Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних систем та технологій
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра
(бакалавра, спеціаліста, магістра)

Студента Максименка Олександра Володимировича
(ПІБ)
академічної групи 126-22м-1
(шифр)
спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)
за освітньо-професійною програмою

«Інформаційні системи та технології»
(офіційна назва)
на тему Дослідження метрик ефективності використання MongoDB, зібраної з вихідного коду
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	Сергєєва К.Л.			
розділів:				
Розділ 1	Сергєєва К.Л.			
Розділ 2	Сергєєва К.Л.			
Розділ 3	Сергєєва К.Л.			
Розділ 4	Сергєєва К.Л.			

Рецензент	Ширін А.Л.			

Нормоконтролер	Коротенко Г.М.			

Дніпро
2023

ЗАТВЕРДЖЕНО:
завідувач кафедри

інформаційних технологій та комп'ютерної інженерії
(повна назва)

_____ Гнатушенко В.В.
(підпис) (прізвище, ініціали)

«_____»_____20__ року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня _____магістр_____
(бакалавра, спеціаліста, магістра)

студенту __Максименку О.В._ **академічної групи** __126м-22-1___
(прізвище та ініціали) (шифр)

спеціальності _____126 «Інформаційні системи та технології»_____

за освітньою-професійною програмою _____

_____«Інформаційні системи та технології»_____

на тему ____Дослідження метрик ефективності використання MongoDB,_____
_____зібраної з вихідного коду_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 09.10.23 № 1227-С

Розділ	**Зміст**	**Термін виконання**
Розділ 1	Початкові відомості дослідження	1.10.23 – 31.10.23
Розділ 2	Генерація тестових даних	1.11.23 – 15.11.23
Розділ 3	Порівняння використовуємих ресурсів	16.11.23 – 30.11.23
Розділ 4	Порівняння швидкості виконання CRUD операцій	1.12.23 – 15.12.23

Завдання видано _____ __Сергєєва К.Л._
(підпис) (прізвище, ініціали)

Дата видачі _____

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ Максименко О.В._
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 71 стор., 35 рис., 3 додатків, 31 джерел.

Об’єкт дослідження: швидкість виконання операцій програмою та її потреба у ресурсах як основні метрики продуктивності.

Предмет дослідження: вплив збірки MongoDB з вихідного коду на її продуктивність.

Мета магістерської роботи: виявлення яким чином і на які аспекти продуктивності впливає збірка MongoDB з вихідного коду.

У вступі подано стан проблеми та виконана постановка задачі дослідження.

В першому розділі подана інформація щодо перспективності використання різних опцій компілятору при збірках пакетів.

У другому розділі пояснена проблема великого обсягу тестових даних та запропоновано рішення з їх генерації.

У третьому розділі наведено порівняння обсягу використовуємих ресурсів при роботі з бінарною та скомпільованою версіями MongoDB.

У четвертому розділі наведено порівняння часу виконання CRUD операцій при роботі з бінарною та скомпільованою версіями MongoDB.

Наукова новизна отриманих результатів кваліфікаційної роботи визначається тим, що вперше зроблено комплексний аналіз щодо мотивів збірки MongoDB з вихідного коду на комп’ютері кінцевого користувача.

Практична цінність результатів полягає в тому, що зроблений в роботі аналіз визначає ті категорії користувачів, які виграють від компіляції MongoDB з вихідного коду.

MONGODB, NOSQL, ВІДКРИТИЙ ВИХІДНИЙ КОД, ЗБІРКА, CRUD, CPU, RAM.

ABSTRACT

Explanatory note: 71 pages, 35 figures, 3 applications, 31 sources.

Object of research: operation speed and resource demand as main performance metrics.

Subject of research: impact on MongoDB performance by building it from source.

Purpose of Master's thesis: discovery in which way and on what aspects are affected by MongoDB compilation.

In the introduction the status of the problem and the formulation of the research task are presented.

In the first section prospects of using different compiler options are explained.

In the second section problem of big amount of test data is explained and approach to generate it is proposed.

In the third section it is compared resource consumption by binary MongoDB version and compiled MongoDB version.

In the fourth section it is compared CRUD operations execution time by binary MongoDB version and compiled MongoDB version.

Originality of research is associated with multipurpose analysis of building MongoDB from source code on end users computer done for the first time.

Practical value of research is associated with determination of definite users categories who benefits from compiling MongoDB from source code.

MONGODB, NOSQL, OPEN SOURCE, BUILD, CRUD, CPU, RAM.

Заява

студента про оригінальність роботи

Максименко О.В.

126-22м-1

Я, Максименко Олександр Володимирович , заявляю, що кваліфікаційна робота на тему: «Дослідження метрик ефективності використання MongoDB, зібраної з вихідного коду»

1. Була підготовлена виключно мною* і не порушує авторські права третіх осіб у відповідності до закону про авторське право.

2. Крім того, я заявляю, що представлена мною для перевірки електронна версія моєї роботи збігається з друкованою.

3. Я підтверджую, що був проінформований про права та обов'язки студента університету та правила, що стосуються перевірки оригінальності наукових робіт. Тому, я заявляю, що згоден на обробку моєї кваліфікаційної** роботи у відповідності до антиплагіатних правил і процедур, а також на архівування її в бази даних інтернет-системи unicheck.com та університету.

«___» грудня 2023 р. _____

(підпис студента)

* Беручи до уваги істотний вклад, внесений з боку керівника наукової роботи

** Під обробкою розуміється порівняння змісту роботи, яка передана на перевірку в Інтернет Систему unicheck.com для виявлення фактів запозичення, генерації Звіту подібності та зберігання документів в базі даних.

ЗМІСТ

ВСТУП

Інформатика як наука займається рядом питань, одне з яких можна сформулювати наступним чином: «Як можна вирішувати ті чи інші обчислювальні та інформаційні задачі максимально ефективно?» [1]. Це питання можна розділити на два: «Як зробити так, щоб програма споживала менше ресурсів?» та «Як зробити так, щоб програма працювала швидше?». Відомо, що чим швидше працює програма, тим більше оперативної пам’яті вона споживає [2]. Отже, на комп’ютерах з обмеженими ресурсами програмні продукти працюватимуть повільніше у порівнянні з більш потужними машинами. Які існують рішення цієї проблеми? Першим рішенням є фізичне додавання оперативної пам’яті, але недоліком цього рішення є апаратне обмеження на обсяг оперативної пам’яті. Інший варіант – додавання віртуальної пам’яті (swap). Однак воно теж не є оптимальним, оскільки процесор зчитує дані з вінчестера повільніше.

Ще одним потенційним рішенням є збірка програм з відкритим вихідним кодом на комп’ютері кінцевого користувача. Потенційно така збірка може надати чимало переваг у порівнянні з використанням вже зібраного бінарного файлу. Усі бінарні пакети збираються за принципом «щоб працювало всюди та у всіх». Отже, пакет під 64-бітну архітектуру зібрано під деякий абстрактний 64-бітний процесор та має під собою тільки те, що буде працювати на усіх 64-бітних процесорах без винятків. З іншого боку, при збірці пакетів з вихідного коду, можна включити підтримку SSE4 для процесорів Intel, 3dNow для AMD тощо. Отже, можна припустити, що збірка програм з оптимізацією під конкретне «залізо» може суттєво пришвидшити деякі з них. Проведемо дослідження, у якому порівняємо продуктивність NoSQL бази даних MongoDB зібраної з вихідного коду та MongoDB завантаженої з офіційного сайту. У якості метрик продуктивності візьмемо швидкість виконання базових CRUD [3] операцій та потребу у фізичних ресурсах (завантаженість процесору та кількість використовуваної оперативної пам’яті комп’ютеру).

Чому саме MongoDB було обрано для дослідження? Використання NoSQL баз даних, включаючи MongoDB, обґрунтовано у різних сценаріях, особливо коли вимагається обробка великих об’ємів даних та гнучкість у моделі даних. Ось декілька причин, з яких використання NoSQL бази даних може бути доцільним:

- гнучкість схеми даних: на відміну від реляційних баз даних, де схема даних строго визначена, NoSQL бази даних надають гнучкість у змінюванні структури даних без необхідності змінювати усю базу даних. Це особливо корисно у середовищі, де вимагається швидка адаптація до вимог, що змінюються;
- масштабуємість: NoSQL бази даних забезпечують гарну горизонтальну масштабованість, що означає, що їх легко масштабувати на більш потужних серверах або навіть розподілити дані по деяким серверам для балансування навантаження;
- великі об’єми даних та висока продуктивність: NoSQL забезпечують ефективне зберігання та обробку великих об’ємів даних. Вони можуть бути особливо корисні у додатках з високими вимогами до продуктивності читання та запису;
- робота з неструктурованими даними: якщо додаток працює з неструктурованими даними такими як JSON документи, документоорієнтовані NoSQL бази даних (MongoDB зокрема) є зручним вибором. MongoDB зберігає дані у форматі BSON (binary json), що спрощує роботу з документами у форматі JSON.
- простота розробки: NoSQL бази даних надають більш простий та інтуїтивний інтерфейс для розробника. Відсутність строгої схеми даних дозволяє більш гнучко взаємодіяти з даними.

MongoDB є найбільш популярним рішенням серед NoSQL баз даних [4]. Для її встановлення для завантаження доступний бінарний файл з офіційного сайту (або з репозиторію пакетів для Linux / MacOS). Але MongoDB також має відкритий

вихідний код [5], що надає можливість зібрати базу даних на комп’ютері кінцевого користувача.

Отже кінцеву задачу можна сформулювати так: як збірка з вихідного коду вплине на швидкість виконання CRUD операцій та потреби у ресурсах для бази даних MongoDB?

1. ПОЧАТКОВІ ВІДОМОСТІ ДОСЛІДЖЕННЯ

1.1. MongoDB у Archlinux

Дослідження проводиться на комп’ютері з Archlinux у якості операційної системи, 8 ГБ оперативної пам’яті та процесором 10th Generation Intel Core i3-1005G1.

Для встановлення MongoDB використано AUR (arch user repository, репозиторій користувача Arch) [6]. Цей репозиторій керується спільнотою користувачів Archlinux. Він містить файли опису пакетів (PKGBUILD), які дозволяють компілювати пакет з вихідного джерела з auppkg, а потім встановлювати його через пакетний менеджер Pacman. AUR був створений для організації та обміну новими пакетами спільноти та допомоги прискоренню включення популярних пакетів до репозиторію спільноти.

Значна кількість нових пакетів, які входять до офіційних репозитаріїв, починаються в AUR. Користувачі можуть вносити свої власні скрипти побудови пакетів (PKGBUILD та пов'язаних файлів) в AUR.

PKGBUILD – це скрипт у системі Archlinux, який використовується для збірки та упаковки пакунків у системі. Ці пакунки використовуються менеджером пакунків Archlinux, відомим як pacman.

PKGBUILD є шелл скриптом, що містить інструкції для збірки та конфігурації пакунків. Такий шелл скрипт використовується утилітою makepkg, що на базі PKGBUILD збирає бінарні пакети. Основні елементи PKGBUILD включають:

- pkgname – назва пакунку;
- pkgver – версія пакунку;
- pkgrel – реліз пакунку;
- pkgdesc – опис пакунку;
- arch – архітектура, для якої призначено пакунок (наприклад x86_64);
- url – веб-сайт або URL-адреса, пов’язана з пакунком;
- license – ліцензія пакунку;

- depends – залежності, які необхідно встановити перед цим пакунком;
- makedepends – залежності, які необхідні лише для збірки;
- source – вихідні файли, або URL-адреси для завантаження;
- sha256sum – контрольні суми для перевірки цілісності завантажених файлів.

Спільнота AUR має можливість голосувати за пакети в AUR. Якщо пакет стає досить популярним, і якщо він має сумісну ліцензію та хорошу техніку упаковки, він може бути внесений до сховища спільноти (стаючи прямо доступним через Pacman або Arch build system (ABS).

AUR має два варіанти для встановлення MongoDB – mongodb-bin (pre-compiled версія бази даних) та mongodb (збірка з вихідного коду відбувається на комп'ютері користувача). Базу даних MongoDB було видалено з офіційного репозиторію пакетів Archlinux у 2019 через зміну ліцензії з AGPLv3 на SSPLv1 [7].

Обидва варіанти бази даних можуть бути встановлені наступним чином:

- клонувати репозиторій бази з AUR;
- у разі необхідності встановити залежності;
- виконати команду makepkg. Результатом її роботи буде пакет у форматі tar.zst;
- встановити пакет командою pacman з параметром -U;

Команду makepkg можна використати з наступними параметрами:

- -i – встановлення пакету після його збірки;
- -s – встановити залежності у разі їх відсутності (не допоможе при відсутності AUR залежностей, в цьому випадку необхідно вручну встановлювати необхідний AUR-пакет);
- -c – видалить проміжні файли, що залишились після збірки. Це корисно при багатократній збірці одного пакету або його оновлені через запобігання додавання старих файлів до нової збірки.

Мати на системі обидва пакети встановленими одночасно неможливо (насправді можливо, але це означатиме великі витрати часу на конфігурування abs,

aur та makepkg), тож спочатку заміри виконаються для mongodb, а потім для mongodb-bin.

Також зазначимо, що при встановленні pacman-ом розмір пакетів mongodb та mongodb-bin відрізняються. Так, розмір пакету mongodb становить 269.19 мегабайт (рисунок 1.1).

Рисунок 1.1 – Розмір пакету mongodb

Натомість розмір mongodb-bin становить лише 217.15 мегабайт (рисунок 1.2).

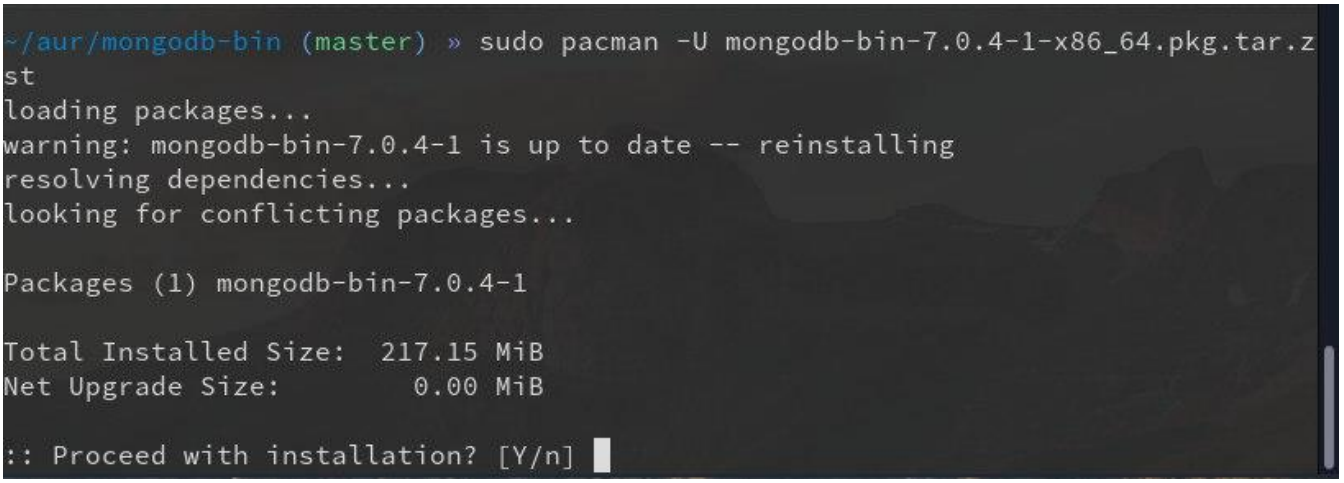

Рисунок 1.2 – Розмір пакету mongodb-bin

1.2. Оптимізація збірки параметрами компілятору

Ціль дослідження є виявлення впливу збірки MongoDB з вихідного коду та підвищення продуктивності MongoDB на етапі збірки за рахунок можливостей компілятору по оптимізації коду. Однак треба мати на увазі, що двійкові файли, скомпільовані під специфічну архітектуру процесору, не будуть правильно працювати на інших машинах. Типові рекомендації щодо використання «тонкого» налаштування компілятору звучить наступним чином: «якщо не можна довести підвищення продуктивності шляхом тестів, скоріш за все, його просто немає» [8].

Опції, що передаються компілятору C/C++ (наприклад, gcc або clang) залежать від змінних оточення CFLAGS, CXXFLAGS та CPPFLAGS. У системі збірки Archlinux makepkg дістає значення змінних оточення з файлу налаштувань makepkg.conf та використовує їх у якості опцій. Значення за замовчуванням обрані таким чином, щоб створювані двійкові програми працювали на широкому діапазоні комп’ютерів.

Слід пам’ятати, що не усі системи збірки використовують змінні, що вказані у makepkg.conf файлі. Наприклад, cmake ігнорує змінні CPPFLAGS. Наслідком цього є те, що часто у файлах PKGBUILD можна побачити різні обхідні рішення для систем збірки, що використовуються для конкретного програмного забезпечення. Також налаштування у файлі Makefile та аргументи у командах компіляції мають пріоритет над значенням makepkg.conf, що може призвести до їх перевизначення.

Компілятор GCC може автоматично виявляти та вмикати безпечні оптимізації, які доступні для конкретної архітектури. Для використання цієї особливості, спочатку слід видалити існуючі -march та -mtune налаштування, а потім додати опцію -march=native. Приклад використання наведено на рисунку 1.3.

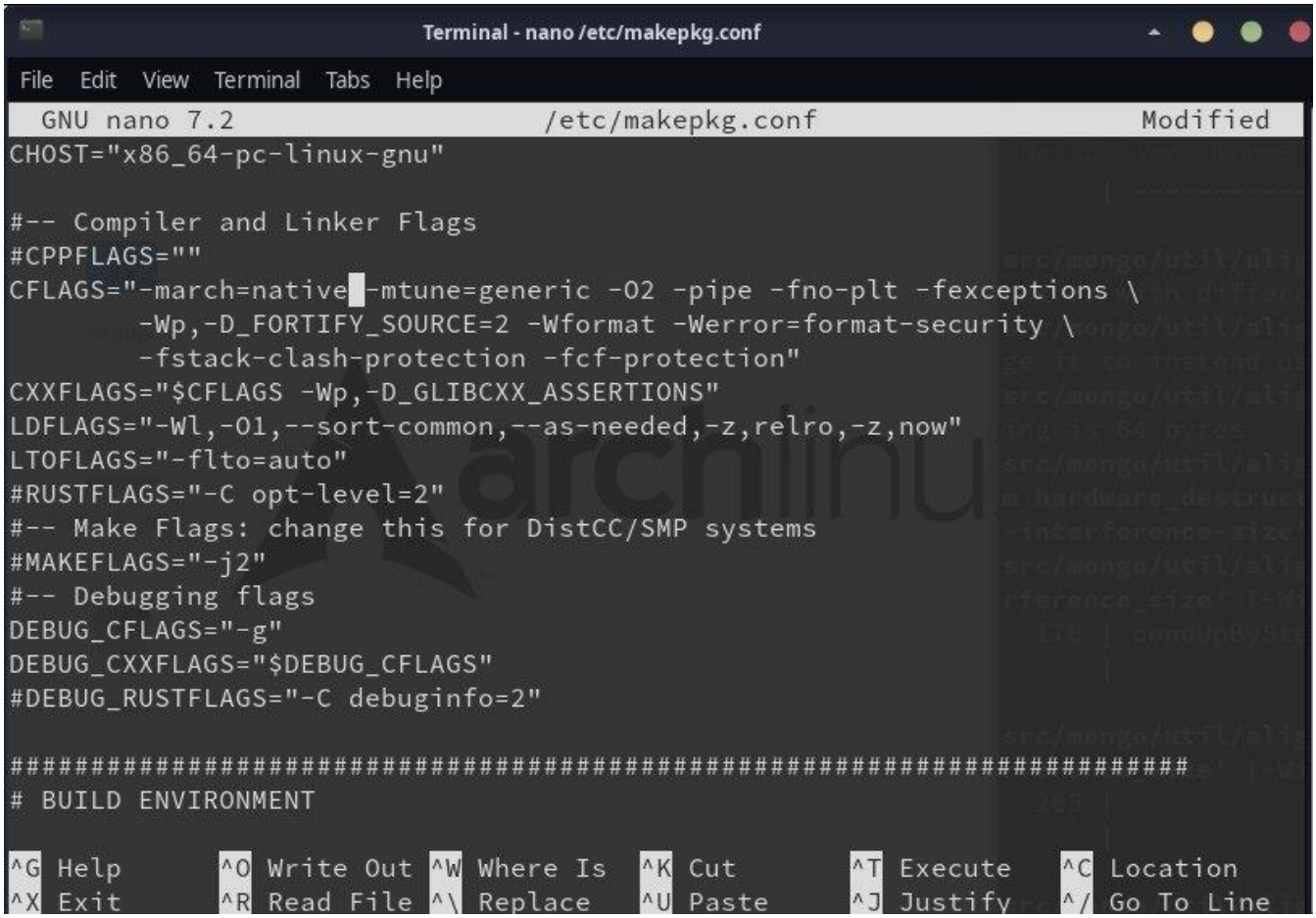

Рисунок 1.3 – Файл /etc/makepkg.conf

Для збірки пакетів за допомогою make використовується змінна середовища MAKEFLAGS, яка визначає додаткові опції для утиліти make. Встановити значення цієї змінної можна також у файлі makepkg.conf.

Користувачі з багатоядерними/багатопроцесорними системами можуть вказати кількість задач, що запускаються одночасно. Це робиться за допомогою утиліти для визначення кількості доступних ядр nproc, наприклад MAKEFLAGS=”-j$(nproc)”. Деякі PKGBUILD перевизначають це значення на -j1, щоб уникнути гонки або тому що багатопотоковість не підтримується від самого початку.

Оптимізація збірки програми за допомогою параметрів компілятору може бути важливою для забезпечення ефективності використання коду. Параметри компілятору використовуються для налаштування різних аспектів оптимізації. Ось деякі загальні параметри, які можна використовувати для оптимізації збірки

програми:

1. Рівень оптимізації (-O):
   - -O0 – без оптимізації;
   - -O1 – базова оптимізація;
   - -O2 – середня оптимізація;
   - -O3 – максимальна оптимізація.
2. Вибір архітектури процесору
   - -march=native – використання оптимальну архітектуру для поточного процесору;
   - -march=architecture – вказує конкретну архітектуру, наприклад -march=core2;
3. Розгортання циклів (-funroll-loops). Може поліпшити швидкість виконання коду за рахунок зменшення накладних витрат на управління циклом
4. Інлайнінг функцій (-finline-functions). Вбудовує виклик коротких функцій безпосередньо в код, що може зменшити накладні витрати на виклик функцій.
5. Видалення недосяжного коду (-feliminate-unused-debug-types, - fwhole-program). Дозволяє видалити непотрібний або недосяжний код.
6. Використання векторних операцій (-ftree-vectorize). Активує автоматичну векторизацію для покращення роботи з масивами.
7. Вибір рівня оптимізації для плаваючої крапки (-ffast-math). Дозволяє компілятору вживати більше агресивних оптимізацій для операцій з плаваючою крапкою.

Ці параметри можна комбінувати для досягнення оптимального результату в конкретному випадку. Однак, важливо пам’ятати, що агресивні оптимізації можуть призводити до непередбачуваної поведінки у деяких випадках, тому їх слід використовувати з обачливістю, особливо при розробці критично важливих програм.

1.3. Прискорення компіляцію за допомогою ccache

Також слід зазначити, що при збірці пакетів з вихідного коду час збірки можна суттєво скоротити за рахунок використання ccache. Ccache – це утиліта для кешування компіляції програмного коду у мові C/C++ та інших. Основна ідея полягає в тому, щоб зберігати результати попередніх компіляцій та використовувати їх замість нової компіляції, якщо вихідний код залишився незмінним. Це може значно прискорити час компіляції проектів, особливо великих програм. Основні переваги ccache включають:

- швидше компілювання: якщо код не змінюється, ccache може використовувати результат попередньої компіляції, збережений у кеш, замість повторної компіляції всього коду;
- зменшення навантаження на ресурси: кеш може значно зменшити навантаження на ресурси комп’ютера, так як менше часу і ресурсів витрачається на повторну компіляцію.
- ефективне використання системних ресурсів: зменшення кількості повторних компіляцій дозволяє ефективніше використовувати ресурси комп’ютера для інших завдань.

Для використання ccache необхідно його встановити та налаштувати для використання. Наприклад, якщо використовується GCC компілятор (крім GCC, ccache є сумісним з Clang компілятором), можна встановити змінні середовища CC та CXX на шлях до ccache (рисунок 1.4).

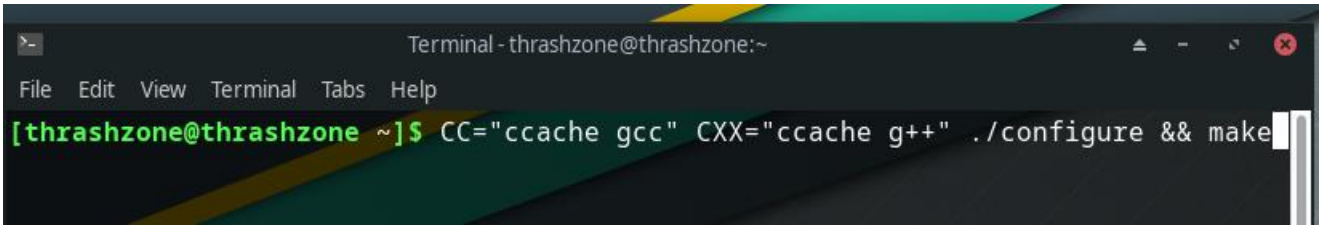

Рисунок 1.4 – Використання ccache

1.4. Існуючі рішення

Ідея оптимізації програм за допомогою збірки з вихідного коду не є новою. Так, в дистрибутиві Gentoo Linux ця ідея реалізована за допомогою USE flags [9]. Взагалі, в основі Gentoo полягає філософія, яка базується на наступних принципах:

- користувальницька налаштованість, тобто користувач вміє обирати, як саме буде налаштована та оптимізована його система. Встановлення та конфігурація кожного пакету здійснюється відповідно до потреб користувача;
- використання системи Portage для управління пакетами. Користувач має можливість збирати та встановлювати програмне забезпечення з вихідного коду, що дозволяє максимально налаштувати пакунок під конкретне апаратне забеспечення та потреби системи;
- оптимізація під конкретний апаратний засіб, тобто користувач має можливість налаштовувати параметри компіляції пакунків, щоб вони оптимально працювали на конкретному обладнанні. Це може включати вибір архітектури процесору, рівні оптимізації та інші параметри, що впливають на продуктивність;
- принцип відсутності визначеного стану (Rolling Release), тобто користувач може встановлювати та оновлювати пакети в будь-який час без необхідності очікування випуску нової версії дистрибутиву;
- документація та освіта: Gentoo покладає великий наголос на Gentoo Handbook та інші ресурси для забезпечення користувачів інформацією, яка допомагає їм розуміти систему та ефективно використовувати її можливості;
- співпраця та спільнота: Gentoo побудована на відкритій спільноті, де користувачі можуть долучитися до розробки та підтримки. Спільнота грає дуже важливу роль у розвитку та підтримці дистрибутиву.

Ці принципи роблять Gentoo привабливим для тих, хто шукає високий рівень

користувальницької налаштованості та бажає більш активно управляти своєю Linux-системою.

Повернемось до USE Flags. Під час встановлення Gentoo, користувач має вибір в залежності від робочого середовища. Встановлення серверу відрізняється від встановлення робочої станції, а робоча станція геймеру відрізняється від робочої станції 3D візуалізатора.

Це відноситься не тільки до вибору пакетів, а й до функцій, які певні пакети повинні підтримувати. Наприклад, якщо немає потреби в OpenGL, навіщо треба встановлювати та підтримувати OpenGL, або збирати підтримку OpenGL для встановлюємих пакетів? Або якщо користувач використовую XFCE у якості оболонки робочого столу, навіщо збирати підтримку KDE для встановлюємого пакету?

Щоб допомогти користувачам вирішити, що їм треба встановлювати/активувати, а що ні, у Gentoo запровадили концепцію USE Flags.

В Gentoo USE Flag – це механізм контролю функціональних опцій пакетів, який надає користувачам можливість вказати, які функції чи підтримка мають бути включені або виключені для конкретного програмного забезпечення при його збірці та інсталюванні. Це надає користувачам велику гнучкість і контроль над конфігурацією своєї системи. Коректне використання USE Flags адаптує системи під конкретні вимоги її користувача.

Налаштувати USE Flags можна у файлі /etc/portage/make.conf. Так, на рисунку 1.5 наведено налаштування компіляції програм з підтримкою GTK та GNOME, але без підтримки QT5 та KDE.

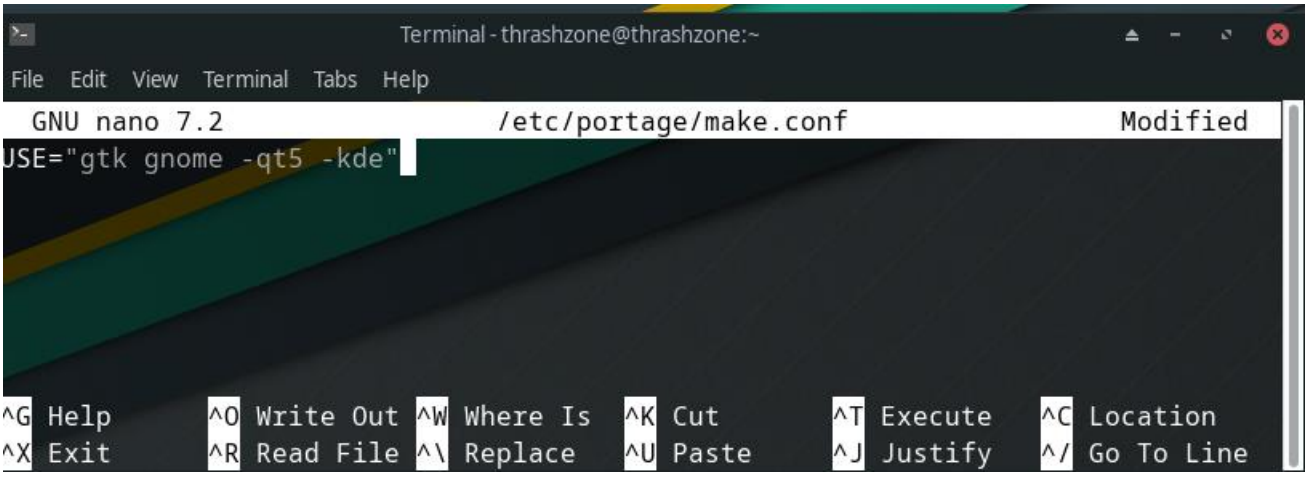

Рисунок 1.5 – Приклад використання USE Flags у файлі make.conf

Також є можливість використання конкретних USE Flag для конкретного пакету за рахунок їх перевизначення. Так, встановлення пакету з протилежними значенням USE зображено на рисунку 1.6. Інший спосіб задати конкретні USE Flags для пакету – використання package.use файлів. На рисунку 1.7 наведено приклад з використанням специфічних USE Flags для пакетів dev-lang/php та media-video/vlc за допомогою package.use файлу.

Рисунок 1.6 – Перевизначення USE Flags для конкретного пакету

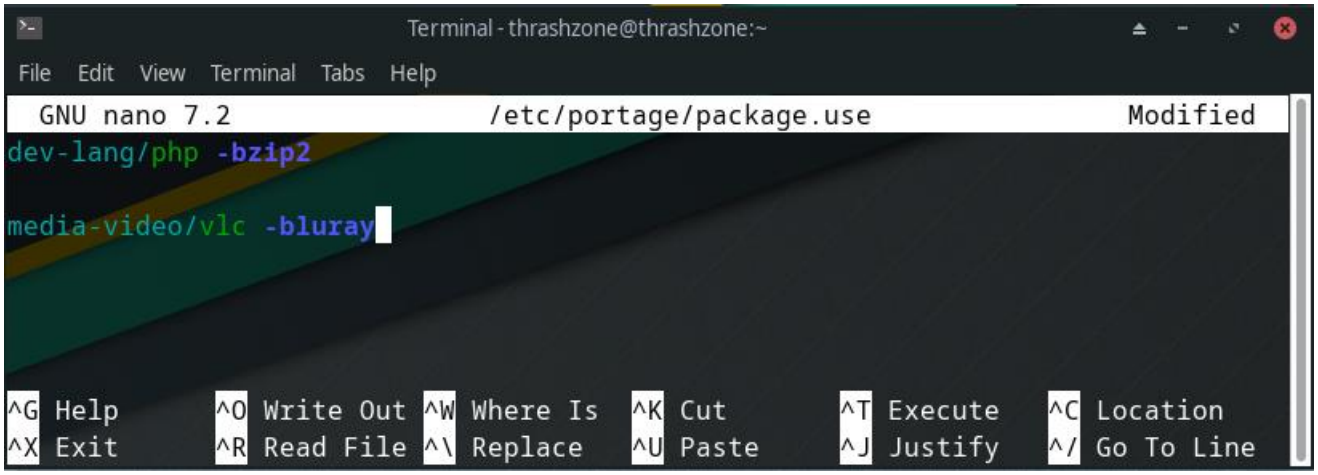

Рисунок 1.7 – Використання package.use

До сих пір мова йшла про GNU компілятори (gcc та g++), але виходячи з докладу Університету штата Нью-Йорк, у окремих випадках можна добитися більш високих показників продуктивності, якщо для кожної задачі використовувати доцільний компілятор [10]. Так, згідно статті використання Intel компіляторів має сенс у наступних ситуаціях:

- оптимізація під архітектуру Intel (Intel компілятор володіє оптимізаціями, спрямованими на архітектури процесорів виробництва Intel);
- підтримка специфічних технологій Intel (Intel компілятор підтримує специфічні функції та інструкції, які можуть бути доступні тільки на процесорах Intel. При необхідності таких функцій використання Intel компілятору є необхідністю);
- підтримка Intel Math Kernel Library (підтримка бібліотеки, що надає ефективні реалізації математичних алгоритмів, таких як лінійна алгебра, оптимізовані під архітектуру Intel).

Дослідження виконується вимірювальними та порівняльними методами загального емпіричного методу. Спочатку для кожної з баз виконується однаковий набір CRUD операцій та вимірюється кількість спожитих ресурсів разом зі

швидкістю виконання (вимірювання), потім для кожної бази ці дані зрівнюються (порівняння). Актуальність дослідження обґрунтована тим, що потенційним результатом дослідження є метод пришвидшення роботи MongoDB без втручання в її вихідний код та без втручання в апаратну частину комп’ютеру (зміни процесору на більш потужний, додавання оперативної пам’яті). Разом з тим наразі немає аналогічного дослідження щодо бази даних MongoDB.

2. ГЕНЕРАЦІЯ ТЕСТОВИХ ДАНИХ

Для порівняння маємо провести одні й ті ж тести для MongoDB встановленої з офіційного бінарного файлу та для MongoDB, що була скомпільована на комп'ютері кінцевого користувача. Але для проведення дослідження необхідні великі об'єми тестових даних. Об'єм даних може вважатися великі (big data), якщо має щонайменше 1000000 рядків [11] (або його розмір є не меншим за 1 гігабайт [12]). Створення такого об'єму даних вручну займе дуже багато часу. Отже маємо необхідність у створенні скрипту, який згенерує необхідну кількість даних автоматично.

2.1. Схеми алгоритмів основних обчислювальних процесів

У якості джерела даних використаємо сервіс https://randomuser.me. Це безкоштовне API з відкритим вихідним кодом для генерації випадкових користувальницьких даних, щось на кшталт Lorem ipsum. Сервіс надає можливість вказати необхідну кількість згенерованих користувачів та формат, у якому буде відповідь (JSON (варіант за замовчуванням), XML, CSV та YAML). Щодо недоліків, randomuser накладає обмеження на максимальну кількість документів за один запит (не більше 5000) та на кількість документів на хвилину часу.

Використовуваний сервіс є RESTful. Взагалі REST (Representational State Transfer) є структурованим підходом до проектування архітектури програмного забезпечення, який використовується для побудови веб-сервісів (як https://randomuser.me). REST є стандартом, який визначає правила та обмеження для створення легких, масштабованих, та добре розширюваних веб-сервісів. Основні принципи REST описані Ройом Філдінгом в його докторській дисертації [13]. 2000 року.

Основні характеристики RESTful сервісів включають:

- ресурси: у REST дані представлені як ресурси (наприклад, об’єкти чи дані), які можуть бути доступні або модифіковані за домогою URI (Uniform Resource Identifier);
- представлення: ресурси можуть бути представлені у різних форматах, таких як JSON або XML, клієнт та сервер можуть обмінюватися представленнями ресурсів;
- стан: стан ресурсу представляється між клієнтом і сервером у відповідних запитах. Клієнт може зберігати або оновлювати стан ресурсу;
- одноразові запити: RESTful архітектура є безстановою, що означає, що кожен запит від клієнта до сервера має весь необхідний контекст для розуміння і обробки. Ніякого стану не зберігається на сервері між запитами від клієнта;
- операції CRUD є частими у RESTful сервісах і відображають дії, які клієнт може виконувати з ресурсами.
- HTTP методи: REST використовує стандартні HTTP методи, такі як GET, POST, PUT, DELETE для виконання різних операцій над ресурсами;
- безпека: RESTful сервіси можуть використовувати засоби забезпечення безпеки HTTP, такі як HTTPS (HTTP Secure), для захисту обміну між клієнтом і сервером.

Підкреслимо, що на відміну від того ж SOAP, у REST HTTP є активним учасником, а не просто транспортним засобом. Взагалі, HTTP є мережевим протоколом прикладного рівня [14], що служить для передачі гіпертекстових даних через мережу, основним призначенням якого є передача інформації між клієнтом і сервером. Цей протокол використовується для взаємодії з веб-сторінками, завантаження ресурсів, виконання запитів до серверів та отримання відповідей.

Наведемо основні характеристики HTTP:

1. Клієнт-серверна архітектура (HTTP використовує модель клієнт-сервер, де клієнт робить запит, а сервер відповідає на цей запит).
2. Безстанівість (кожен запит від клієнта до серверу є незалежним, і сервер не зберігає інформацію про попередні запити). Це полегшує масштабування та робить систему більш простою.
3. Методи HTTP. Протокол визначає різні методи, які вказують, яку дію слід виконати для заданого ресурсу. Найпоширеніші методи:
   - GET (отримати дані з серверу);
   - POST (відправити дані на сервер);
   - PUT (оновити існуючі дані);
   - DELETE (видалити існуючі дані на сервері).
4. URI (Uniform Resource Identifier). URI визначає ідентифікатор ресурсу, з яким взаємодіє клієнт або сервер. URL (Uniform Resource Locator) є підтипом URI та вказує на місцезнаходження ресурсу.
5. Заголовки. HTTP використовує заголовки для передачі додаткової інформації про запит або відповідь. Наприклад, заголовок Content-Type вказує на тип вмісту (текст, зображення, JSON тощо).
6. Статус-коди. Сервер повертає статус код у відповідь на запит, який вказує на результат обробки запиту (наприклад, 200 OK, 404 Not Found, 500 Internal Server Error).
7. Безпека. HTTPS (HTTP Secure) – це розширення HTTP, яке використовує шифрування за допомогою протоколу TLS/SSL для захисту передачі даних між клієнтом і сервером.

HTTP використовується в широкому спектрі застосувань, включаючи веб-додатки, API, мобільні додатки та інші системи, які вимагають обміну даними через мережу.

Для написання програми використаємо NodeJS та бібліотеки axios [15] (бібліотека для HTTP запитів, що побудована на концепції промісів [16]) і sleep [17]. Вибір NodeJS у якості платформи зумовлений наступними факторами:

- асинхронна та подійно-орієнтована модель. Така модель виконання коду дозволяє обробляти багато запитів без блокування потоку виконання [18], що сприяє високій продуктивності та ефективному використанню ресурсів. Саме такий підхід у виконанні є доцільним у створеній програмі, адже необхідно одночасно робити запити до API, обробляти їх та передавати у потік запису [19] у файл;
- висока ефективність: оскільки NodeJS використовує V8, що було розроблено командою Google для браузера Chrome, він демонструє високу продуктивність завдяки швидкій компіляції JavaScript у машинний код;
- багата екосистема: NodeJS має велику бібліотеку модулей, яка включає в себе багато інструментів і фреймворків для швидкої розробки додатків. Розробники можуть використовувати готові модулі для вирішення різних задач. Така особливість стає у нагоді при розробці, оскільки буде використано бібліотеки NPM репозиторію для HTTP запитів та зупинки основного потоку програми на заданий період часу.
- JSON як стандарт даних: при імпорту даних у MongoDB використовується формат даних JSON. JavaScript та NodeJS добре підходять для роботи з цим форматом, оскільки вони нативно підтримують об'єкти JSON.
- легка масштабованість: NodeJS спрощує горизонтальне мастабування, оскільки його асинхронний характер дозволяє легко розподіляти завдання на багато ядр та серверів.

Опишемо алгоритм роботи генератору. Створюємо вихідний потік у файл, куди дані будуть записані. Далі слід зробити запит до RESTful сервісу для отримання даних. Якщо відповідь сервісу має статус код 200 – записуємо дані у вихідний потік та чекаємо хвилину що зробити наступний запит, якщо відповідь 429

(Too Many Requests) – чекаємо 3 хвилини на повторюємо запит. Коли у вихідному потоці буде достатня кількість документів – генератор запише дані з потоку у файл та завершить свою роботу.

Схематично роботу програми можна зображено на рисунку 2.1:

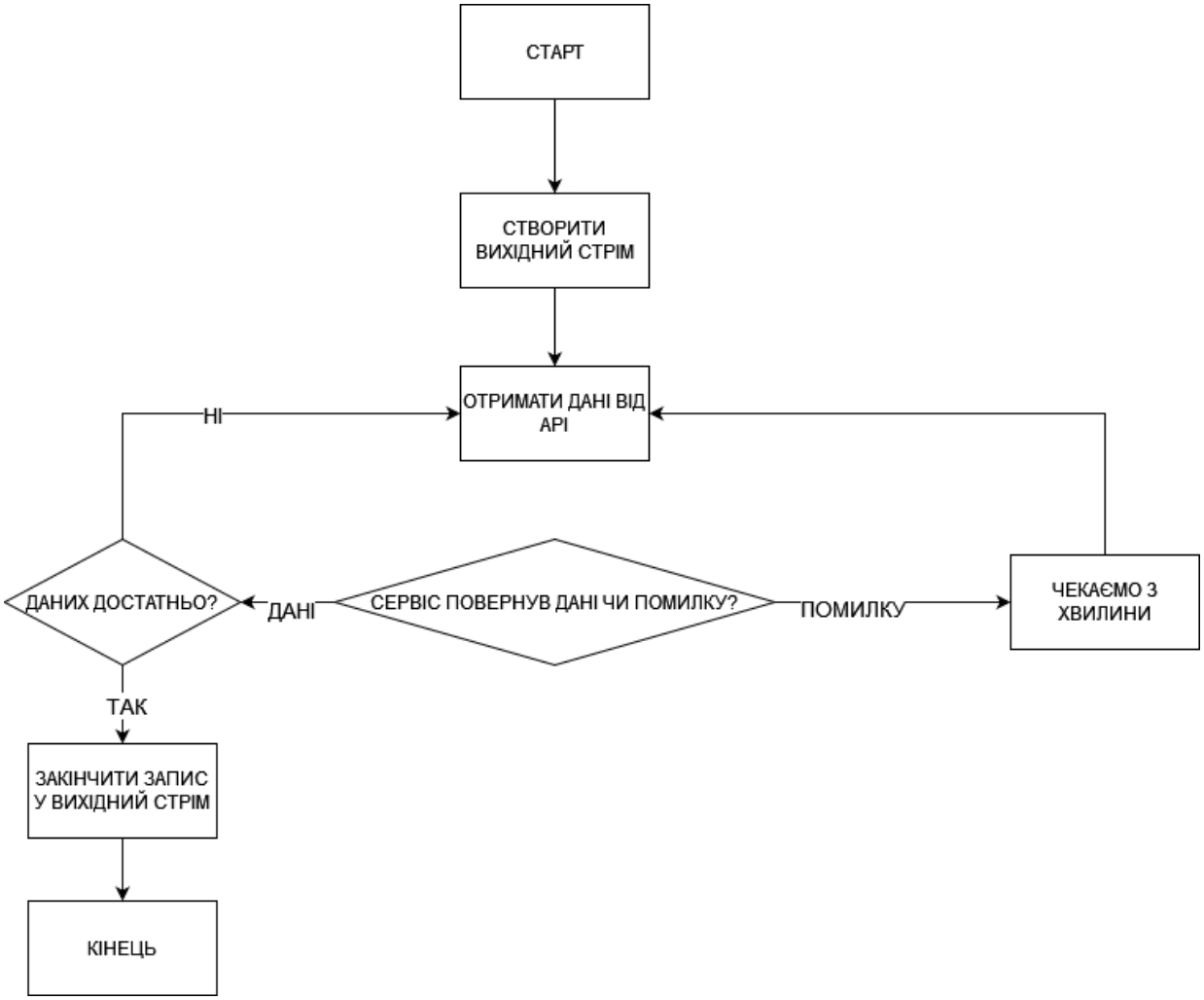

Рисунок 2.1 – Схема роботи генератору даних

2.2. Керівництво користувача програмою

Для користування програмою користувачу необхідно встановити NodeJS, перейти у корньову директорію (там де файли generator.js та package.json) та

виконати команди node install (для встановлення необхідних залежностей) та node start (для запуску генерації). За необхідності збільшення або зменшення обсягу генеруємих даних користувач може змінити змінні numberOfUsers та max у файлі generator.js. Ці змінні можна інтерпретувати наступним чином: «згенеруй numberOfUsers max разів». Тобто якщо numberOfUsers буде 10, а max = 5, то буде згенеровано 10 користувачів 5 раз (50 користувачів загалом). Використання даних команд та стандартний вивід програми наведено на рисунку 2.2.

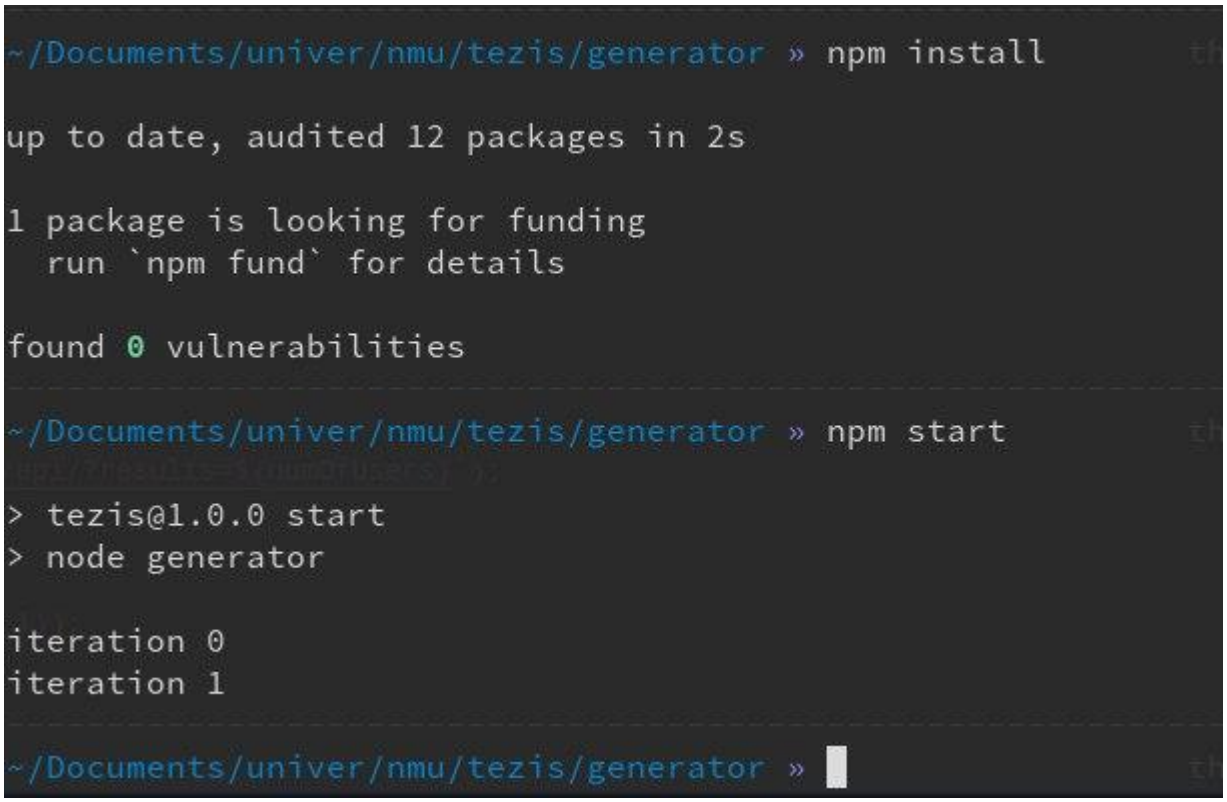

Рисунок 2.2 - Встановлення та запуск програми з її виводом

Слід зауважити, що час між ітераціями необхідний. У випадку частих запитів API поверне помилку 429 (Too Many Requests) (рисунок 2.3).

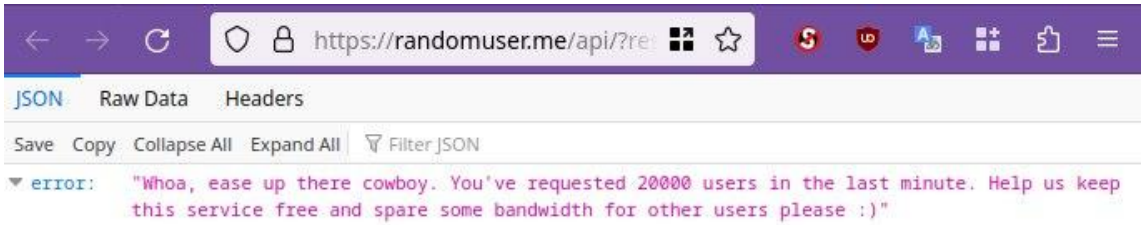

Рисунок 2.3 – Помилка при частих запитах

2.3. Результати тестування програми

У результаті роботи програми буде створено файл data.json у тій же директорії, де знаходиться файл generator.js. Частково вміст згенерованого файлу наведено на рисунку 2.4.

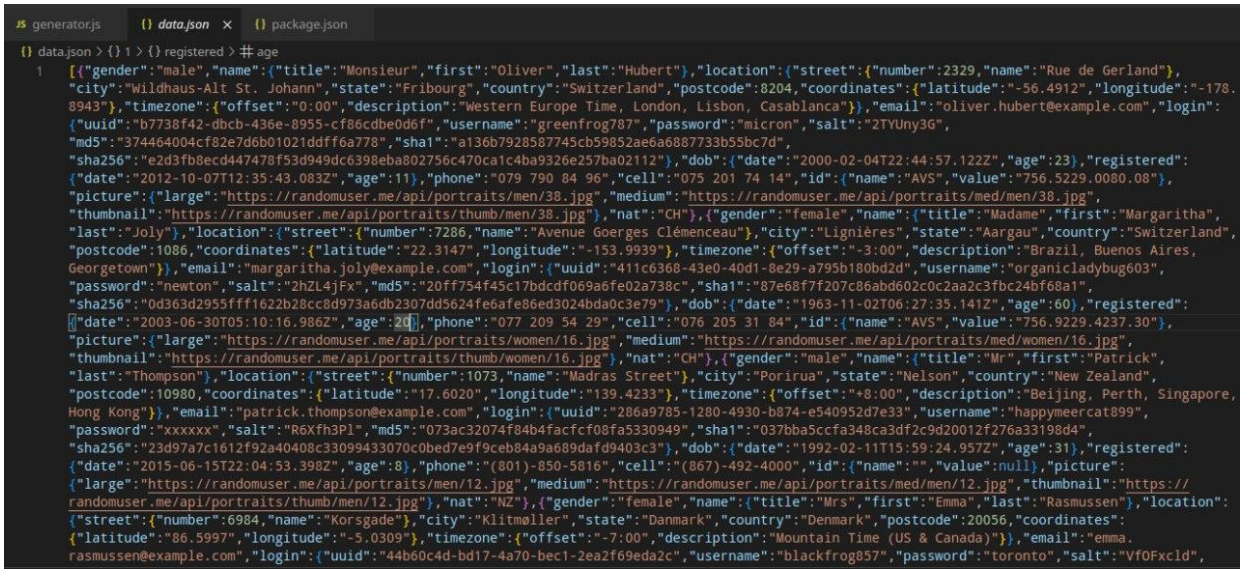

Рисунок 2.4 – Результат роботи генератору

Дані генеруються у форматі JSON (JavaScript Object Notation), який став повсюдним стандартом представлення структурованих даних на базі синтаксису

JavaScript [20]. Популярність JSON зумовлена його легковажністю та зручністю для читання та запису для людини. Він легко парситься та генерується комп'ютером [21]. Крім того, MongoDB зберігає дані у форматі BSON, тобто базі легко працювати з JSON форматом.

Пакет mongo-tools має у своєму складі утиліту mongoimport [22]. mongoimport – це утиліта командного рядка для MongoDB, яка дозволяє імпортувати дані з різних форматів у базу даних MongoDB. Зазвичай її використовують для завантаження великих обсягів даних у MongoDB за допомогою командного рядка. Основні параметри та опції:

- --db – вказує ім'я бази даних, у яку дані імпортуються;
- --collection – ім'я колекції, у яку дані імпортуються;
- --file – шлях до файлу, який містить дані для імпорту;
- --host – хост MongoDB сервера;
- --port – порт MongoDB сервера;
- --username та --password – ім'я користувача та пароль для аутентифікації на сервері MongoDB;
- --jsonArray – опція, що зазначає, що вхідний файл є масивом JSON-документів, а не окремим JSON-документом.
- --type – визначає формат вхідних даних (csv, json, tsv, тощо).

З її допомогою в майбутньому згенерований файл будемо імпортувати у базу даних. Команда має наступний вигляд: mongoimport –db=DATABASE_NAME –collection COLLECTION_NAME –type json –file /path/to/file –jsonArray

Перевіримо, чи є робота генератору коректною та чи валідним JSON документом є згенерований ним файл. Для цього згенеруємо файл з 10 елементами та імпортуємо його до БД командою mongoimport (рисунок 2.5).

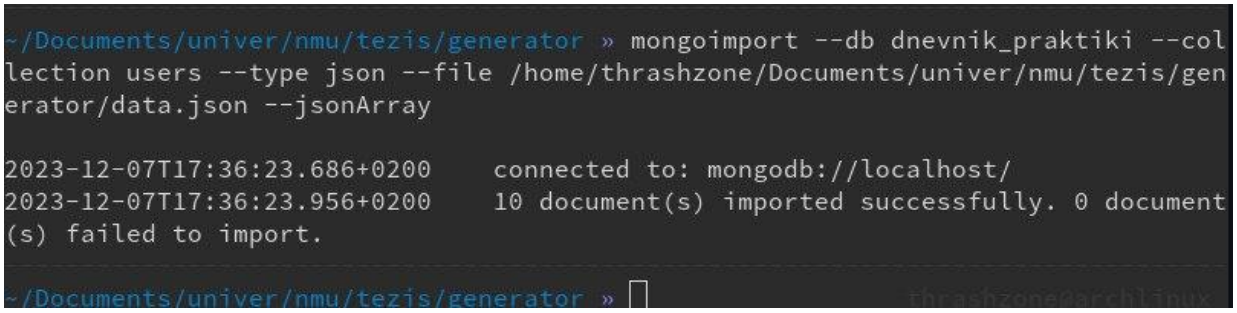

Рисунок 2.5 – Імпорт згенерованого документу до БД

У якості фінальної перевірки спробуємо імпортувати невалідний JSON файл. Зміст битого файлу наведено на рисунку 2.6.

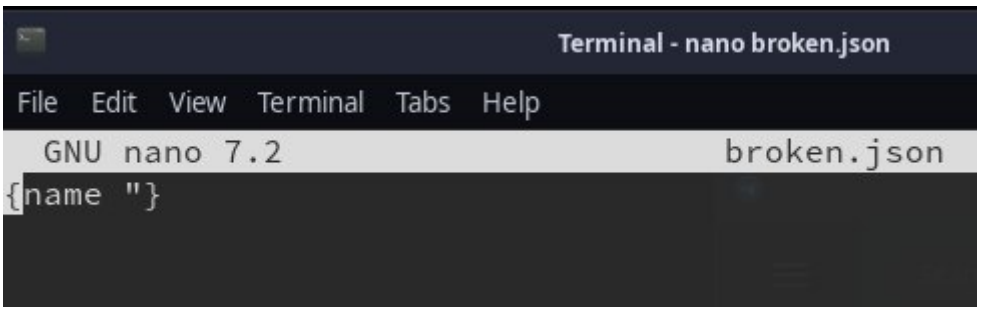

Рисунок 2.6 – Битий JSON файл

Вимогами до файлів JSON є:

- об’єкт повинен розпочинатися і закінчуватися фігурними дужками {};
- масив повинен розпочинатися і закінчуватися квадратними дужками [];
- ключі та значення об’єктів розділені двокрапкою;
- елементи масиву розділені комою;
- строки повинні бути вказані у подвійних лапках;
- числа, булеві значення та null вказуються без лапок.

Битий JSON не відповідає вимогам до файлів цього формату, що спричинило помилку mongoimport (рисунок 2.7).

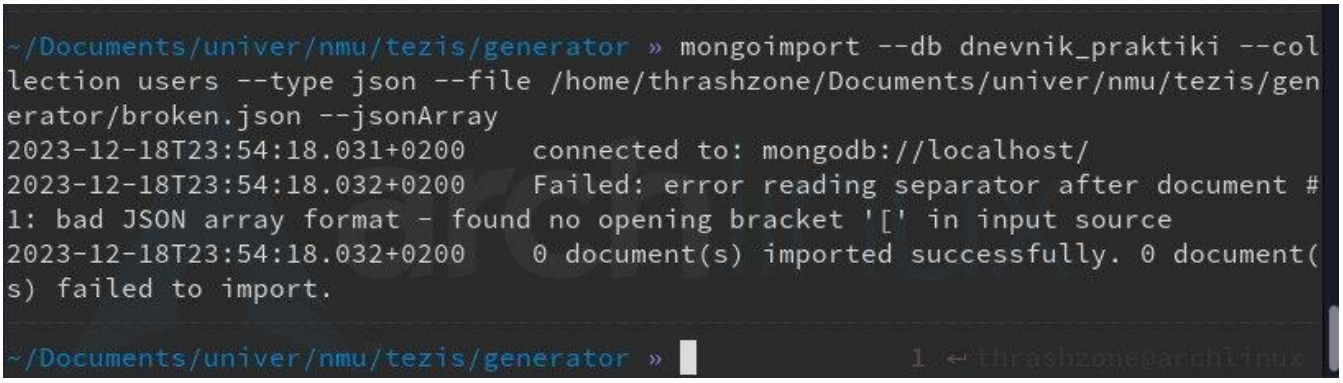

Рисунок 2.7 – Помилка імпорту невалідного JSON

Помилка битого JSON є гарантією коректності роботи генератору та валідності його вихідних файлів.

Далі перевіримо, чи існує така колекція в базі та скільки в ній документів (рисунок 2.8).

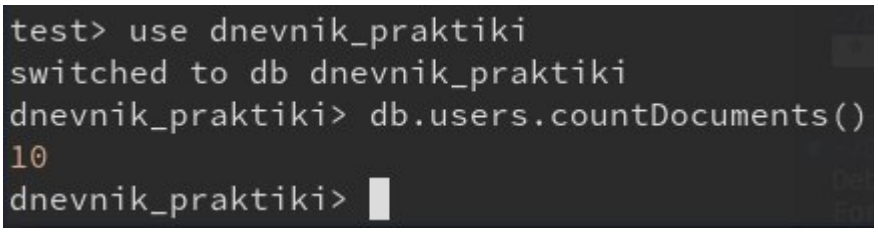

Рисунок 2.8 – MongoDB має імпортовані документи у новій колекції

2.4. Результат роботи генератору

У результаті роботи скрипту було згенеровано дані, що будуть використовуватися у дослідженні (саме над ними будуть виконуватися операції

вставки, зміни та видалення). Перед генерацією даних слід перевірити, чи достатньо вільного місця на жорсткому диску та наявність інтернет з’єднання. Так, файл, що містить 500000 згенерованих користувачів займає 515.8 мегабайтів (рисунок 2.9) на жорсткому диску.

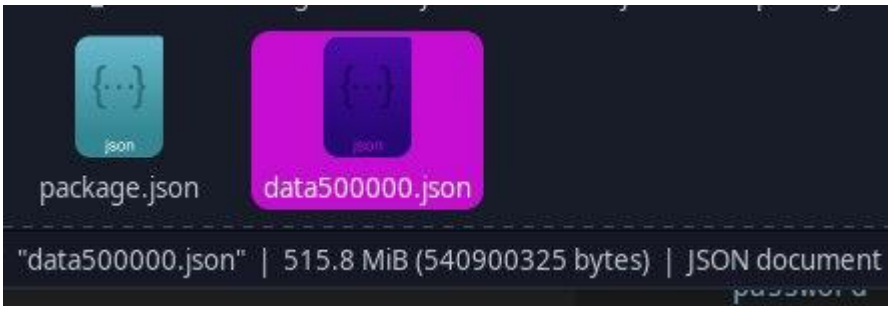

Рисунок 2.9 – Розмір файлу з 500000 користувачами

Оскільки усі користувачі мають однакову структуру, можна стверджувати, що файл з 1000000 користувачів буде займати більше 1 ГБ.

Згенеровані дані можуть бути інтерпретовані як Big Data. Big Data – це термін, що вказує на обсяги даних, які великі за розміром, та які складно аналізувати за допомогою традиційних методів обробки даних. Основні характеристики великих даних включать 3V – об’єм (Volume), розмах (Velocity) і різноманітність (Variety). Розберемо кожну з цих V детальніше:

- Об’єм (Volume) – великі дані відзначаються великим обсягом інформації, яку не можна легко обробити та зберігати за допомогою традиційних засобів. Об’єм великих даних може бути від гігабайту до зеттабайт і більше.
- Розмах (Velocity) – це швидкість збільшення даних. Дані можуть надходити в режимі реального часу, і швидкість потоку даних може бути великою. Це вимагає швидкого та ефективного аналізу для отримання корисної інформації.
- Різноманітність (Variety) – великі дані можуть бути в різних форматах:

структуровані, наприклад, таблиці баз даних, або неструктуровані, такі як текстові документи, зображення, аудіо- та відеофайли. Обробка різноманітності даних є однією з ключових викликів у роботі з великими даними.

Поза основними характеристиками 3V також виділяють інші характеристики, такі як “Value” (вартість) – здатність здобути цінні знання з великих даних, і “Veracity” (достовірність) – важливість точності та надійності даних.

Обробка та аналіз великих даних включає в себе використання спеціалізованих технологій та інструментів, таких як Hadoop, Apache Spark, NoSQL бази даних, машинне навчання, інструменти візуалізації даних та інші.

Окремим підходом є парадігма MapReduce [23]. Ця парадигма є моделлю програмування та обчислювальною архітектурою, яка призначена для обробки та генерації великих об’ємів даних на розподілених системах. Ця парадигма була вперше визначена Google та стала основою для розробки систем обробки великих даних, таких як Apache Hadoop.

Основні етапи парадигми MapReduce включать два основних кроки (Map та Reduce) та один проміжний (Shuffle and Sort):

- Map (відображення): функція Map приймає вхідні дані та створює набір пар «ключ-значення». Це перетворює вхідні дані на список пар ключ-значення, які в подальшому легше обробляти та розподіляти.
- Shuffle and Sort (перемішування та сортування): дані групуються за ключем та відправляються на інші вузли для подальшої обробки. Цей етап є підготовчим для етапу Reduce.
- Reduce (Зведення): на вхід приймаються вже відсортовані та перемішані дані. Використовуючи ключи, над групами виконуються певні операції. Результатом є підсумковий набір пар «ключ-значення», що представляє певне обчислення для певной підмножини вхідних даних.

За допомогою цих етапів Map та Reduce можна виконувати різноманітні завдання обробки та аналізу великих даних, такі як фільтрація, сортування,

підрахунок, агрегація та багато інших. Однак, важливо відзначити, що не всі завдання ідеально підходять для моделі MapReduce, і іноді інші архітектури або інструменти для обробки великих даних можуть бути більш ефективними.

Apache Hadoop є однією з реалізацій парадигми MapReduce та широко використовується для обробки великих даних у розподілених системах.

Крім даного дослідження, великі дані застосовуються у різних галузях, включаючи бізнес, медицину, науку, фінанси, телекомунікації, інтернет речей (IoT), та багато інших. Аналіз великих даних може допомогти виявляти тренди, отримувати нові знання та приймати стратегічні рішення на основі великого об’єму даних.

Вихідний код генератору наведено у додатку Б.

3. ПОРІВНЯННЯ ВИКОРИСТОВУЄМИХ РЕСУРСІВ

Для уникнення фактору «холодного старту» під час замірів, після кожного заміру комп’ютер буде перезапущено (отже дані, що було використано від попереднього старту не зберігаються та програма має почати роботу з нульового стану. Це може вплинути на швидкість запуску та загальну продуктивність, але ця умова буде однаковою як для баз, зібраних вручну, так і для завантажених).

У якості інструменту моніторингу використаємо утиліту htop [24]. htop - це інтерактивна утиліта для моніторингу системи в терміналі в реальному часі. Вона є поліпшеною альтернативою для стандартного інструмента top. htop надає зручний інтерфейс користувача та багато функцій для візуалізації та керування процесами системи. Ось деякі ключові особливості htop:

- зручний інтерфейс: htop відображає список процесів у вигляді таблиці, де можна переглядати і сортувати інформацію;
- кольорове відображення: Різні кольори використовуються для підкреслення різних типів інформації, що полегшує сприйняття;
- інтерактивне керування: Клавіші стрілок та ряд інших клавіш дозволяють вам взаємодіяти з htop, здійснювати дії, такі як завершення процесів, зміна пріоритетів та інше;
- графіки завантаження: htop відображає графіки для візуалізації використання CPU, пам'яті та інших ресурсів системи;
- пошук та фільтрація: Можливість швидкого пошуку та фільтрації процесів за ім'ям, ідентифікатором користувача та іншими параметрами;
- підтримка багатопроцесорних систем: htop може ефективно працювати на системах з багатьма ядрами;
- підтримка сесій: htop може відновлювати стан попереднього сеансу, що зручно для моніторингу змін протягом часу.

Скористаємося цією утилітою після старту бази даних для порівняння

використовуваних ресурсів скомпільованою базою (далі – C (compiled)) (рисунок 3.1) та завантаженої (далі – P-C (pre-compiled)) (рисунок 3.2).

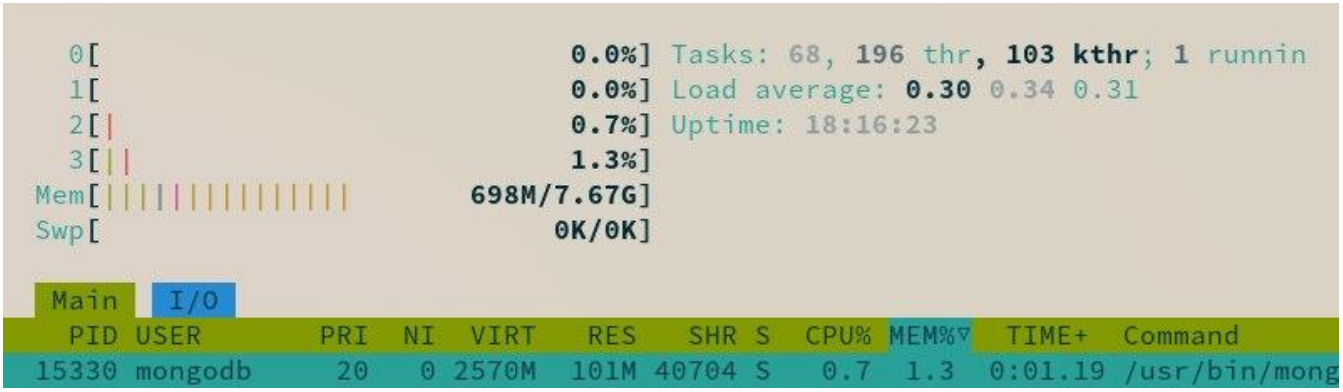

Рисунок 3.1 – Вивід htop після старту бази C

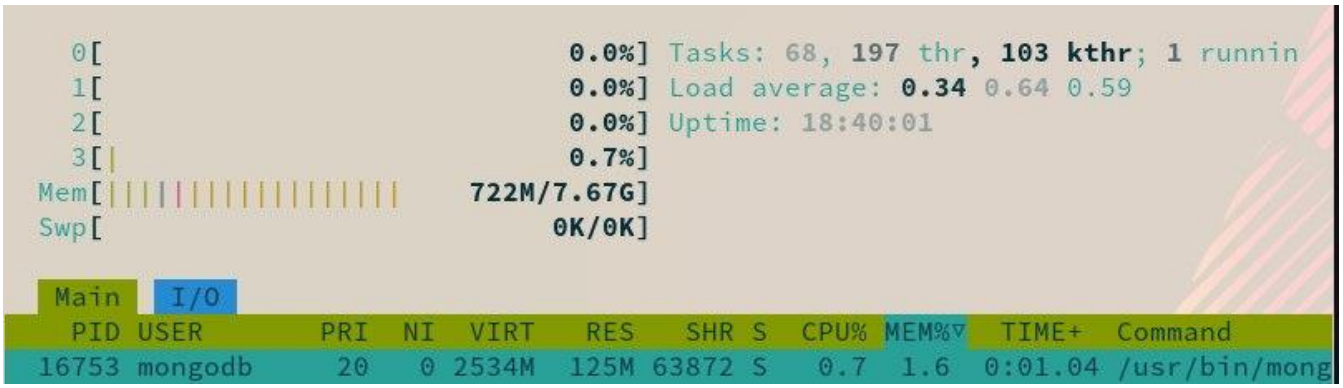

Рисунок 3.2 – Вивід htop після старту бази P-C

Пояснимо вивід утиліти. Вивід htop складається з кількох розділів, які надають інформацію про використання ресурсів та процеси системи. Ось основні елементи:

1. Головний рядок: загальна інформація про систему.
2. Таблиця процесів:
   - PID (Process ID): унікальний ідентифікатор процесу.

- USER: користувач, в якого виконується процес.
- PRI (пріоритет процесу).
- NI: число, яке визначає позначення для підсилення або зменшення пріоритету процесу.
- VIRT, RES, SHR: віртуальна, резидентна та об’єм пам’яті, яку використовує процес.
- S: статус процесу (наприклад, “R” – запущено, “S” - зупинено).
- CPU%: відсоток використання CPU процесом.
- MEM%: відсоток використання оперативної пам'яті процесом.
- TIME+: загальний час виконання процесу.

Крім цього нас буде цікавити значення «Load average». Воно представляє собою середнє навантаження за останні 1, 5 та 15 хвилин. Зазвичай, якщо значення навантаження близько або менше кількості ядр у процесорі, то система вважається незавантаженою. Якщо значення вище кількості ядр, це може вказувати на завантажену систему, і може виникнути затримка у відповіді на запити.

Як видно з рисунків 2.1 та 2.2, вже після старту бази даних, C версія споживає менше оперативної пам’яті та ресурсів процесора, ніж P-C версія.

3.1. Порівняння використовуваємих ресурсів при вставці

Іпортуємо згенеровані дані у базу даних. Для цього скористаємось утилітою mongoimport.

У кінцевому виді команда матиме наступний вигляд:

```
mongoimport –db=research –collection users –type json –file
/home/thrashzone/Documents/univer/nmu/tezis/generator/users.json –jsonArray
```

Під час імпорту скористаємось утилітю htop для моніторингу ресурсів процесом mongodb. У результаті маємо менші показники потреби у RAM та CPU ресурсів для бази даних C (рисунок 3.3) у порівнянні з базою даних P-C (рисунок 3.4). Повторим ці операції до десяти разів та у результаті маємо меншу в середньому

на 1.2% потребу у ресурсах RAM та 1.1% у ресурсах CPU для C бази даних у порівнянні з P-C базою даних.

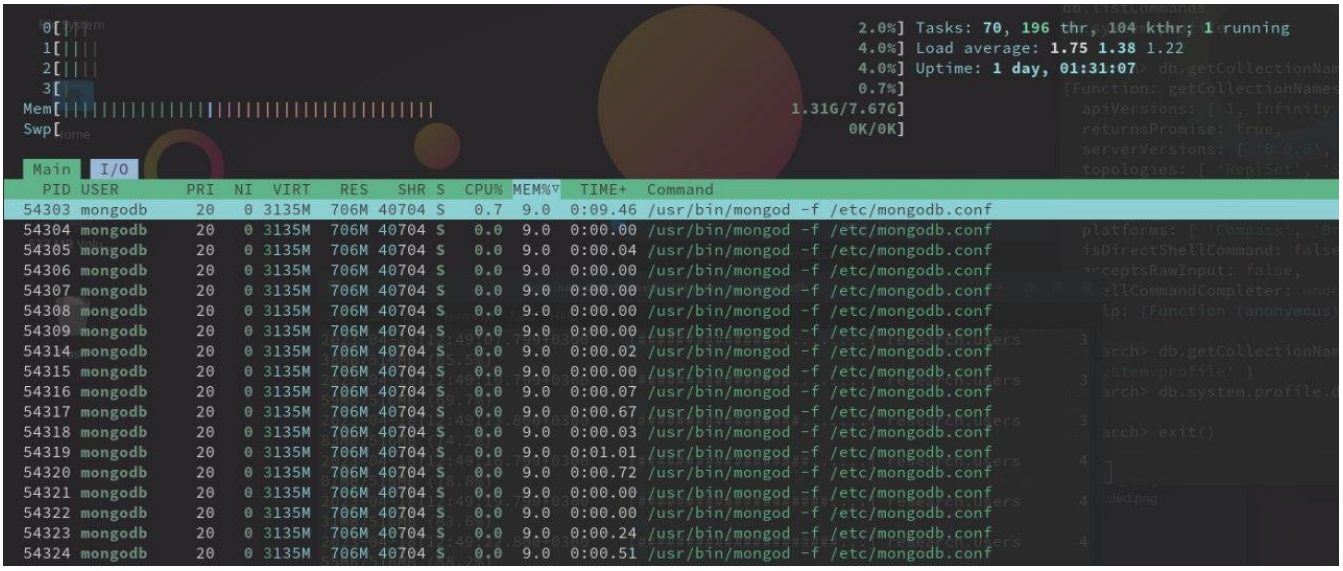

Рисунок 3.3 – Вивід htop під час вставки даних у базу даних C

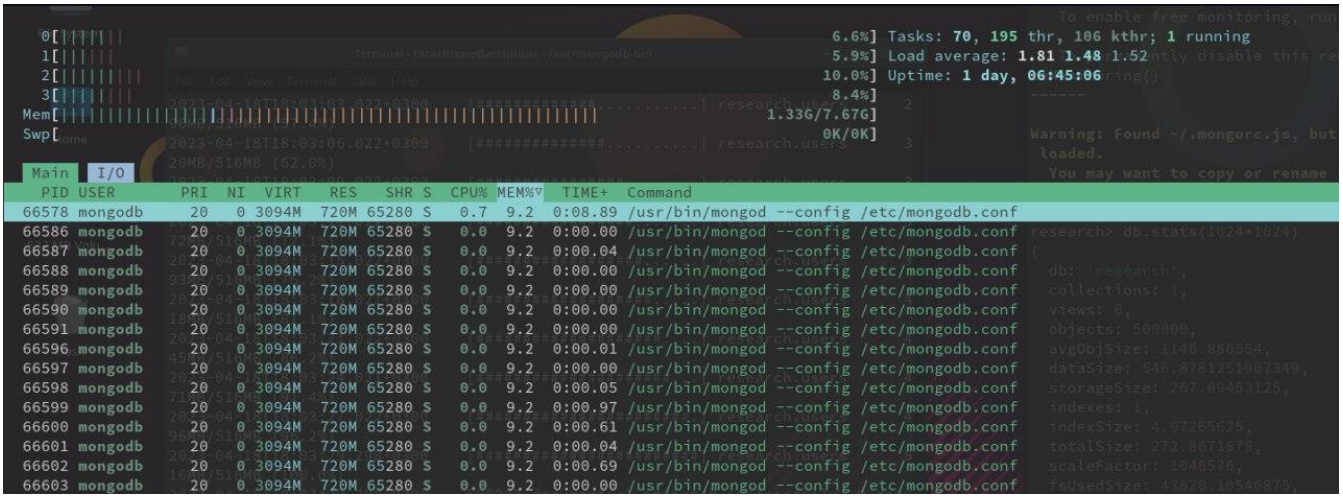

Рисунок 3.4 – Вивід htop під час вставки даних у базу даних P-C

3.2. Порівняння використовуваємих ресурсів при оновлені

Наступним кроком оновимо записи у базі даних. Використаємо такий запит,

при якому буде змінено приблизно половина документів. Оскільки у якості сутностей використовуються користувачі, у якості WHEARE умови візьмемо стать юзеру (за цим поле користувачі розподілені майже порівну (рисунок 3.5)).

```
research> db.users.countDocuments()
500000
research> db.users.countDocuments({gender: 'male'})
250003
research>
```

Рисунок 3.5 – Сумарна користувачів та кількість користувачів-чоловіків

Кінцевий варіант запиту виглядатимете наступним чином: db.users.updateMany({gender: ‘male’}, {$inc: {‘dob.age’: 1}}) (тобто додаємо один рік кожному користувачу-чоловіку). Зазначимо, що тестова колекція немає жодних доданих індексів (окрім згенерованого при вставці індексу _id), отже при оновленні база буде вимушена пройти по кожному рядку.

При операції оновлення даних теж маємо дещо нижчу потребу у ресурсах для С бази (рисунок 3.6) у порівнянні з Р-С базою (рисунок 3.7).

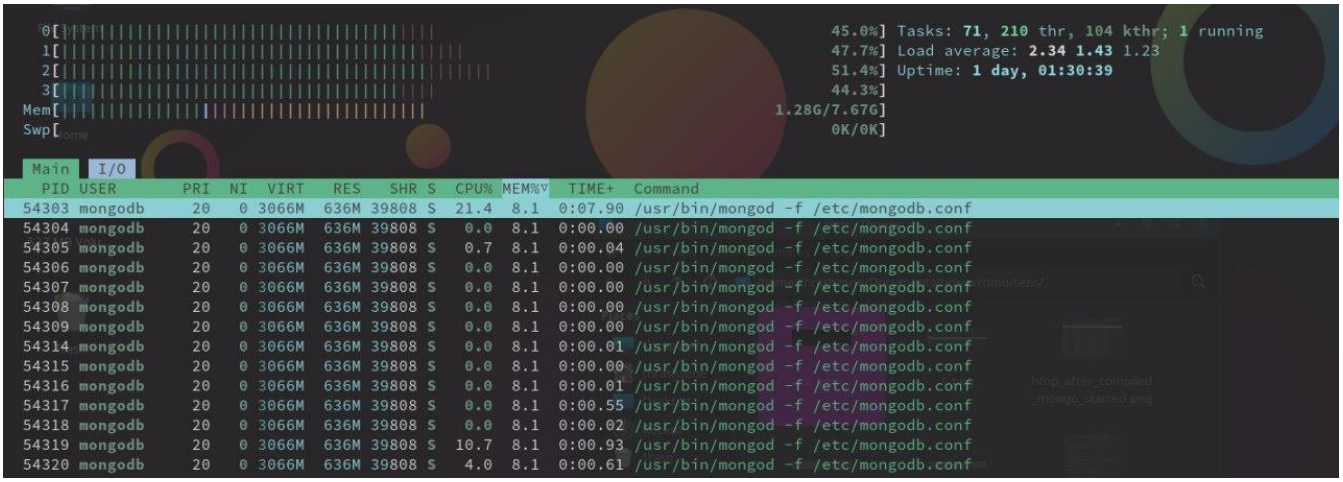

Рисунок 3.6 – Вивів htop під час оновлення документів для С бази даних

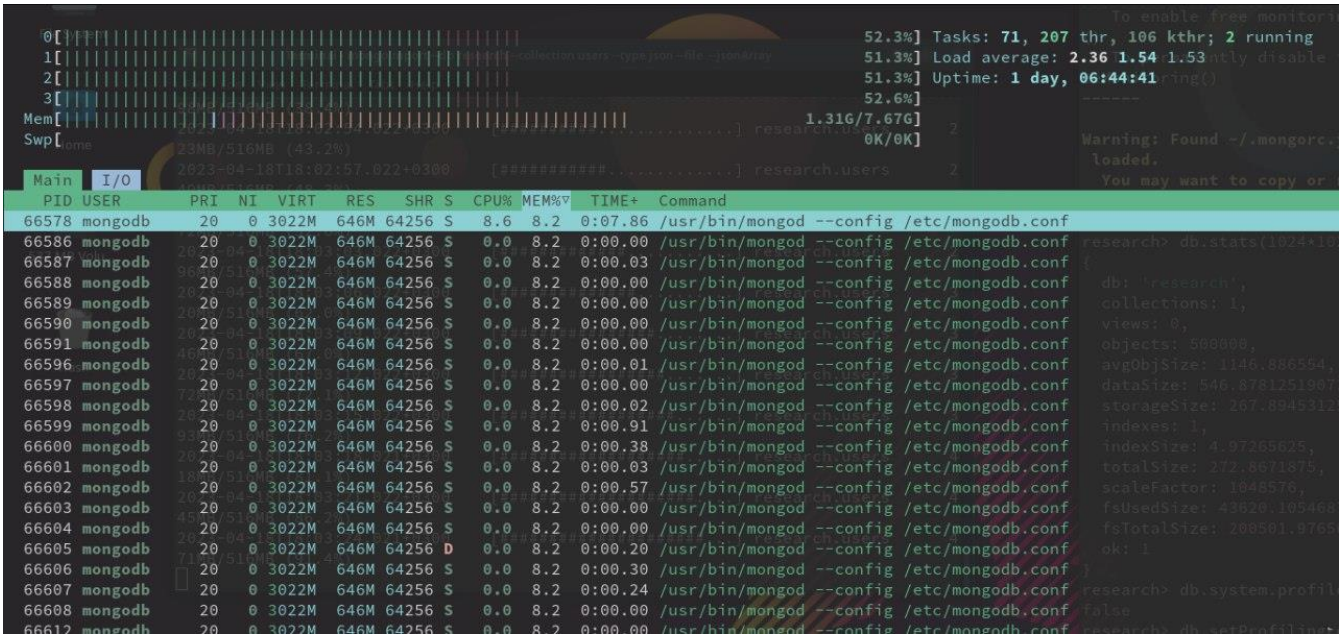

Рисунок 3.7 – Вивід htop під час оновлення документів для Р-С бази даних

Повторим ці операції до десяти разів та у результаті маємо меншу в середньому на 0.8% потребу у ресурсах CPU та на 1.1% менше RAM для С бази даних у порівнянні з Р-С базою даних.

3.3. Порівняння використовуваємих ресурсів при видаленні

Команду видалення побудуємо за наступним принципами:

- Має застосуватися до близько 50% записів;
- Має перевірити усі записи.

Для цього у якості WHEARE умови задаймо gender, який не є проіндексованим (отже, база даних має перевірити кожний наявний рядок). Кінцевий варіант запиту матиме наступний вигляд:

db.users.deleteMany({gender: ‘female’})

У результаті виконання запиту для С бази даних (рисунок 3.8) та Р-С бази

даних (рисунок 3.9) та повторення цих операцій до десяти разім можемо зробити висновок, що в середньому потреба в ресурсах CPU у C бази даних менша на 1.1% порівняно з P-C базою, а потреба в RAM менша на 2.02%.

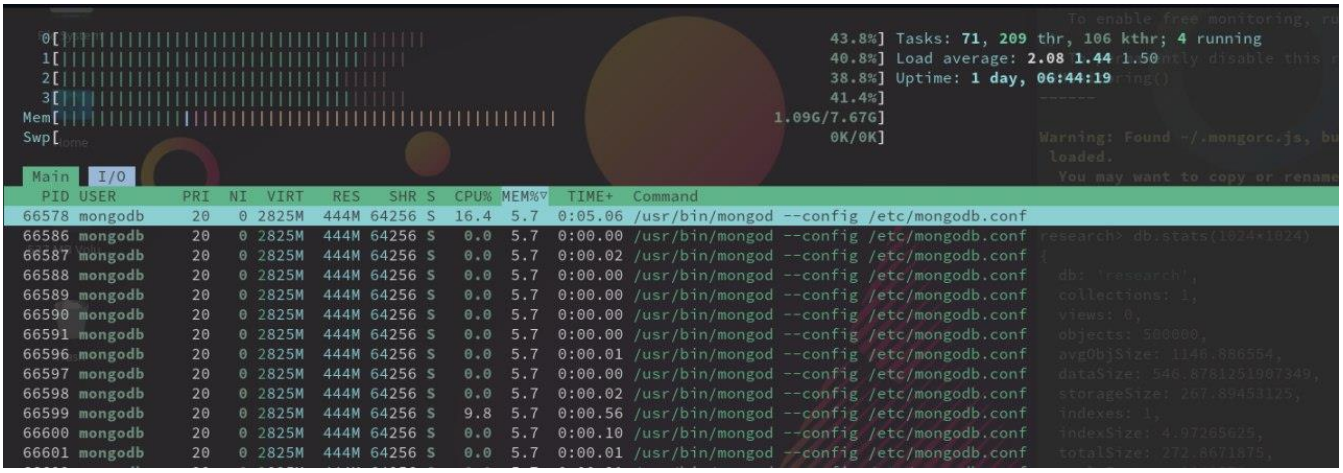

Рисунок 3.8 – Вивід htop під час видалення для C бази даних

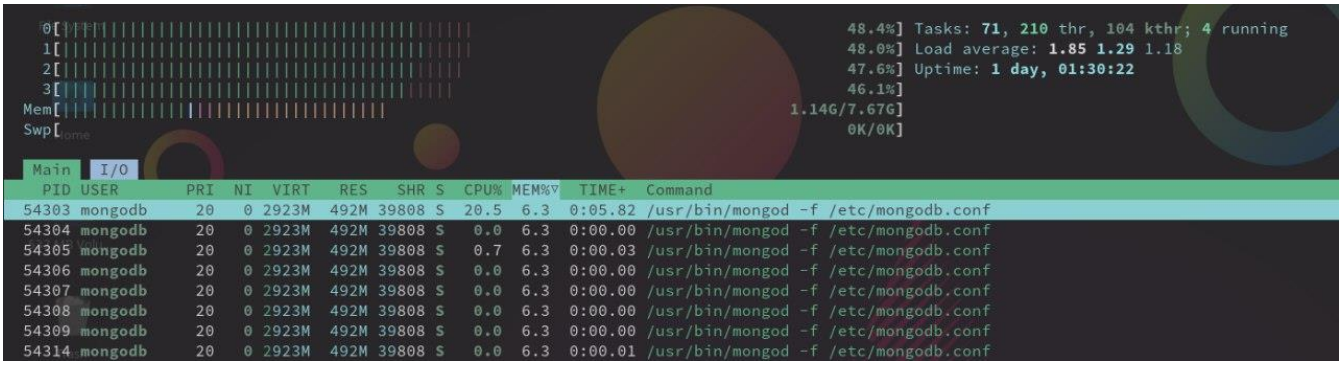

Рисунок 3.9 – Вивід htop під час видалення для P-C бази даних

Залишилось порівняти обсяг пам’яті жорсткого диску, що використовується C та P_C базами даних для зберігання даних. Для цього використаємо команду dbStats [25]. Ця команда поверне статистику сховища для обраної бази даних. У командній оболонці mongosh її аналогом є функція db.stats(), що надає обгортку для dbStats. У якості параметру функція приймає параметр scale. Так, для отримання результатів у кілобайтах, слід встановити параметру scale значення 1024. Оскільки під час дослідження використовується великий об'єм даних, виведемо інформацію в мегабайтах. Для цього встановимо параметру scale значення 1024.

Вивід команди dbStats включає в себе наступні значення:

- db – ім’я бази даних;
- collections – число колекцій у базі даних;
- view – кількість представлень у базі даних (read-only об’єкт, контент якого визначено агрегаційним конвеєром на інших колекціях або представленнях. Представлення є сутністю патерну materialized view, який, в свою чергу є вагомою складовою патерну CQRS (command and query responsibility segregation));
- objects – кількість об’єктів (конкретно документів) у базі даних серед усіх колекцій;
- avgObjSize – середній розмір кожного документу у байтах. Це значення є результатом ділення dataSize на кількість документів;

- dataSize – сумарний розмір нестислих даних у базі;
- storageSize – сумарний розмір дискового простору, що виділено на усі колекції у базі (включаючи вільний простір);
- freeStorageSize – сумарний розмір вільного дискового простору, що виділено на усі колекції у базі;
- indexes – кількість усіх індексів усіх колекцій у базі;
- indexSize – кількість дискового простору, що виділено для зберігання усіх індексів бази даних;
- indexFreeStorageSize – кількість вільного дискового простору, що виділено для зберігання усіх індексів бази даних;
- totalSize – дисковий простір, що виділено для зберігання документів та індексів усіх колекцій бази даних;
- totalFreeStorageSize – вільний дисковий простір, що виділено для зберігання документів та індексів усіх колекцій бази даних;
- scaleFactor – значення параметру scale;
- fsUsedSize – розмір дискового простору у файловій системі, де MongoDB зберігає дані;
- fsTotalSize – сумарне значення розміру дискового простору, де MongoDB зберігає дані.

Слід зазначити, що починаючи з MongoDB 4.4, для запуску dbStat на учаснику набору реплік, учасники повинні бути у стані PRIMARY або SECONDARY [26]. Якщо учасник знаходиться у іншому стані (наприклад STARTUP2), виконання операції закінчиться помилкою.

У попередніх версія MongoDB виконання операції dbStats на учасниках зі статусом STARTUP2 очікувало переходу учасника у статус RECOVERING.

Під час проведення дослідження реплікації бази даних не використовувались (тобто був єдиний екземпляр).

На рисунку 3.10 наведено статистику сховища для C бази даних, на рисунку 3.11 – статистику для сховища P_C бази.

```
research> db.stats(1024*1024)
{
  db: 'research',
  collections: 1,
  views: 0,
  objects: 500000,
  avgObjSize: 1146.886554,
  dataSize: 546.8781251907349,
  storageSize: 267.89453125,
  indexes: 1,
  indexSize: 4.97265625,
  totalSize: 272.8671875,
  scaleFactor: 1048576,
  fsUsedSize: 43620.10546875,
  fsTotalSize: 200501.9765625,
  ok: 1
}
research>
```

Рисунок 3.10 – Статистика сховища C бази даних

```
research> db.stats(1024*1024)
{
  db: 'research',
  collections: 1,
  views: 0,
  objects: 500000,
  avgObjSize: 1146.886554,
  dataSize: 546.8781251907349,
  storageSize: 284.28125,
  indexes: 1,
  indexSize: 4.7421875,
  totalSize: 289.0234375,
  scaleFactor: 1048576,
  fsUsedSize: 43771.05859375,
  fsTotalSize: 200501.9765625,
  ok: 1
}
research>
```

Рисунок 3.11 – Статистика сховища P_C бази даних

Як видно з рисунків 3.10 та 3.11, загальна зарезервований об’єм файлової системи є однаковим для обох баз даних, але обсяг використовуваної пам’яті більше у P_C бази. Потенційно це може призвести до збільшення обсягу зарезервованої пам’яті при збільшенні кількості документів у P_C бази (в свою чергу, пам'ять для C бази збільшена не буде, та ж сама кількість документів займає менше місця).

Підбиваючи проміжні підсумки дослідження, можна стверджувати, що компіляція MongoDB з вихідного коду на комп’ютері кінцевого користувача зменшує її потребу у фізичних ресурсах (RAM в середньому на 1.44% та CPU в середньому на 1%).

4. ПОРІВНЯННЯ ШВИДКОСТІ ВИКОНАННЯ CRUD ОПЕРАЦІЙ

CRUD у якості основних операцій було обрано не випадково. CRUD є скороченням від чотирьох базових операцій, які можна виконати з даними в системі управління базами даних (СУБД). Ці операції визначають основні функції системи обробки даних і використовуються при роботі з багатьма типами баз даних. CRUD включає такі дії:

- Create (Створення): операція, яка дозволяє створити новий запис або об'єкт у базі даних. Це може включати введення нового рядка в таблицю бази даних або створення нового документа в колекції у NoSQL базі даних.
- Read (Читання): Операція, що дозволяє отримати (читати) інформацію з бази даних. Це може включати вибірку даних з таблиці чи колекції, використання запитів для отримання конкретних даних.
- Update (Оновлення): Операція, яка дозволяє змінювати існуючі дані в базі даних. Це може включати оновлення значень у вже існуючому рядку таблиці або оновлення атрибутів документа у NoSQL базі даних.
- Delete (Видалення): Операція, яка дозволяє видаляти дані з бази даних. Це може бути видалення рядка з таблиці або видалення документа з колекції.

Ці чотири операції становлять основні операції для взаємодії з даними в будь-якій базі даних. Коли ви працюєте з додатками або веб-сайтами, ви часто стикаєтеся з цими операціями при роботі зі збереженням, витягуванням, оновленням та видаленням даних. Це є основною моделлю для обробки даних у системах управління базами даних.

Для вимірювання часу виконання операцій скористаємось механізмом профілювання у MongoDB. В MongoDB профіль відноситься до функціональності, яка дозволяє включити детальне ведення журналу для аналізу та оптимізації діяльності бази даних. Зазвичай профілювання використовується для відстеження

того, які запити до бази даних виконуються і як довго часу це займає. Основні елементи профілювання MongoDB:

1. Види профілювання:
   - Off (вимкнено): профілювання вимкнено (опція за замовчуванням);
   - Slow Op (повільні операції): записуються лише операції, які тривають довше певного порогового значення (зазвичай 100 мілісекунд);
   - All Op (всі операції): записуються всі операції, незалежно від їхньої тривалості.
2. Де зберігаються дані профілювання: профільні дані зберігаються у спеціальній колекції в базі даних. Зазвичай це колекція з назвою “system.profile” [27].
3. Як включити профілювання: для цього можна скористатися командою db.setProfilingLevel(1) (1 – профілювання всіх операцій). Зазначимо, що рівень профілювання не зберігається, тобто після перезапуску бази даних він знов буде 0.
4. Аналіз даних профілювання: після включення профілювання, можна аналізувати дані з колекції “system.profile”, щоб переглядати, які операції виконуються, і визначити, які можливі покращення можна внести до коду чи структури бази даних.

Слід зауважити, що використання профілювання може призвести до збільшення розміру бази даних через зберігання профільованих даних.

4.1. Швидкість створення

Проведемо операцію імпорту аналогічну той, що проводилась у розділі 3.1, але вже з ввімкненим профілюванням. Після імпорту даних маємо відповідні записи у колекції system.profile. Зробимо запит на максимальний час виконання запиту для C бази даних (рисунок 4.1) та для P_C бази даних (рисунок 4.2). Для виконання запиту скористаємось функцією агрегації для обробки багатьох документів та

повернення обчислювального результату [28]. Результат обчислюватиметься функцією $sum [29].

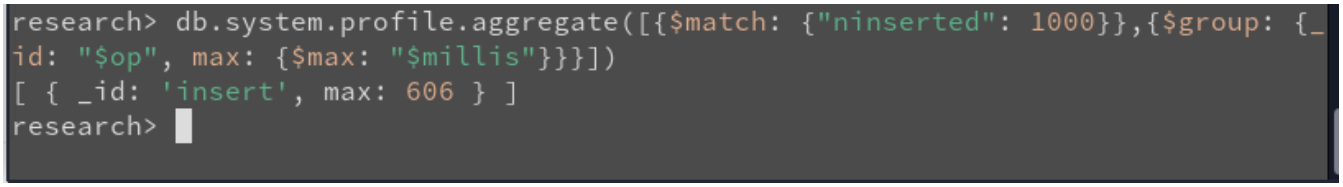

Рисунок 4.1 – Максимальний час операції запису для C бази даних

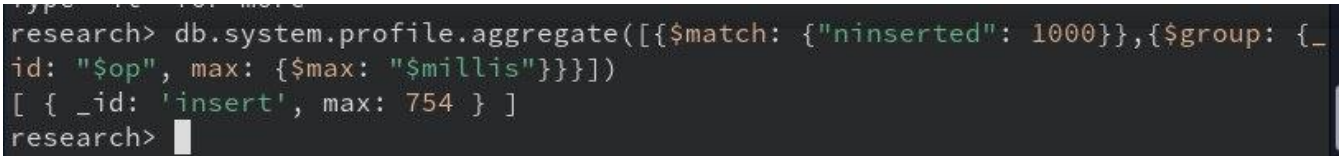

Рисунок 4.2 – Максимальний час операції запису

Як видно з рисунків 4.1 та 4.2, обидва запити за часом значно перевищують допустимий ліміт у 100 мілісекунд (запити, що займають більше часу, вважаються повільними [30]).

Тепер перевіримо середній час запису (маємо 5000 документів на 1 запис та 1000 операцій запису). Для цього скористаємось функцією aggregate [31]. Середній час запису для C бази даних наведено на рисунку 4.3, середній час запису для P_C бази даних наведено на рисунку 4.4.

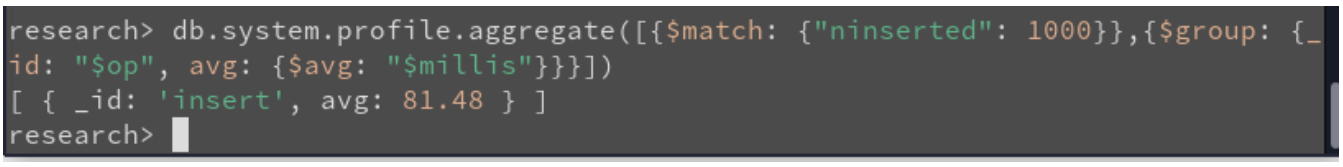

Рисунок 4.3 – Середній час запису для C бази даних

```
research> db.system.profile.aggregate([{$match: {"ninserted": 1000}},{$group: {_
id: "$op", avg: {$avg: "$millis"}}}])
[ { _id: 'insert', avg: 65.312 } ]
research>
```

Рисунок 4.4 – Середній час запису для P_C бази даних

Як видно з рисунків 4.3 та 4.4, середній час для обох баз є прийнятним (менше 100 мілісекунд), але для бази P_C він є швидшим на 24%.

4.2. Швидкість оновлення

Проведемо операцію оновлення аналогічну той, що проводилась у розділі 3.2, але вже з ввімкненим профілюванням. Для C бази даних операція оновлення тривала в середньому 8400 мілісекунд (рисунок 4.5).

```
research> db.system.profile.find()
[
  {
    op: 'update',
    ns: 'research.users',
    command: {
      q: {},
      u: { '$inc': { 'dob.age': 1 } },
      multi: true,
      upsert: false
    },
    keysExamined: 0,
    docsExamined: 500000,
    nMatched: 500000,
    nModified: 500000,
    nUpserted: 0,
    numYield: 551,
    queryHash: '17830885',
    locks: {
      ParallelBatchWriterMode: { acquireCount: { r: Long("552") } },
      FeatureCompatibilityVersion: { acquireCount: { r: Long("1"), w: Long("552
") } },
      ReplicationStateTransition: { acquireCount: { w: Long("553") } },
      Global: { acquireCount: { r: Long("1"), w: Long("552") } },
      Database: { acquireCount: { w: Long("552") } },
      Collection: { acquireCount: { w: Long("552") } },
      Mutex: { acquireCount: { r: Long("1") } }
    },
    flowControl: { acquireCount: Long("552"), timeAcquiringMicros: Long("865")
,
    storage: {
      data: { bytesRead: Long("46901718"), timeReadingMicros: Long("8833") }
    },
    millis: 8402,
    planSummary: 'COLLSCAN',
    execStats: {
      stage: 'UPDATE',
      nReturned: 0,
```

Рисунок 4.5 – Операція оновлення у C бази даних

З іншого боку, операції оновлення для P_C бази даних були в середньому швидші на 14 відсотків. Приклад статистичних викладок по такому запиту наведено на рисунку 4.6.

Рисунок 4.6 – Операція оновлення у P_C бази даних

4.3. Швидкість видалення

Проведемо операцію видалення аналогічну той, що проводилась у розділі 3.3, але вже з ввімкненим профілюванням. Для C бази даних операція оновлення тривала в середньому 4501 мілісекунд (рисунок 4.7).

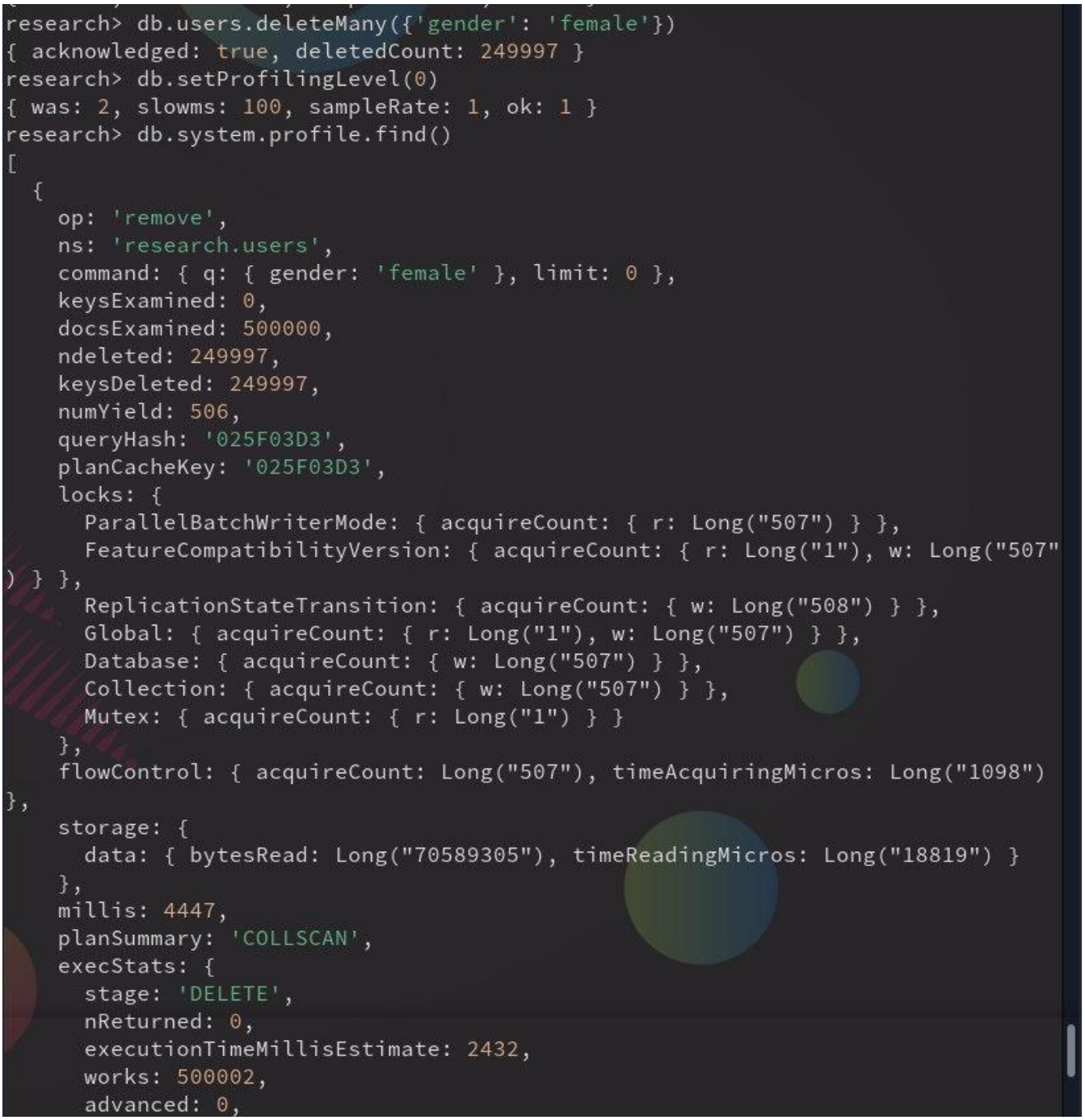

Рисунок 4.7 – Статистика операції видалення для C бази даних

В свою чергу операції видалення для P_C бази даних тривали в середньому 3461 мілісекунду. Приклад статистики по однієї за таких операцій наведено на рисунку 4.8.

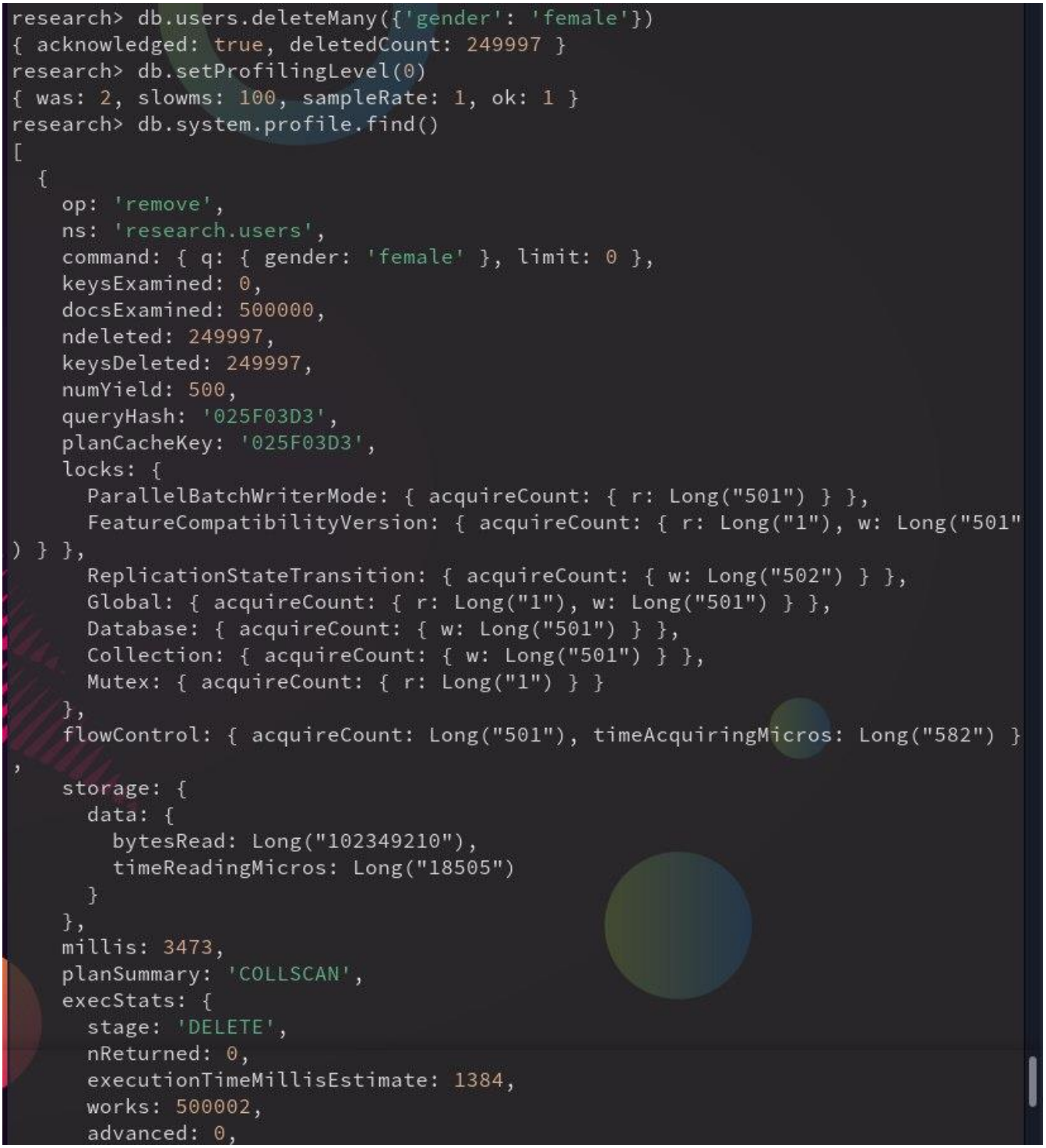

Рисунок 4.8 – Статистика операції видалення для P_C бази даних

Можна стверджувати, що операції видалення у P_C базі даних в середньому на 30% швидші, ніж аналогічні операції у C базі даних.

Підсумовуючи підсумки щодо швидкості виконання CRUD операцій у compile базі даних порівняно з pre-compiled, можна стверджувати, що вони в середньому на 22.67% швидші у pre-compiled версії.

ВИСНОВКИ

Було проведено дослідження метрик ефективності використання MongoDB, зібраної з вихідного коду, у порівнянні зі стандартною pre-compiled версією, завантаженої з офіційного сайту. Під час підготовки до дослідження було проаналізовано перспективність збірки програм з вихідного коду з різними параметрами компілятору, вплив на продуктивність таких програм. Було створено створено програму-генератор для створення великого обсягу JSON-даних, якими і маніпулювали під час дослідження. Було застосовано емпіричні методи наукового дослідження: вимірювання та порівняння.

Дослідження включало в себе наявність MongoDB, скомпільованої з вихідного коду, та MongoDB, завантаженої з офіційного сайту (pre-compiled binary). Для обох баз проводились однакові CRUD операції та робилися заміри щодо часу їх виконання та потреби баз у ресурсах комп'ютеру (RAM та CPU). У результаті дослідження було виявлено, що при використанні MongoDB скомпільованої на комп'ютері кінцевого користувача потреба у RAM в середньому на 1.44% нижча порівняно з pre-compiled MongoDB та навантаження на CPU в середньому на 1%. Але разом з цим швидкість виконання CRUD операцій у такій базі на 22.67% менша. Отже можна рекомендувати збірку MongoDB з вихідного коду тим користувачам, у яких «залізо» є слабким та «кожен мегабайт оперативної пам'яті на рахунку» для зменшення навантаження на RAM та CPU. Використання MongoDB з вихідного коду зменшить навантаження на фізичні ресурси комп'ютеру та використовувати ресурси більш ефективно для виконання інших задач. Але разом з цим слід враховувати, що операції запису, оновлення та видалення даних будуть виконуватися повільніше. Слід зауважити, що для кінцевого користувача «втрата у швидкості» буде помітна лише на великому об'ємі даних, оскільки в середньому CRUD операції виконувались за менше, чим 100 мілісекунд, а ця швидкість є прийнятною.

Дослідження має наступні перспективи:

- проведення на комп'ютерах з різним «залізом»;

- проведення на комп’ютерах під керівництвом ОС Windows та MacOS;
- вивчення можливостей використання різних параметрів компілятору для оптимізації процесу збірки та вихідних виконуваних файлів під конкретну платформу та архітектуру процесора.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ

1. Computer science: Вікіпедія [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Computer_science (дата звернення: 16.12.2023).
2. How Random Access Memory (RAM) affets performance: Dell Technologies [Електронний ресурс]. URL: https://www.dell.com/support/kbdoc/ru-ua/000129805/how-random-access-memory-ram-affects-performance (дата звернення: 16.12.2023).
3. CRUD: Wikipedia [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete (дата звернення).
4. DB-Engines Ranking: DB-Engines. [Електронний ресурс]. URL: https://db-engines.com/en/ranking (дата звернення 16.12.2023).
5. MongoDB source code: GitHub. [Електронний ресурс]. URL: https://github.com/mongodb/mongo (дата звернення 16.12.2023).
6. Arch User Repository: Archlinux [Електронний ресурс]. URL: https://wiki.archlinux.org/title/Arch_User_Repository (дата звернення 18.12.2023).
7. Mongodb and SSPL: lists.archlinux.org [Електронний ресурс]. URL: https://lists.archlinux.org/archives/list/arch-dev-public@lists.archlinux.org/thread/OY2DLNWTZOBPAHFE5FSV4Q6AIWGZO6KV/ (дата звернення 18.12.2023).
8. GCC optimization: Gentoo Linux [Електронний ресурс]. URL: https://wiki.gentoo.org/wiki/GCC_optimization (дата звернення: 18.12.2023).
9. Handbook:Parts/Working/USE: Gentoo Linux [Електронний ресурс]. URL: https://wiki.gentoo.org/wiki/Handbook:Parts/Working/USE (дата звернення: 19.12.2023).
10. Compiling Code: NC State University [Електронний ресурс]. URL:

https://hpc.ncsu.edu/Documents/Compilers.php (дата звернення: 19.12.2023).

11. 4 ways to tell if your data is big data: Domo. [Електронний ресурс]. URL: https://www.domo.com/learn/article/4-ways-to-tell-if-your-data-is-big-data (дата звернення 16.12.2023).
12. What is Big Data: ITChronicles. [Електронний ресурс]. URL: https://itchronicles.com/what-is-big-data/ (дата звернення 16.12.2023).
13. Representational State Transfer (REST): Architectural Styles and the Design of Network-base Software Architectures. [Електронний ресурс]. URL: https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення 19.12.2023).
14. What is OSI Model: AWS. [Електронний ресурс]. URL: https://aws.amazon.com/what-is/osi-model/#:~:text=The%20Open%20Systems%20Interconnection%20(OSI,across%20geographical%20and%20political%20boundaries. (дата звернення 19.12.2023).
15. Promise based HTTP client for the browser and node.js: Node Package Manager. [Електронний ресурс]. URL: https://www.npmjs.com/package/axios (дата звернення 16.12.2023).
16. Promise: MDN Web Docs. [Електронний ресурс] URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise (дата звернення 16.12.2023).
17. node-sleep: GitHub. [Електронний ресурс]. URL: https://github.com/erikdubbelboer/node-sleep (дата звернення 16.12.2023).
18. The event loop: mdn web docs. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Event_loop (дата звернення 18.12.2023)
19. Node.js v21.4.0 documentation: NodeJS. [Електронний ресурс]. URL: https://nodejs.org/api/stream.html (дата звернення 16.12.2023).

20. JSON to MongoDB: MongoDB. [Електронний ресурс]. URL: https://www.mongodb.com/compatibility/json-to-mongodb (дата звернення 16.12.2023).
21. JSON Defined: Oracle. [Електронний ресурс]. URL: https://www.oracle.com/database/what-is-json (дата звернення 18.12.23).
22. mongoimport: MongoDB. [Електронний ресурс]. URL: https://www.mongodb.com/docs/database-tools/mongoimport/ (дата звернення 18.12.2023).
23. What is Apache MapReduce: IBM [Електронний ресурс]. URL: https://www.ibm.com/topics/mapreduce (дата звернення 19.12.2023)
24. htop explained: Pēteris Ņikiforovs. [Електронний ресурс]. URL: https://peteris.rocks/blog/htop/ (дата звернення: 16.12.2023).
25. db.stats(): MongoDB [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/reference/method/db.stats (дата звернення 19.12.2023).
26. Replication: MongoDB. [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/replication/ (дата звернення 19.12.2023).
27. System Collections: MongoDB. [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/reference/system-collections/ (дата звернення 18.12.2023).
28. Aggregation Operations: MongoDB. [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/aggregation/ (дата звернення 17.12.2023).
29. $sum: MongoDB. [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/reference/operator/aggregation/sum/ (дата звернення 17.12.2023).
30. db.setProfilingLevel(): MongoDB. [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/reference/method/db.setProfilingLevel/

(дата звернення 17.12.2023).

31. db.collection.aggregate(): MongoDB [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/reference/method/db.collection.aggregate/ (дата звернення 19.12.23).

ДОДАТОК А
Відомість матеріалів кваліфікаційної роботи

		Позначення	Найменування	Кільк. аркушів	Примітка
1					
2			Документація		
3					
4		**ІСТ.КР 23.04.ДА.ПЗ**	Пояснювальна записка	71	
5					
6			Презентація	12	
7					
8			Диск CD з презентацією	1	

					ІСТ.КР 23.04.ДА.ПЗ
Зм.	Ар-куш	№ докум	Підпис	Дата	

Розроб.		Ініц., прізв.			**Матеріали кваліфікаційної роботи**	Літ.	Аркуш	Арку-шів
Керівник		Сергєєва К.Л.				Н	1	1
Н. контр.						НТУ «ДП», 12; 126-22м-1		
Зав. каф.		Гнатушенко В.В						

ДОДАТОК Б. Вихідний код генератору

Файл package.json:

```
{
  "name": "generator",
  "version": "1.0.0",
  "description": "Script for generation random user data",
  "main": "generator.js",
  "scripts": {
    "start": "node generator"
  },
  "author": "Oleksandr Maksymenko",
  "license": "ISC",
  "dependencies": {
    "axios": "^1.6.2",
    "sleep": "^6.3.0"
  }
}
```

Файл generator.js:

```
let everything = [];

const start = async () => {
  const fs = require("fs");
  const axios = require('axios');
  const sleep = require('sleep');

  const writeStream = fs.createWriteStream("data.json");

  writeStream.write("[");
  const numOfUsers = 5000;
  const max = 200;

  for(let i = 0; i < max; i++) {
```

```
    console.log(iteration ${i});
    try {
      const { data } = await axios.get(https://randomuser.me/api/?results=${numOfUsers});

      sleep.sleep(60);
      const dataStr = JSON.stringify(data.results);
      writeStream.write(dataStr.substring(1, dataStr.length - 1));
      if(i + 1 < max) writeStream.write(",");
    } catch (e) {
      console.error("api error, will wait more");
      sleep.sleep(180);
      i--;
    }
  }
  writeStream.write("]");
  writeStream.end();
};

start();
```

Файл sleep.js

```
var sleep = require('./build/Release/node_sleep.node');

sleep.sleep = function(seconds) {
  if (seconds < 0 || seconds % 1 != 0) {
    throw new Error('Expected number of seconds');
  }
  sleep.usleep(seconds * 1000000);
}


sleep.msleep = function(miliseconds) {
  if (miliseconds < 0 || miliseconds % 1 != 0) {
    throw new Error('Expected number of miliseconds');
  }
  sleep.usleep(miliseconds * 1000);
}

module.exports = sleep;
```

Файл sleep.test.js

```
/* globals describe, it */
var sleep = require('./');
var assert = require('assert');
var child_process = require('child_process');
```

```
function assertApproxEqual(val1, val2) {
  // We require accuracy to the nearest N millisecond.
  // Windows Sleep is not super accurate (depends on scheduling policy) so use a high value.
  var epsilon = 100;
  var diff = val1 - val2;
  if (diff > epsilon) {
    assert.fail('wait was too long: ' + diff + ' > ' + epsilon);
  } else if (diff < -epsilon) {
    assert.fail('wait was too long: ' + diff + ' < ' + -epsilon);
  }
}

describe('sleep', function () {
  it('works for normal input', function () {
    var sleepTime = 1;
    var start = new Date();
    sleep.sleep(sleepTime);
    var end = new Date();
    assertApproxEqual(end - start, sleepTime * 1000);
  });

  it('works for zero', function () {
    var sleepTime = 0;
    var start = new Date();
    sleep.sleep(sleepTime);
    var end = new Date();
    assertApproxEqual(end - start, sleepTime * 1000);
  });

  it('does not allow negative numbers', function () {
    assert.throws(function () {
      sleep.sleep(-1);
    });
  });

  it('works with child_process', function () {
    var sleepTime = 1;
    child_process.exec('echo', function (err, stdout, stderr) { });
    var start = new Date();
    sleep.sleep(sleepTime);
    var end = new Date();
    assertApproxEqual(end - start, sleepTime * 1000);
  });
});

describe('usleep', function () {
  it('works for values smaller than a second', function () {
    var sleepTime = 250;
    var start = new Date();
    sleep.usleep(sleepTime);
    var end = new Date();
    assertApproxEqual(end - start, sleepTime / 1000);
  });

  it('works for zero', function () {
    var sleepTime = 0;
    var start = new Date();
    sleep.usleep(sleepTime);
    var end = new Date();
    assertApproxEqual(end - start, sleepTime / 1000);
  });

  it('works for values larger than a second', function () {
    this.timeout(4000); // necessary for mocha to not complain
    var sleepTime = 3000000;
    var start = new Date();
    sleep.usleep(sleepTime);
    var end = new Date();
```

```
    assertApproxEqual(end - start, sleepTime / 1000);
  });

  it('does not allow negative numbers', function () {
    assert.throws(function () {
      sleep.usleep(-100);
    });
  });
});

describe('msleep', function () {
  it('works for normal input', function() {
    var sleepTime = 1;
    var start = new Date();
    sleep.msleep(sleepTime);
    var end = new Date();
    assertApproxEqual(end - start, sleepTime);
  });

  it('works for zero', function () {
    var sleepTime = 0;
    var start = new Date();
    sleep.msleep(sleepTime);
    var end = new Date();
    assertApproxEqual(end - start, sleepTime);
  });

  it('does not allow negative numbers', function () {
    assert.throws(function () {
      sleep.msleep(-100);
    });
  });

  it('does not allow decimal numbers', function () {
    assert.throws(function () {
      sleep.msleep(1.5);
    });
  });
});

describe('error', function () {
  it('should throw a valid error', function () {
    try {
      sleep.msleep(Infinity);
    }
    catch (e) {
      assert.equal(e.message, 'Expected number of miliseconds');
    }
  });
});
```

Файл cancellableFunction.js

```
/**
 * @param {Generator} generator
 * @return {[Function, Promise]}
 */
var cancellable = function (generator) {
    let rejectPromise;
    const promise = new Promise(async (resolve, reject) => {
        rejectPromise = () => {
            try {
                const result = generator.throw('Cancelled').value;
                resolve(result);
            } catch(e) {
```

```
            reject('Cancelled');
        }
    };

    const process = async (val, isErr) => {
        try {
            let next;
            if (isErr) {
                try {
                    next = generator.throw(val);
                    isErr = false;
                } catch (e) {
                    return {
                        done: true,
                        result: val,
                        isError: true
                    }
                }
            } else {
                next = generator.next(val);
            }
            if (next.done) {
                return {
                    done: true,
                    result: next.value === undefined ? val : next.value,
                    isError: isErr
                }
            } else {
                const res = await next.value;
                return {
                    done: false,
                    result: res,
                    isError: false
                }
            }
        } catch (e) {
            return {
                done: false,
                result: e,
                isError: true
            }
        }
    }

    let { done, result, isError } = await process();
    while (!done) {
        ({ done, result, isError } = await process(result, isError));
    }

    isError ? reject(result) : resolve(result);
});

return [
    () => {
        rejectPromise();
    },
    promise
];
};


// const generator = function* () { try { yield new Promise((resolve, reject) => reject("Promise Rejected")); } catch (e) { let a = yield new Promise(resolve => resolve(2)); let b = yield new Promise(resolve => resolve(2)); return a + b; }; }

// const generator = function*() { return 42; }

// const generator = function*() { const msg = yield new Promise(res => res("Hello")); throw `Error: ${msg}`; }
```

```
// const generator = function*() { let result = 0; try { yield new Promise(res => setTimeout(res, 10)); result += yield new Promise(res => res(1)); yield new Promise(res => setTimeout(res, 10)); result += yield new Promise(res => res(1)); } catch(e) { return result; } return result; }

const generator = function*() { try { yield new Promise(res => setTimeout(res, 200)); yield new Promise((resolve, reject) => reject("Promise Rejected")); } catch(e) { return e; }; return "Hi"; };

const [cancel, promise] = cancellable(generator())
promise.then((v) => console.log('resolved', v), (e) => console.error('rejected', e));
//setTimeout(cancel, 15);
/**
 * function* tasks() {
 *   const val = yield new Promise(resolve => resolve(2 + 2));
 *   yield new Promise(resolve => setTimeout(resolve, 100));
 *   return val + 1;
 * }
 * const [cancel, promise] = cancellable(tasks());
 * setTimeout(cancel, 50);
 * promise.catch(console.log); // logs "Cancelled" at t=50ms
 */
```

Файл dataEventEmmiter.js

```
class EventEmitter {
    constructor() {
        this.map = new Map();
    }

    subscribe(event, cb) {
        if (this.map.has(event)) his.map.get(event).push(cb);
        else  this.map.set(event, [cb]);

        return {
            unsubscribe: () => {
                const cbs = this.map.get(event);
                if (cbs.length === 1) {
                    this.map.delete(event);
                } else {
                    cbs.splice(cbs.indexOf(cb), 1);
                }
            }
        };
    }

    emit(event, args = []) {
        const cbs = this.map.get(event);
        if (cbs === undefined) return [];
        else return cbs.map(cb => cb(...args));
    }
}

const emitter = new EventEmitter();

// Subscribe to the onClick event with onClickCallback
function onClickCallback() { return 99 }
const sub = emitter.subscribe('onClick', onClickCallback);

console.log(emitter.emit('onClick')); // [99]
sub.unsubscribe(); // undefined
console.log(emitter.emit('onClick')); // []
```

Файл memorize.js

```
/**
 * @param {Function} fn
 */
function memoize(fn) {
    const argsNResult = new Map();

    return function () {
        if(arguments.length === 0) {
            if(argsNResult.has('empty')) return argsNResult.get('empty');
            const result = fn();
            argsNResult.set('empty', result);
            return result;
        }

        const key = [ ...arguments].map(arg => {
            if(arg === undefined) return 'undefined';
            return JSON.stringify(arg);
        }).join(':');

        if(argsNResult.has(key)) {
            const value = argsNResult.get(key);
            let flag = true;
            for(let i = 0; i < value.length; i++) {
                let entry = value[i];
                for(let j = 0; j < entry.args.length; j++) {
                    if(arguments[j] !== entry.args[j]) {
                        flag = false;
                        break;
                    } else {
                        flag = true;
                    }
                }
                if(flag) {
                    break;
                }
            }

            if(!flag) {
                    const result = fn(...arguments);
                    value.push({args: [...arguments], result: result});
                    return result;
            }

            return value[0].result;
        }

        const result = fn(...arguments);
        argsNResult.set(key, [{args: [...arguments], result: result}]);

        return result;
    }
}

const o = {};
let callCount = 0;
const memoizedFn = memoize(function (a, b) {
    callCount += 1;
    return ({...a, ...b});
});
```

Файл groupBy.js

```
/**
 * @param {Function} fn
 * @return {Array}
 */
Array.prototype.groupBy = function (fn) {
    result = {};
    for (let i = 0; i < this.length; i++) {
        const key = fn(this[i]);
        if (result.hasOwnProperty(key)) {
            result[key].push(this[i]);
        } else {
            result[key] = [this[i]];
        }
    }
    return result;
};


console.log([1, 2, 3].groupBy(String)); // {"1":[1],"2":[2],"3":[3]}
```

Файл reduceNElements.js

```
function ListNode(val, next) {
    this.val = (val === undefined ? 0 : val)
    this.next = (next === undefined ? null : next)
}

/**
 * @param {ListNode} l1
 * @param {ListNode} l2
 * @return {ListNode}
 */
var addTwoNumbers = function (l1, l2) {
    const num1 = getNum(l1);
    const num2 = getNum(l2);

    const sum = (num1 + num2).toString().split("").reverse().join("");

    return convertToListNode(sum, 0, new ListNode());
};

const convertToListNode = (str, index, listNode) => {
    if (index === str.length) return null;
    listNode.val = parseInt(str.charAt(index));
    listNode.next = convertToListNode(str, index + 1, new ListNode());
    return listNode;
};

const getNum = (listNode) => {
    let numReverse = "";
    while (listNode.next !== null) {
        numReverse = numReverse.concat(listNode.val);
        listNode = listNode.next;
    }
    numReverse = numReverse.concat(listNode.val);
    return parseInt(numReverse.split("").reverse().join(""));
}

const head = new ListNode();
//[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1].forEach(v => )

const l1 = new ListNode(1, new ListNode(0, new ListNode(0, new ListNode(0, new ListNode(0, new ListNode(0, new
```

```
ListNode(0,
            new ListNode(0, new ListNode(0)))))))));
        const l2 = new ListNode(5, new ListNode(6, new ListNode(4)));
        const sum = addTwoNumbers(l1, l2);

        console.log('end');
```

Файл package-lock.json

```
{
  "name": "tezis",
  "version": "1.0.0",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
   "": {
    "name": "tezis",
    "version": "1.0.0",
    "license": "ISC",
    "dependencies": {
     "axios": "^1.6.2",
     "sleep": "^6.3.0"
    }
   },
   "node_modules/asynckit": {
    "version": "0.4.0",
    "resolved": "https://registry.npmjs.org/asynckit/-/asynckit-0.4.0.tgz",
    "integrity": "sha512-
Oei9OH4tRh0YqU3GxhX79dM/mwVgvbZJaSNaRk+bshkj0S5cfHcgYakreBjrHwatXKbz+IoIdYLxrKim2MjW0Q=="
   },
   "node_modules/axios": {
    "version": "1.6.2",
    "resolved": "https://registry.npmjs.org/axios/-/axios-1.6.2.tgz",
    "integrity": "sha512-
7i24Ri4pmDRfJTR7LDBhsOTtcm+9kjX5WiY1X3wIisx6G9So3pfMkEiU7emUBe46oceVImccTEM3k6C5dbVW8A==",
    "dependencies": {
     "follow-redirects": "^1.15.0",
     "form-data": "^4.0.0",
     "proxy-from-env": "^1.1.0"
    }
   },
   "node_modules/combined-stream": {
    "version": "1.0.8",
    "resolved": "https://registry.npmjs.org/combined-stream/-/combined-stream-1.0.8.tgz",
    "integrity": "sha512-
FQN4MRfuJeHf7cBbBMJFXhKSDq+2kAArBlmRBvcvFE5BB1HZKXtSFASDhdlz9zOYwxh8lDdnvmMOe/+5cdoEdg==",
    "dependencies": {
     "delayed-stream": "~1.0.0"
    },
    "engines": {
     "node": ">= 0.8"
    }
   },
   "node_modules/delayed-stream": {
    "version": "1.0.0",
    "resolved": "https://registry.npmjs.org/delayed-stream/-/delayed-stream-1.0.0.tgz",
    "integrity": "sha512-
ZySD7Nf91aLB0RxL4KGrKHBXl7Eds1DAmEdcoVawXnLD7SDhpNgtuII2aAkg7a7QS41jxPSZ17p4VdGnMHk3MQ==",
    "engines": {
     "node": ">=0.4.0"
    }
   },
   "node_modules/follow-redirects": {
    "version": "1.15.3",
    "resolved": "https://registry.npmjs.org/follow-redirects/-/follow-redirects-1.15.3.tgz",
```

```
      "integrity": "sha512-
1VzOtuEM8pC9SFU1E+8KfTjZyMztRsgEfwQl44z8A25uy13jSzTj6dyK2Df52iV0vgHCfBwLhDWevLn95w5v6Q==",
      "funding": [
       {
        "type": "individual",
        "url": "https://github.com/sponsors/RubenVerborgh"
       }
      ],
      "engines": {
       "node": ">=4.0"
      },
      "peerDependenciesMeta": {
       "debug": {
        "optional": true
       }
      }
     },
     "node_modules/form-data": {
      "version": "4.0.0",
      "resolved": "https://registry.npmjs.org/form-data/-/form-data-4.0.0.tgz",
      "integrity": "sha512-
ETEklSGi5t0QMZuiXoA/Q6vcnxcLQP5vdugSpuAyi6SVGi2clPPp+xgEhuMaHC+zGgn31Kd235W35f7Hykkaww==",
      "dependencies": {
       "asynckit": "^0.4.0",
       "combined-stream": "^1.0.8",
       "mime-types": "^2.1.12"
      },
      "engines": {
       "node": ">= 6"
      }
     },
     "node_modules/mime-db": {
      "version": "1.52.0",
      "resolved": "https://registry.npmjs.org/mime-db/-/mime-db-1.52.0.tgz",
      "integrity": "sha512-
sPU4uV7dYlvtWJxwwxHD0PuihVNiE7TyAbQ5SWxDCB9mUYvOgroQOwYQQOKPJ8CIbE+1ETVlOoK1UC2nU3gYvg==",
      "engines": {
       "node": ">= 0.6"
      }
     },
     "node_modules/mime-types": {
      "version": "2.1.35",
      "resolved": "https://registry.npmjs.org/mime-types/-/mime-types-2.1.35.tgz",
      "integrity": "sha512-
ZDY+bPm5zTTF+YpCrAU9nK0UgICYPT0QtT1NZWFv4s++TNkcgVaT0g6+4R2uI4MjQjzysHB1zxuWL50hzaeXiw==",
      "dependencies": {
       "mime-db": "1.52.0"
      },
      "engines": {
       "node": ">= 0.6"
      }
     },
     "node_modules/nan": {
      "version": "2.18.0",
      "resolved": "https://registry.npmjs.org/nan/-/nan-2.18.0.tgz",
      "integrity": "sha512-
W7tfG7vMOGtD30sHoZSSc/JVYiyDPEyQVso/Zz+/uQd0B0L46gtC+pHha5FFMRpil6fm/AoEcRWyOVi4+E/f8w=="
     },
     "node_modules/proxy-from-env": {
      "version": "1.1.0",
      "resolved": "https://registry.npmjs.org/proxy-from-env/-/proxy-from-env-1.1.0.tgz",
      "integrity": "sha512-
D+zkORCbA9f1tdWRK0RaCR3GPv50cMxcrz4X8k5LTSUD1Dkw47mKJEZQNunItRTkWwgtaUSo1RVFRIG9ZXiFYg=="
     },
     "node_modules/sleep": {
      "version": "6.3.0",
      "resolved": "https://registry.npmjs.org/sleep/-/sleep-6.3.0.tgz",
      "integrity": "sha512-
```

+WgYl951qdUlb1iS97UvQ01pkauoBK9ML9I/CMPg41v0Ze4EyMlTgFTDDo32iYj98IYqxIjDMRd+L71lawFfpQ==",
"hasInstallScript": true,
"dependencies": {
"nan": "^2.14.1"
},
"engines": {
"node": ">=0.8.0"
}
}
}
}
}