hadoop/mapreduce/introduction-to-hadoop-mapreduce/ (дата звернення: 02.12.2023).

5. Apache Hadoop Architecture URL - https://www.cloudduggu.com/hadoop/architecture/ (дата звернення: 02.12.2023).

6. Architecture Hadoop YARN URL - https://www.researchgate.net/figure/Apache-Hadoop-Yarn-Architecture_fig5_329387483 (дата звернення: 02.12.2023).

7. Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley. The Java Virtual Machine Specification, Java SE 7 Edition, URL: https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-1.html#jvms-1.2 (дата звернення: 02.05.2024).

8. CLR vs JVM: How the Battle Between C# and Java Extends to the VM-Level. URL: https://blog.overops.com/clr-vs-jvm-how-the-battle-between-net-and-java-extends-to-the-vm-level/ (дата звернення: 02.05.2024).

9. The JVM Architecture Explained. URL: https://dzone.com/articles/jvm-architecture-explained (дата звернення: 03.05.2024).

10. JVM Garbage Collectors. URL: https://www.baeldung.com/jvm-garbage-collectors/ (дата звернення: 04.05.2024).

11. Study of Prediction and Classification Models in the Problems of Diabetes Among Patients with a Stroke in Different Living Conditions / Huliiev, N., Peretiaha, M., Khovrat, A., Teslenko, D., Nazarov, A. // Innovative Technologies and Scientific Solutions for Industries, (2 (24)), 54-61. URL: https://journals.uran.ua/itssi/article/view/285501/279567

12. Selection of Artificial Neural Networks for Disease Prediction / Iryna Kyrychenko, Oleksii Nazarov, Nural Huliiev, Oleksii Avdieiev // COLINS-2023: 7th International Conference on Computational Linguistics and Intelligent Systems, April 20–21, 2023, Kharkiv, Ukraine (CEUR Workshop Proceedings (CEUR-WS.org)). Volume I: Machine Learning Workshop, pp. 236–248. URL: https://ceur-ws.org/Vol-3387/paper18.pdf

13. Parallelization of the VAR Algorithm Family to Increase the Efficiency of Forecasting Market Indicators During Social Disaster / Artem Khovrat, Volodymyr Kobziev, Alexei Nazarov, Sergiy Yakovlev // Information Technology and Implementation (IT&I-2022), November 30 - December 02, 2022, Kyiv, Ukraine. – 12 pp. CEUR Workshop Proceedings, 2022, 3347, pp. 222–233. – URL: https://ceur-ws.org/Vol-3347/Paper_19.pdf

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

11. Study of Prediction and Classification Models in the Problems of Diabetes Among Patients with a Stroke in Different Living Conditions / Huliiev, N., Peretiaha, M., Khovrat, A., Teslenko, D., Nazarov, A. // Innovative Technologies and Scientific Solutions for Industries, (2 (24)), 54-61. URL: https://journals.uran.ua/itssi/article/view/285501/279567

12. Selection of Artificial Neural Networks for Disease Prediction / Iryna Kyrychenko, Oleksii Nazarov, Nural Huliiev, Oleksii Avdieiev // COLINS-2023: 7th International Conference on Computational Linguistics and Intelligent Systems, April 20–21, 2023, Kharkiv, Ukraine (CEUR Workshop Proceedings (CEUR-WS.org)). Volume I: Machine Learning Workshop, pp. 236–248. URL: https://ceur-ws.org/Vol-3387/paper18.pdf

13. Parallelization of the VAR Algorithm Family to Increase the Efficiency of Forecasting Market Indicators During Social Disaster / Artem Khovrat, Volodymyr Kobziev, Alexei Nazarov, Sergiy Yakovlev // Information Technology and Implementation (IT&I-2022), November 30 - December 02, 2022, Kyiv, Ukraine. – 12 pp. CEUR Workshop Proceedings, 2022, 3347, pp. 222–233. – URL: https://ceur-ws.org/Vol-3347/Paper_19.pdf