Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерна інженерія та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Моделі паралельного та розподіленого моделювання в хмарних системах

(тема)

Виконав:

студент II курсу, групи СПм-21-2

Єрохіна Б. О.
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Волк М. О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти Другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Єрохіну Богдану Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Моделі паралельного та розподіленого моделювання в хмарних системах

затверджена наказом по університету від " 03 " квітня 2023 р. № 318 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 17 травня 2023 р.

3. Вхідні дані до роботи хмарні системи, паралельні та розподілені системи, гіпервізор Microsoft Hyper-V, система керування базами даних PostgreSQL, мова програмування Golang

4. Перелік питань, що потрібно опрацювати у роботі

1	Провести аналіз сучасних робіт в галузі
2	Обрати технологічний стек
3	Розробити структурні та функціональні моделі компонентів.
4	Створити програмні компоненти
5	Проведення експериментів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

Слайдів презентації – 18 шт.

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Провести аналіз сучасних робіт в галузі	04.04.23-07.04.23	
2	Обрати хмарні засоби для реалізації	08.04.23-13.04.23	
3	Створити програмні компоненти	14.04.23-18.04.23	
4	Розробити структурні та функціональні моделі	19.04.23-25.04.23	
5	Проведення експериментів.	26.04.23-03.05.23	
6	Оформлення матеріалів кваліфікаційної роботи	04.05.23-08.05.23	
7	Подання кваліфікаційної роботи керівникові та її	10.05.23-11.05.23	
	попередній захист		
8	Подання кваліфікаційної роботи на рецензування	12.05.23-16.05.23	

Дата видачі завдання 03 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Волк М.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 71 с., 18 рис., 4 табл., 2 дод., 25 джерел.

ХМАРНА СИСТЕМА, ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ, РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ, ВІРТУАЛІЗАЦІЯ, POSTGRESQL, MICROSOFT HYPER-V, GOLANG, СЕРВЕР, REST API, БАЗИ ДАНИХ

Метою кваліфікаційної роботи є аналіз існуючих моделей хмарних обчислень, вибір технологічного стеку для реалізації хмарної системи розподіленого обчислення та подальше тестування роботи цієї системи.

У ході виконання кваліфікаційної роботи для досягнення мети було проведено аналіз актуальних моделей хмарних систем, було розглянуто, що собою представляють системи паралельного та розподіленого обчислення, проведено аналіз і розглянуто деякі дослідження для вибору оптимального технологічного стеку для даної системи. Після проведення аналізу і формуванню відповідних висновків було розроблено структуру та алгоритм для розробки хмарної системи розподіленого генерування, збору та обробки даних. Після проведення аналізу та розробки такої системи було проведено тестування даної системи, яка підтвердила її ефективність та надійність.

ABSTRACT

Master’s thesis: 71 pages, 18 figures, 4 tables, 2 appendices, 25 sources.

CLOUD SYSTEM, PARALLEL COMPUTING, DISTRIBUTED COMPUTING, VIRTUALIZATION, POSTGRESQL, MICROSOFT HYPER-V, GOLANG, SERVER, REST API, DATABASE

The purpose of the qualification work is to analyse existing cloud computing models, select a technological stack for the implementation of a cloud distributed computing system and further test the operation of this system.

During the qualification work to achieve the goal, an analysis of current models of cloud systems was carried out, it was considered what parallel and distributed computing systems are, an analysis was carried out and some studies were considered to select the optimal technological stack for this system. After analysing and forming the appropriate conclusions, the structure and algorithm for developing a cloud distributed generation system were developed, data collection and processing. After analysing and developing such a system, this system was tested, which confirmed its effectiveness and reliability.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTTP – HyperText Transfer Protocol

BPaaS – Business process as a service;

IaaS – Infrastructure as a service;

PaaS – Platform as a service;

SaaS – Software as a service

IT – Information technologies

AWS – Amazon Web Services

GCP – Google Cloud Platform

VMCP – Virtual Machine Cloud Platforms

CUDA – Compute Unified Device Architecture

MPI – Message Passing Interface

HDFS – Hadoop Distributed File System

VDI – Virtual Desktop Infrastructure

VHD – Virtual Hard Drive

VMDK – Virtual Machine Disk

NAT – Network Address Translation

SAN – Storage Area Network

СКБД – Система керування базами даних

OLTP – Online Transaction Processing

JVM – Java Virtual Machine

HTML – HyperText Markup Language

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

ВСТУП

В останні роки зростає значення хмарних обчислень у багатьох галузях, зокрема у сфері досліджень та моделювання складних систем. Це пов'язано з можливістю використання великої кількості ресурсів для обчислень та забезпечення високої продуктивності.

Моделювання складних систем в хмарних середовищах потребує використання паралельних та розподілених алгоритмів, що дозволяють розв'язувати складні задачі швидше та ефективніше. Тому, актуальним є вивчення та аналіз моделей паралельного та розподіленого моделювання в хмарних системах.

Метою даної дипломної роботи є дослідження та порівняння різних моделей паралельного та розподіленого моделювання, їх особливостей та переваг у хмарних середовищах, а також розробка ефективних методів та моделей для паралельного та розподіленого моделювання в хмарних системах, що дозволить забезпечити оптимальне використання обчислювальних ресурсів та підвищити продуктивність систем.. Для досягнення цієї мети було розглянуто теоретичні аспекти паралельного та розподіленого моделювання, а також проведені практичні експерименти для порівняння різних підходів.

З одного боку, паралельне та розподілене моделювання може підвищити продуктивність та швидкість обчислень, а також забезпечити масштабованість та гнучкість використання ресурсів хмарних систем. З іншого боку, ці моделі можуть бути складні в реалізації та потребувати значних зусиль для забезпечення надійності та безпеки.

У наукових дослідженнях та практичних застосуваннях паралельне та розподілене моделювання в хмарних системах вже використовувалося, проте є багато аспектів, що ще не досліджені та потребують подальшого розвитку. У кваліфікаційній роботі було досліджено різні аспекти використання паралельних та розподілених моделей у хмарних системах, такі як алгоритми розподіленого обчислення, проблеми масштабування та ефективного використання ресу-

рсів, а також розробка оптимальної архітектури для паралельного та розподіленого моделювання в хмарних системах.

Тема роботи обумовлена швидким розвитком хмарних технологій та зростаючим попитом на обчислювальні ресурси, що можуть забезпечити високу продуктивність та ефективність роботи різноманітних додатків та систем.

Одним з викликів, що стоять перед сучасними хмарними системами, є необхідність оптимізації використання обчислювальних ресурсів та забезпечення їх максимальної продуктивності. У зв'язку з цим, паралельне та розподілене моделювання стають все більш важливими компонентами хмарних систем, які можуть забезпечити високу ефективність та швидкодію роботи систем.

Додатково, використання паралельних та розподілених моделей у хмарних системах може допомогти вирішити такі проблеми, як недостатня масштабованість систем, проблеми з безпекою та конфіденційністю даних, а також може забезпечити більш ефективне використання ресурсів та зменшення витрат на обслуговування систем.

Головними завданнями в кваліфікаційній роботі був, аналіз літератури з питань паралельного та розподіленого моделювання в хмарних системах, опис існуючих підходів та методів, розробка та реалізація моделей паралельного та розподіленого моделювання для хмарних систем, що дозволяє забезпечити ефективне використання обчислювальних ресурсів та підвищити продуктивність систем, аналіз та порівняння результатів експериментів з реалізованими моделями з існуючими методами паралельного та розподіленого моделювання для хмарних систем, дослідження можливостей використання розподіленого моделювання для забезпечення безпеки та конфіденційності даних в хмарних системах, розробка рекомендацій щодо використання моделей паралельного та розподіленого моделювання в хмарних системах для досягнення оптимального використання ресурсів та підвищення продуктивності систем.

Практичним завданням роботи була розробка та реалізація ефективних методів та моделей для паралельного та розподіленого моделювання в хмарних системах, що дозволить забезпечити оптимальне використання обчислюваль-

них ресурсів та підвищити продуктивність систем. Це особливо важливо у випадку обробки великих обсягів даних та розгортання розподілених систем у хмарних середовищах.

Результати дослідження можуть бути використані для оптимізації роботи хмарних систем та підвищення їхньої продуктивності. Використання ефективних моделей та методів дозволить знизити витрати на обчислювальні ресурси та забезпечити швидкий доступ до даних у хмарних середовищах.

Ця робота може бути корисною для спеціалістів з області хмарних технологій та розподілених систем, що займаються розробкою та підтримкою хмарних сервісів. Розуміння принципів паралельного та розподіленого моделювання в хмарних системах дозволить їм вдосконалити свої навички та підвищити кваліфікацію.

Дослідження може стати основою для подальших наукових досліджень в області розподілених систем та хмарних технологій. Розроблені методи та моделі можуть бути вдосконалені та адаптовані до різних випадків застосування, що відкриває широкі можливості для подальших наукових досліджень у цій області.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Хмарні обчислення

Хмарні обчислення - це модель надання послуг, яка дозволяє отримати доступ до комп'ютерних ресурсів через Інтернет або в локальній мережі. Відбувається це за допомогою віртуалізації обчислювальних ресурсів, мереж та зберігання даних, що дозволяє розподілити їх використання між користувачами.

Усі ресурси, такі як сервери, зберігання даних, програмне забезпечення, мережеві ресурси та інші, знаходяться на віддалених серверах або на серверах підприємства у випадку приватних хмар, які керуються провайдерами або локальними адміністраторами хмарних послуг. Користувачі можуть звернутися до цих ресурсів за допомогою Інтернету і отримувати доступ до необхідного обчислювального середовища в режимі реального часу.

Хмарні обчислення дозволяють користувачам зменшити витрати на придбання та підтримку обладнання, зменшити час на налагодження та належне функціонування інфраструктури, підвищити ефективність та мобільність, а також забезпечити більшу масштабованість використання ресурсів.

Хмарні обчислення є дуже гнучкими, оскільки дозволяють користувачам використовувати лише ті ресурси, які необхідні для вирішення конкретних завдань, та масштабувати їх у випадку потреби. Крім того, користувачі можуть вибирати з різних видів хмарних послуг, що відповідають їхнім потребам, включаючи публічні, приватні та гібридні хмарні сервіси.

Хмарні обчислення застосовуються в різних сферах, таких як бізнес, освіта, наука, медицина, фінанси та інші. Наприклад, бізнес може використовувати хмарні обчислення для забезпечення безпеки даних, підвищення продуктивності та зниження витрат на IT-інфраструктуру. Освітні заклади можуть використовувати хмарні сервіси для доступу до онлайн-курсів та навчальних матеріалів.

Хмарні обчислення є досить новою технологією, і вони все ще розвиваються. Із зростанням попиту, як видно в таблиці (таблиця 1.1) [1] на хмарні сервіси, очікується, що вони стануть ще більш гнучкими та ефективними, і будуть використовуватися для вирішення більш складних завдань.

Таблиця 1.1 – Прогноз доходів від публічних хмарних сервісів у всьому світі (мільярди доларів США)

	2018	2019	2020	2021	2022
Хмарні сервіси бізнес-процесів (BPaaS)	45.8	49.3	53.1	57.0	61.1
Послуги інфраструктури хмарних додатків (PaaS)	15.6	19.0	23.0	27.5	31.8
Хмарні сервіси додатків (SaaS)	80.0	94.8	110.5	126.7	143.7
Хмарне управління та безпека	10.5	12.2	14.1	16.0	17.9
Послуги інфраструктури хмарних систем (IaaS)	30.5	38.9	49.1	61.9	76.6
Загальний ринок	182.4	214.3	249.8	289.1	331.2

Одним з головних переваг хмарних обчислень є можливість забезпечення високої рівня доступності та надійності. Віддалені сервери та обчислювальні ресурси можуть забезпечити автоматичне резервне копіювання даних, а також можливість автоматичного переміщення ресурсів з одного сервера на інший у разі виникнення неполадок. Це дозволяє забезпечувати високу доступність та надійність навіть у разі виникнення проблем з окремими ресурсами.

Іншою важливою перевагою хмарних обчислень є можливість використання різних видів сервісів та ресурсів без необхідності власноруч здійснювати їх розгортання та налаштування. Наприклад, підприємство може використовувати хмарний сервіс для забезпечення необхідних функцій та послуг, таких як електронна пошта, спільний доступ до документів, відеоконференції та інші.

Однак, при використанні хмарних обчислень, існують питання щодо при-

ватності та безпеки даних. Оскільки дані зберігаються на віддалених серверах, існує ризик їхнього несанкціонованого доступу та втрати. Тому, важливо вибирати провайдерів хмарних послуг, які гарантують високий рівень безпеки та приватності даних.

Хмарні обчислення можна класифікувати за декількома параметрами.

За моделлю обслуговувань[2]:

- інфраструктура як сервіс – це вид хмарних обчислень надає користувачам доступ до обчислювальної інфраструктури, такої як сервери, зберігання даних, мережі та інші обчислювальні ресурси. Користувачі можуть використовувати ці ресурси для розгортання своїх додатків та послуг;
- платформа як сервіс – це вид хмарних обчислень надає користувачам доступ до платформи, яка дозволяє розробляти та розгортати свої додатки в хмарному середовищі. Користувачі можуть використовувати інструменти, сервіси та фреймворки, які надає платформа, для розробки своїх додатків;
- програмне забезпечення як сервіс – це вид хмарних обчислень надає користувачам доступ до програмного забезпечення через Інтернет. Користувачі можуть використовувати це програмне забезпечення без необхідності його розгортання та обслуговування. Прикладами SaaS є Google Docs, Microsoft Office 365 та Dropbox.

На рисунку 1.1 видно, як саме розподілені хмарні обчислення за моделлю обслуговування [3]

Рисунок 1.1 – Види хмарних сервісів за моделлю обслуговувань

Існує багато провайдерів хмарних послуг. На рисунку 2 зображені деякі з найбільш популярних вендорів хмар [3]:

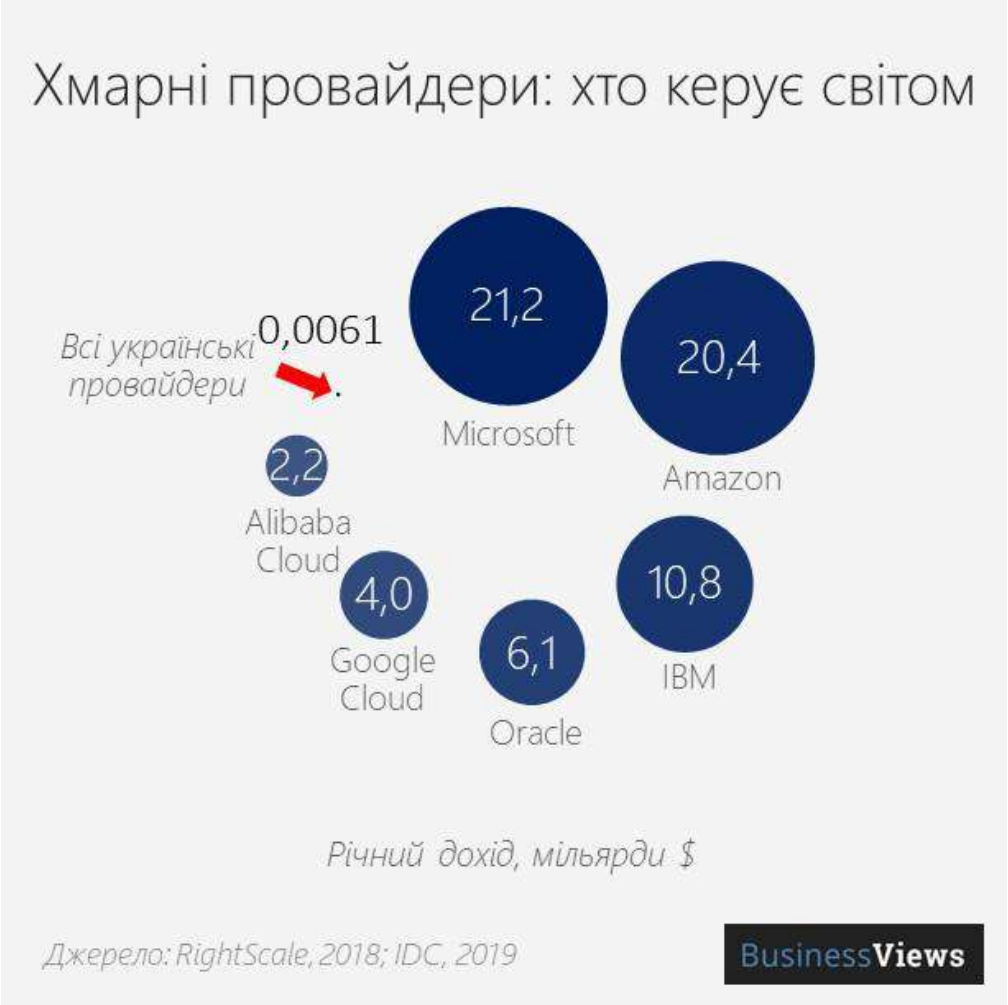

Рисунок 1.2 – Деякі з найбільш популярних вендорів хмарних послуг

Кожен з цих провайдерів пропонує різні рівні обчислювальної потужності, можливості масштабування та функціональність, що відповідає різним потребам користувачів. Для вибору найбільш підходящого провайдера важливо враховувати потреби вашої компанії, вартість послуг та якість обслуговування.

Крім того, на вибір провайдера впливає також його регіональна наявність, рівень безпеки, підтримка та доступність додаткових сервісів.

Ось деякий список послуг, які надають вищезгадані вендори хмарних обчислень:

- Amazon Web Services (AWS) є найбільш популярним вендором хмар. AWS пропонує широкий спектр послуг, включаючи обчислювальні, зберігання,

мережеві та інші послуги. AWS також має найбільшу глобальну інфраструктуру з дата-центрами у багатьох країнах світу;

- Microsoft Azure є другим за популярністю вендором хмар, пропонуючи широкий спектр послуг для підприємств будь-якого розміру. Azure має багато інтегрованих сервісів, таких як Azure DevOps, Azure Machine Learning та інші, що забезпечують підтримку сучасних розробок та аналізу даних;

- Google Cloud Platform (GCP) є ще одним популярним вендором хмар, з фокусом на швидкість та ефективність роботи. GCP має сильні бібліотеки для машинного навчання та інших розробок, а також велику кількість додаткових сервісів, таких як Google BigQuery, що забезпечують підтримку аналітики даних та рішень бізнесу;

- IBM Cloud та Oracle Cloud Infrastructure також є відомими вендорами хмар. IBM Cloud пропонує широкий спектр обчислювальних послуг, зокрема підтримку блокчейн технологій та інших сучасних розробок. Oracle Cloud Infrastructure надає підтримку для великих корпорацій зі складними інфраструктурними потребами, зокрема в галузі баз даних.

За архітектурою [4]:

- публічні хмари - це хмарна інфраструктура, яка доступна для широкого загалу або співробітникам корпорації і належить організації, що надає хмарні сервіси;

- приватні хмари – це хмари, які створені для виняткового використання однією групою користувачів. Вони можуть бути побудовані і управлятися власною IT-службою компанії або постачальником хмарних сервісів. Це дозволяє організації забезпечувати вищу рівень безпеки, контролювати доступ до даних та ресурсів і зменшувати ризики порушення безпеки даних;

- приватні хмари можуть бути розгорнуті як власноруч, так і за допомогою зовнішніх постачальників хмарних послуг. В останньому випадку, постачальник хмарних послуг надає платформу для створення приватної хмари на інфраструктурі, що належить клієнту, та забезпечує її підтримку і обслуговування.

За типом платформи:

- хмарні платформи на віртуальних машинах;
- хмарні платформи на контейнерах.

Хмарні платформи на віртуальних машинах (VMCP) - це хмарні обчислювальні платформи, які надають користувачам можливість запускати віртуальні машини на інфраструктурі хмари. Вони забезпечують апаратний рівень віртуалізації, який дозволяє розгортати віртуальні машини з різними операційними системами і конфігураціями на обладнанні, що знаходиться в хмарі.

Користувачі можуть керувати своїми віртуальними машинами на хмарній платформі, запускаючи, зупиняючи і перенаправляючи їх на різні фізичні сервери. Такі платформи дозволяють користувачам зосередитися на розгортанні та управлінні програмними додатками, не витрачаючи час на управління інфраструктурою.

Окрім того, хмарні платформи на віртуальних машинах надають користувачам гнучкість та масштабованість, оскільки вони можуть легко змінювати розмір своїх віртуальних машин та кількість ресурсів, які вони використовують, в залежності від потреб. Це дозволяє компаніям ефективно використовувати обчислювальні ресурси, забезпечуючи достатню потужність в обчислювальних процесах при мінімальних витратах.

Віртуалізація може забезпечити значні переваги в хмарних обчисленнях, уможливлюючи міграцію віртуальних машин для балансування навантаження в центрі обробки даних. Крім того, міграція віртуальних машин забезпечує надійні та швидко реагуючі центри обробки даних. Міграція віртуальних машин розвинулась із методів міграції процесів .

Нещодавно Xen і VMWare реалізували «живу» міграцію віртуальних машин, яка передбачає надзвичайно короткий час простою від десятків мілісекунд до секунди. Кларк та ін. зазначив, що міграція всієї ОС і всіх її додатків як єдиного цілого дозволяє уникнути багатьох труднощів, з якими стикається підхід до міграції на рівні процесу, і проаналізував переваги живої міграції

віртуальних машин.

Основні переваги міграції віртуальних машин – уникати гарячих точок; однак це непросто. Наразі для виявлення гарячих точок робочого навантаження та ініціювання міграції бракує гнучкості, щоб реагувати на раптові зміни робочого навантаження. Крім того, стан пам’яті має передаватись послідовно та ефективно, з комплексним урахуванням ресурсів для додатків та фізичних серверів.

Хмарні платформи на контейнерах - це тип хмарних обчислень, який використовує контейнеризацію для забезпечення ізольованого середовища для розгортання і запуску додатків. Контейнеризація дозволяє виконувати додатки в стандартизованих середовищах, які можуть бути перенесені з одного середовища в інше без зміни налаштувань та залежностей.

Хмарні платформи на контейнерах зазвичай базуються на системі керування контейнерами, такі як Docker або Kubernetes. Вони забезпечують швидкий та ефективний процес розгортання додатків та масштабування їх ресурсів.

Контейнеризація також дозволяє зменшити ризик конфліктів між додатками, що запущені на одному сервері, тому що кожен контейнер має своє власне ізольоване середовище. Крім того, хмарні платформи на контейнерах дозволяють зменшити час простою додатків та забезпечити більшу доступність, оскільки вони можуть автоматично масштабувати ресурси в залежності від потреб користувачів.

Усі види хмарних обчислень мають свої переваги та недоліки, і вибір конкретної моделі залежить від потреб користувачів та конкретної задачі.

1.2 Паралельні та розподілені хмарні обчислення

Паралельні обчислення [5] - це процес обчислення, який використовує більше одного процесора або ядра, щоб виконувати обчислення одночасно. Паралельні обчислення можуть бути використані для виконання складних обчис-

лювальних завдань, які вимагають великої кількості ресурсів, таких як обробка великих обсягів даних або створення складних моделей машинного навчання.

У паралельних обчисленнях, завдання розбивається на частини, які можуть бути виконані паралельно на різних процесорах або ядрах. Процесори, створені останнім часом такими компаніями, як Intel, AMD і ARM, реалізують паралельні обчислення з використанням кількох процесорних ядер . Це дозволяє зменшити час виконання завдання та забезпечити більш ефективне використання ресурсів.

Існує кілька різних методів паралельних обчислень, включаючи розподілені обчислення, багатопоточність та розпаралелювання на рівні операцій. У багатопоточних обчисленнях, кожен потік виконує окрему операцію, що дозволяє забезпечити паралельне виконання багатьох операцій одночасно. Розпаралелювання на рівні операцій означає, що окремі операції розбиваються на більші кількості менших операцій, які можуть бути виконані паралельно.

Для розробки та розгортання паралельних обчислень необхідні високі технічні знання та досвід. Потрібно розуміти різні методи паралельного програмування, такі як MPI, OpenMP, CUDA, OpenCL тощо. Крім того, необхідно володіти знаннями про розподілених системах, мережеву інфраструктуру та забезпечення безпеки в мережі.

Паралельні обчислення можуть бути використані в багатьох галузях, включаючи науку, фінанси, інженерію, медицину та інформаційні технології. Наприклад, у науці можуть використовуватися паралельні обчислення для обробки великих обсягів даних з експериментів або симуляцій.

У фінансовій галузі, паралельні обчислення можуть використовуватися для розрахунку складних фінансових інструментів, таких як опціони та ф’ючерси. У медицині, паралельні обчислення можуть використовуватися для аналізу великих обсягів медичних даних, таких як зображення МРТ або скринінги геномів.

Узагалі, паралельні обчислення дозволяють розглядати проблеми, що були раніше недосяжні з точки зору можливостей обчислювальної техніки та від-

кривають нові можливості для розвитку науки та інновацій.

Однією з переваг паралельних обчислень є можливість зменшити час виконання завдання та збільшити продуктивність, що може бути важливо для великих проектів з великою кількістю даних. Однак, існують певні виклики та обмеження, пов'язані з розробкою та розгортанням паралельних обчислень, такі як складність управління ресурсами та потреба в спеціальному обладнанні та програмному забезпеченні.

Паралельне моделювання хмарних систем відноситься до використання паралельних обчислювальних ресурсів для моделювання різних хмарних архітектур та поведінки хмарних систем.

Паралельне моделювання використовує багато додаткових ресурсів, що дає змогу ефективніше вирішувати проблеми масштабування, продуктивності та безпеки в хмарних системах, хоча і можуть виникнути деякі проблеми, а саме в таких системах є ризик колізії даних, але метод вирішення цієї проблеми запропоновано в статті за авторством доцента кафедри ЕОМ Ткачова В. М..[6]

Розподілені обчислення [5] - це обчислювальний процес, що використовує кілька комп'ютерів, які працюють разом, щоб виконати завдання. У розподілених обчисленнях, завдання розбивається на частини, які розподіляються між різними комп'ютерами, які працюють над цими частинами паралельно.

Розподілені обчислення можуть бути використані для виконання завдань, що потребують багато ресурсів, таких як обробка великих обсягів даних, створення складних моделей машинного навчання або симуляції складних процесів.

Однією з особливостей розподілених обчислень є те, що кожен комп'ютер може працювати незалежно від інших, і, якщо один комп'ютер не працює належним чином, інші комп'ютери можуть продовжувати роботу.

На рисунку 1.3 наведено примітивний варіант зображення моделі розподіленого та паралельного моделювання в хмарних системах

Розподілені обчислення можуть бути реалізовані за допомогою різних технологій і архітектур, таких як клієнт-серверні системи, peer-to-peer мережі або хмарні обчислення.

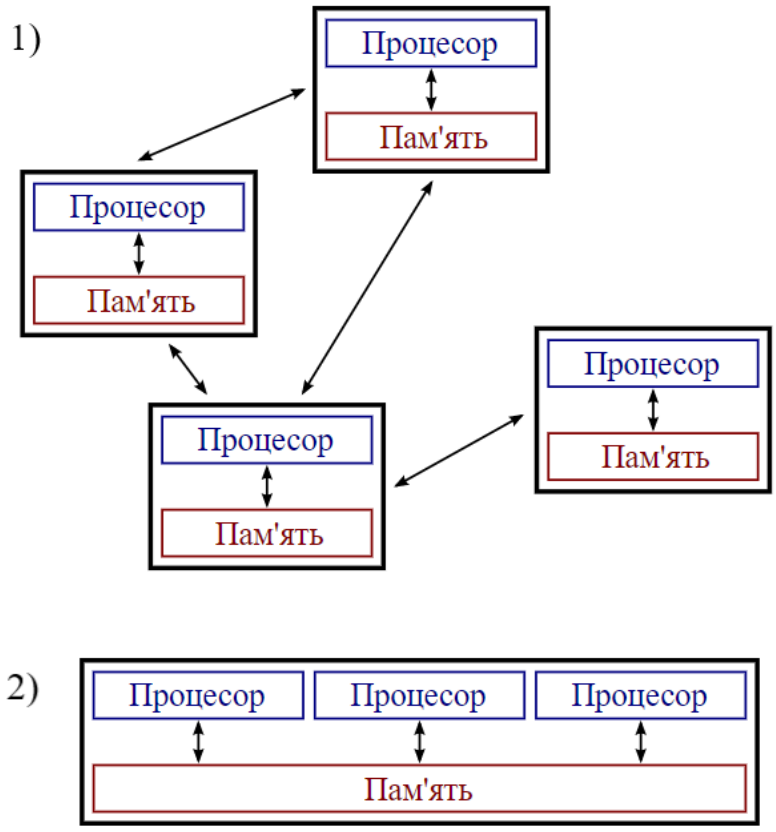

Рисунок 1.3 – Моделі розподіленого (1) та паралельного (2) моделювання в хмарних системах

Однією з найбільш поширених технологій розподілених обчислень є Hadoop, який є відкритим програмним забезпеченням для обробки великих обсягів даних. Hadoop використовує розподілену файлову систему (HDFS) та фреймворк для обробки даних MapReduce, що дозволяє розподіляти обчислювальні завдання між багатьма вузлами.

Інші технології розподілених обчислень включають Apache Spark, Apache Storm, Apache Flink, та інші. Ці технології використовуються в різних сферах, включаючи аналітику даних, обробку відео, створення ігор, машинне навчання та багато іншого.

Однією з переваг розподілених обчислень є можливість швидкої обробки великих обсягів даних та виконання складних обчислювальних завдань. Також розподілені обчислення забезпечують більш високу надійність та стійкість до збоїв, оскільки завдання можуть бути розподілені між багатьма комп'ютерами,

які можуть замінювати один одного в разі виникнення проблем.

Однак, в розподілених обчисленнях можуть виникати складнощі з синхронізацією та управлінням даними між різними вузлами. Також можуть виникати проблеми з безпекою даних та захистом від зловмисних атак.

1.3 Сучасні засоби паралельних та розподілених хмарних обчислень

Сучасні засоби паралельних та розподілених хмарних обчислень надають користувачам можливість легко та ефективно виконувати складні обчислення з використанням багато-ядерних процесорів та масштабованих обчислювальних кластерів. Ось деякі з найпопулярніших засобів:

- Amazon Web Services - це хмарний сервіс, який дозволяє користувачам орендувати обчислювальні ресурси від Amazon. AWS надає різноманітні послуги, такі як Elastic Compute Cloud, Simple Storage Service, та Elastic MapReduce, які дозволяють використовувати обчислювальні ресурси за потребою;
- Google Cloud Platform - це хмарний сервіс, який надає користувачам доступ до великого діапазону обчислювальних ресурсів, таких як Google Compute Engine, Google App Engine, та Google Cloud Storage;
- Microsoft Azure - це хмарний сервіс, який надає користувачам доступ до широкого спектру обчислювальних та інших послуг, таких як Azure Virtual Machines, Azure App Service, та Azure Storage;
- IBM Cloud - це хмарний сервіс, який надає користувачам доступ до обчислювальних та інших послуг, таких як IBM Cloud Virtual Servers, IBM Cloud Functions, та IBM Cloud Object Storage;
- Apache Hadoop - це відкрите програмне забезпечення для обробки великих обсягів даних з використанням розподілених систем. Hadoop включає такі компоненти, як Hadoop Distributed File Systemта MapReduce, які дозволяють виконувати обчислення на масштабованих кластерах;
- Apache Spark - це відкрите програмне забезпечення для розподілених обчислень з використанням масштабованих кластерів. Spark підтримує різні

мови програмування, такі як Java, Scala, та Python, та має підтримку для різних джерел даних, включаючи Hadoop Distributed File System , Cassandra, та Amazon S3;

- Kubernetes - це відкрите програмне забезпечення для автоматизації розгортання, масштабування та управління контейнеризованими додатками. Kubernetes дозволяє легко керувати ресурсами та масштабувати додатки на масштабованих кластерах;
- Docker - це відкрите програмне забезпечення, яке дозволяє ізолювати додатки в контейнери, що дозволяє легко розгортати та виконувати їх на будь-яких обчислювальних ресурсах. Docker дозволяє ефективно використовувати ресурси, так як кожен контейнер має свої власні ізольовані обчислювальні ресурси.

Ці засоби дозволяють розробникам та дослідникам легко та ефективно використовувати паралельні та розподілені обчислення на масштабованих кластерах хмарних сервісів, що дозволяє розв'язувати складні завдання обчислювальної науки, штучного інтелекту та аналізу даних.

1.4 Сучасні системи паралельного та розподіленого збору та обробки даних з метеорологічних даних

Сучасні системи паралельного та розподіленого збору та обробки даних з метеорологічних даних можуть включати в себе різноманітні технології та інструменти для збору, зберігання, обробки та аналізу великих обсягів даних.

Один з прикладів такої системи може включати в себе деякі з наступних елементів:

- системи зберігання даних - ці системи зазвичай використовуються для зберігання великих обсягів даних про метеорологічні умови. Для цього можуть використовуватися бази даних, облака даних або системи зберігання на основі Hadoop;
- фреймворки для обробки даних - ці фреймворки дозволяють виконува-

ти різноманітні операції з обробки даних, такі як фільтрування, агрегація, маппінг та злиття. Найбільш популярними з них є Apache Hadoop, Apache Spark та Apache Storm;

- алгоритми машинного навчання - ці алгоритми можуть використовуватися для прогнозування метеорологічних умов на основі даних, які були зібрані та оброблені. Деякі з таких алгоритмів включають в себе лінійну регресію, деревні мережі, дерева рішень, глибоке навчання та нейронні мережі;
- системи розподіленого збору та обробки даних - ці системи дозволяють збирати та обробляти дані з різних джерел у режимі реального часу. Для цього можуть використовуватися системи, такі як Apache Kafka, Apache Flume та Apache Nifi.

Загалом, сучасні системи паралельного та розподіленого збору та обробки даних з метеорологічних даних дозволяють збирати, зберігати та аналізувати великі обсяги даних у режимі реального часу. Ці системи використовують різноманітні технології та інструменти, що дозволяє забезпечити точність та швидкість обробки даних, а також підвищити рівень передбачуваності метеорологічних умов.

1.5 Постанова задачі

Метою концепції впровадження хмарної системи паралельного або розподіленого збору даних є покращення ефективності збору та обробки великих обсягів даних.

Хмарні системи дозволяють також зберігати дані в безпечному та доступному для користувачів місці, що робить їх більш ефективними для організацій та бізнесу.

Крім того, концепція використання хмарної системи забезпечує більшу масштабованість та гнучкість, оскільки можна легко додавати нові ресурси в систему, щоб забезпечити потрібну потужність обробки даних;

Завданням розробки хмарної системи паралельного та розподіленого збо-

ру та обробки даних з метеорологічних датчиків є:

- забезпечення інформованості метеорологічних компанії погодних умов в тому числі для подальшої передачі цієї інформації, наприклад, з аеропортами;
- вирішення проблем впливу людського фактору під час збору інформації з датчиків, якби вона проводилася метеорологом, а не комп'ютерною системою;
- забезпечити такими системами метеорологічні служби, задля того, щоб можна було збирати та зберігати ще більше даних аби в майбутньому її можна було проаналізувати і можливо навіть розробити систему яка давала би змогу передбачати те чи інше стихійне лихо.

Хмарна система паралельного та розподіленого збору та обробки інформації має:

- бути досить продуктивною, щоб забезпечувати безперебійну роботу системи;
- бути не дуже дорогою, щоб навіть невеликі компанії могли її використовувати у своїх цілях;
- бути легкою в розробці та подальшій підтримці цієї програми, навіть спеціалістами, які не володіють на високому рівні мовою програмування, яка використовується в даній системі;
- споживати невелику кількість ресурсів, як, наприклад, оперативна пам'ять, аби не робити таку систему більш дорогою, а значить зменшувати її конкурентоспроможність;
- найбільш раціонально споживати ресурси процесора, щоб система в будь-який момент часу була оптимально навантажена;
- бути приватною задля забезпечення максимального доступу до налаштування та конфігурування системи;
- мати архітектуру IaaS для того, щоб можно було більш гнучко налаштовувати кожен сервер.

2 ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ ДЛЯ ХМАРНОЇ СИСТЕМ РОЗПОДІЛЕНОГО ОБЧИСЛЕННЯ

2.1 Загальна інформація про технології віртуалізації

Віртуалізація - це технологія, що дозволяє створювати віртуальні екземпляри обчислювального апаратного забезпечення, операційних систем, програмного забезпечення, мереж та інших обчислювальних ресурсів. Гіпервізори можна поділити на два типи [8] в залежності від їх структури (рисунок 2.1)

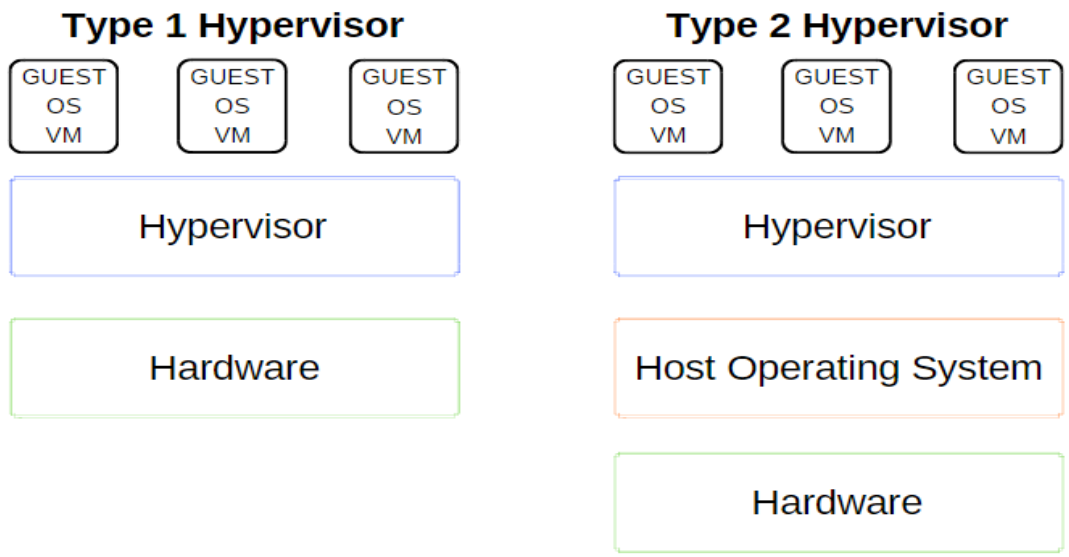

Рисунок 2.1 – Типи гіпервізорів за структурою

. Ось деякі з найбільш відомих технологій віртуалізації:

- VMware vSphere,
- Oracle VM VirtualBox,
- Microsoft Hyper-V.

2.1.1 VMware ESXi

VMware ESXi - це гіпервізор 1 типу, розроблений компанією VMware[9].

Він надає інфраструктуру для розгортання, керування та моніторингу віртуальних машин на фізичних серверах.

Основні компоненти VMware ESXi включають гіпервізор ESXi, управління ресурсами, управління віртуальними машинами, мережеве управління та зберігання даних. Він дозволяє розгортати віртуальні машини на фізичних серверах та керувати ними, надаючи можливість автоматичного масштабування та резервного копіювання.

VMware ESXi є гіпервізором, який надає середовище для запуску віртуальних машин на фізичному сервері. Гіпервізор ESXi дозволяє кільком віртуальним машинам спільно використовувати ресурси фізичного сервера, такі як процесор, пам'ять та зберігання даних.

При запуску сервера з гіпервізором ESXi, він бере на себе контроль над апаратними ресурсами сервера і розподіляє їх між запущеними на ньому віртуальними машинами.

Гіпервізор ESXi надає гнучкість для налаштування віртуальних машин та доступу до них. Він підтримує різноманітні операційні системи та конфігурації мереж, що дозволяє використовувати його в різних сценаріях віртуалізації.

ESXi також надає ряд інструментів для керування віртуальними машинами, такі як управління мережевими налаштуваннями, збереження даних, моніторинг продуктивності та управління безпекою. Крім того, ESXi підтримує віддалений доступ до віртуальних машин, що дозволяє адміністраторам віддалено керувати та моніторити стан віртуальних машин та гіпервізора.

VMware vSphere надає широкий спектр можливостей для віртуалізації, включаючи підтримку різних операційних систем, мережевих налаштувань та інших функцій, необхідних для розгортання та управління віртуальними машинами. Крім того, він забезпечує високу доступність та надійність системи, що є важливим для бізнес-застосувань та критичних застосувань.

Загалом, VMware ESXi є потужним рішенням для віртуалізації та управління інфраструктурою обчислювання, який дозволяє бізнесам ефективно використовувати свої ресурси та знижувати витрати на обладнання та управління

інфраструктурою.

2.1.2 Oracle VM VirtualBox

Oracle VM VirtualBox[10] - це програмне забезпечення гіпервізора 2 типу, яке дозволяє запускати віртуальні машини на різних операційних системах, включаючи Windows, Linux, Macintosh та інші.

Він надає можливість запуску багатьох віртуальних машин на одному фізичному комп'ютері та дозволяє відтворювати різні конфігурації апаратного забезпечення, що допомагає розробникам та тестувальникам при розробці програмного забезпечення.

Основні функції Oracle VM VirtualBox включають можливість налаштування віртуальної машини згідно з потребами користувача, включаючи обсяг оперативної пам'яті, кількість процесорів, тип дискового простору тощо. Також можна створювати знімки віртуальної машини, що дозволяє повертати її до попереднього стану у разі необхідності. Гіпервізор забезпечує ізоляцію віртуальних машин одну від одної, що дозволяє їм працювати незалежно одна від іншої та забезпечує безпеку віртуальних середовищ.

Oracle VM VirtualBox також підтримує різноманітні гостьові операційні системи, такі як Windows, Linux, macOS, Solaris тощо, і може працювати з різними форматами віртуальних дисків, включаючи VDI, VHD, VMDK тощо. Крім того, він підтримує різноманітні мережеві настройки, включаючи NAT, мостове підключення та хмарні мережі.

При запуску віртуальної машини, гіпервізор створює віртуальне середовище для неї, яке включає в себе віртуальний процесор, віртуальну пам'ять, віртуальний диск та інші віртуальні пристрої. Гіпервізор перетворює запити, що надходять від віртуальної машини, на команди, що можуть бути виконані на реальному апаратному забезпеченні, і повертає результати до віртуальної машини.

Окрім цього, Oracle VM VirtualBox надає можливість налаштування па-

раметрів віртуальної машини, включаючи кількість процесорів, обсяг оперативної пам'яті, розмір віртуального диска та інші параметри. Також можна створювати знімки віртуальної машини, що дозволяє повертати її до попереднього стану у разі необхідності.

У загальному, Oracle VM VirtualBox є потужним та гнучким гіпервізором, який дозволяє запускати різні операційні системи на одному фізичному комп'ютері, що допомагає розробникам, тестувальникам та іншим користувачам у різних сферах діяльності.

2.1.3 Microsoft Hyper-V

Microsoft Hyper-V[11] -це гіпервізор типу 1, який дозволяє запускати кілька віртуальних машин (ВМ) на одному конкретному комп'ютері. Ця технологія віртуалізації вбудована в операційну систему Windows, яка включає установку декількох гостьових систем на одному хост-комп'ютері.

Hyper-V дозволяє користувачам створювати та керувати віртуальними машинами, що являють собою високе середовище, які спрямовують себе як фізичний комп'ютер із власними оцінками, висловлюваннями та додатками. Hyper-V можна використовувати для різних цілей, таких як консолідація серверів, розробка та тестування, а також аварійне відновлення.

Hyper-V був вперше представлений у Windows Server 2008 і відтоді оновлювався та покращувався у випуску Windows. Це вибір для підприємств та IT-фахівців, які розширюють можливості віртуалізації, створюють як підвищену ефективність, гнучкість та масштабованість.

Hyper-V надає безліч функцій та можливостей для підтримки віртуалізації, у тому числі:

- Hyper-V використовує можливості апаратної віртуалізації сучасних процесорів для створення та керування віртуальними машинами;
- Hyper-V надає інструменти та інтерфейси для створення, налаштування та керування віртуальними машинами;

- Hyper-V включає віртуальні комутатори, які дозволяють віртуальним машинам взаємодіяти один з одним і із зовнішньою мережею;

- Hyper-V підтримує створення віртуальних жорстких дисків (VHD) та віртуальних мереж зберігання даних (SAN) для зберігання даних віртуальних машин.

Hyper-V широко використовується у корпоративних середовищах, де віртуалізація необхідна для консолідації серверів, розподілу ресурсів та аварійного відновлення. Він також використовується розробниками та тестувальниками, яким необхідно створювати віртуальні середовища для розробки та тестування програмного забезпечення. Hyper-V включений до різних випусків Windows Server, а також доступний як окремий продукт Microsoft Hyper-V Server.

2.1.4 Порівняння продуктивності технологій віртуалізації

На рисунку 2.2 приведено порівняння [12] VirtualBox і VMWare за таким параметром, як швидкість виконання базових операцій (наприклад, копіювання пам’яті та виконання обробки чисел у ЦП) з однаковою швидкістю; обидва мають передати якомога більше цієї продуктивності базовій системі, тому цифри мають бути подібними:

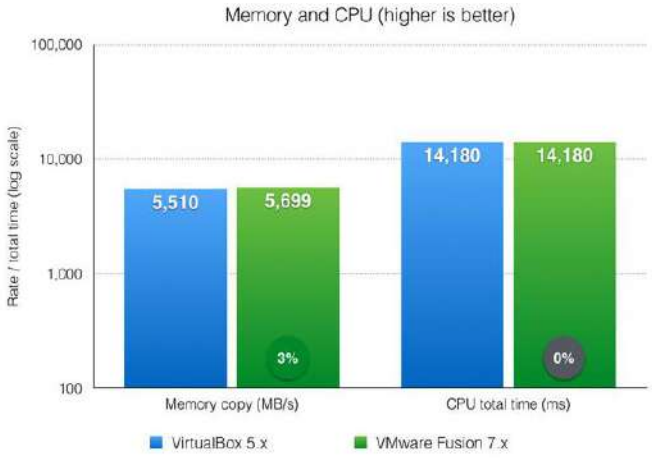

Рисунок 2.2 – Порівняння використання процесору та пам’яті гіпервізорами

VMware і VirtualBox є непереборними, коли йдеться про необроблену

пам’ять і продуктивність ЦП, і цього слід очікувати сьогодні, оскільки обидва рішення (як і більшість інших рішень віртуалізації) можуть використовувати функції сучасних процесорів Intel і сучасних чіпсетів у повному обсязі. Робочі навантаження з великими навантаженнями на процесор або оперативну пам’ять мають працювати однаково, хоча VMware Fusion має невелику перевагу.

Порівнявши Microsoft Hyper-V та VMware vSphere [13]за деякими параметрами (таблиця 2.1), можна зрозуміти, що для нашої системи більше підходить гіпервізор Microsoft Hyper-V.

Таблиця 2.1 – Порівняльна характеристика Microsoft Hyper-V та VMware vSphere за деякими з параметрів можливостей віртуалізації

Система	Ресурс	Microsoft Hyper-V 2019	VMware vSphere 6.7 Free
Хост	Логічні процесори	512	768
	Фізична пам'ять	24 ТВ	4ТВ
	Віртуальні процесори на хост	2048	4096
VM	Віртуальні процесори на віртуальну машину	240 Gen 2 64 Gen 1	8
	Пам'яті на віртуальну машину	12 ТБ Gen 2 1 ТВ Gen 1	6128GB
	Максимум віртуального диска	64 ТБ VHDX 2040 ГБ VHD	62ТВ
Кластер	Максимальна кількість вузлів	64	Н/Д
	Максимальна кількість віртуальних машин	8000	Н/Д

2.2 Загальна інформація про системи керування базами даних

2.2.1 MongoDB

MongoDB[14] — це безкоштовна міжплатформна документоорієнтована програма з відкритим кодом. Це база даних NoSQL, яка використовує JSON-подібні документи без схем. Спеціальні запити, індексація та агрегація в реальному часі забезпечують ефективні способи доступу та аналізу даних. За своєю суттю це розподілена база даних, яка забезпечує високу доступність, горизонтальне масштабування та географічний розподіл.

2.2.2 MySQL

MySQL[15] — це система керування реляційною базою даних з відкритим кодом. MySQL використовується в багатьох веб-додатках і на веб-сайтах, і є кілька комерційних баз даних, сумісних з ним. Користувачі взаємодіють з базою даних через SQL. Стандартна версія MySQL використовує InnoDB як механізм зберігання. Однак архітектура механізму зберігання MySQL є підключаємою, що дозволяє використовувати спеціалізований механізм зберігання даних. MySQL активно оновлюється та підтримується з 1995 року, нові версії випускаються кожні 1-3 роки. Існує кілька розгалужень MySQL, особливо MariaDB, яку очолюють оригінальні розробники MySQL. Oracle, яка придбала MySQL у 2010 році, пропонує кілька платних версій СУБД, які пропонують додаткові функції.

2.2.3 Oracle RDBMS

База даних Oracle[16], спочатку реляційна СУБД, перетворилася на конвергентну багатомодельну СУБД, наприклад. він підтримує кілька типів робочого навантаження (OLTP, сховище даних, сховище операційних даних, конце-

нтратор даних), він може керувати та об’єднувати декілька типів даних (реляційні, XML, JSON, просторові, граф властивостей, RDF, текстові, двійкові), що зберігаються всередині або поза базою даних, він підтримує численні типи розгортання, і до нього можна отримати доступ за допомогою кількох API.

2.2.4 PostgreSQL

PostgreSQL[17] — це об’єктно-реляційна база даних на основі Postgres, розроблена Каліфорнійським університетом у Берклі. PostgreSQL - це [17]програмне забезпечення з відкритим кодом під ліцензією PostgreSQL, яке багато людей досі часто називають Postgres. Postgres підтримужться глобальною групою розвитку.

2.2.5 Порівняння систем керування базами даних

Таблиця 2.2 - Порівняльна характеристика СКБД

Критерії	MongoDB	MySQL	Oracle RDBMS	PostgreSQL
1	2	3	4	5
Безпека	Надає різні функції безпеки, такі як автентифікація, авторизація та шифрування.	Забезпечує механізми аутентифікації та шифрування..	Надає широкий спектр функцій, таких як шифрування та автентифікація.	Забезпечує механізми аутентифікації та шифрування. Має репутацію безпечного.
Ліцензування	Має комбінацію ліцензій з відкритим вихідним кодом.	Має комбінацію ліцензій з відкритим вихідним кодом.	Має комерційну ліцензію.	Має ліцензію з відкритим вихідним кодом.

Продовження таблиці 2.2

1	2	3	4	5
Ліцензування	Має комбінацію ліцензій з відкритим вихідним кодом та комерційної ліцензії.	Має комбінацію ліцензій з відкритим вихідним кодом та комерційних ліцензій.	Має комерційну ліцензію.	Має ліцензію з відкритим вихідним кодом.
Типи даних	Підтримує широкий спектр типів даних, включаючи масиви, дати та геопросторові дані.	Підтримує широкий спектр типів даних, включаючи цілі, десяткові та рядкові числа.	Підтримує широкий спектр типів даних, включаючи цілі, десяткові та рядкові числа.	Підтримує широкий спектр типів даних, включаючи цілі, десяткові та рядкові числа.
Архітектура	Дотримується розподіленої архітектури. Забезпечує горизонтальне масштабування.	Дотримується клієнт-серверної архітектури. Горизонтально масштабується .	Дотримується клієнт-серверної архітектури. Забезпечує вертикальне масштабування.	Дотримується клієнт-серверної архітектури. Горизонтально масштабується.

Щоб протестувати деякі з популярних СКБД було написано просту програму Go[18]. Ця програма створює одну таблицю benchmark_data з 6 стовпцями. Вам потрібно створити базу даних benchmark вручну за допомогою create database benchmark.

Ця програма вставляє 10 000 рядків у цю таблицю один за одним. Без пакетних вставок, лише прості вставки.

У цьому тесті було зроблено 5 послідовних пакетів вставок, кожен з 10

000 рядків. Результати відображено на рисунку (рисунок 2.3)

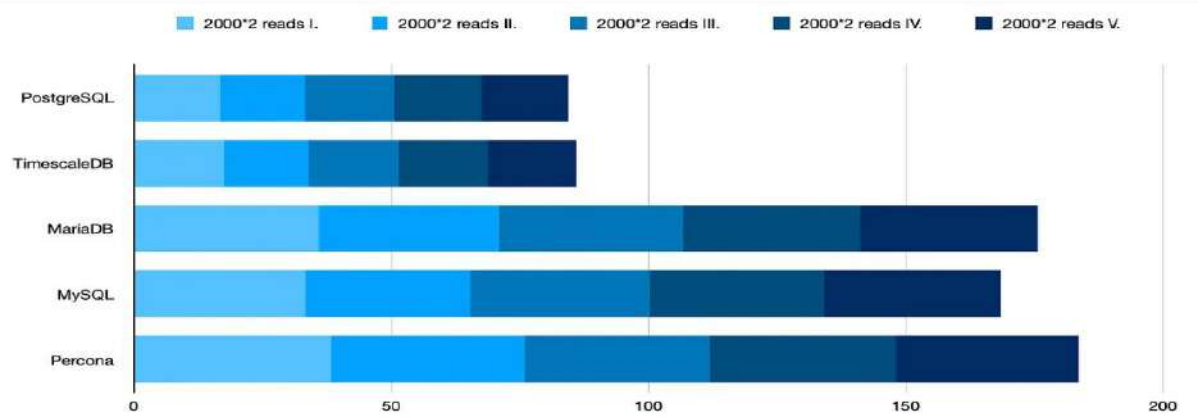

Рисунок 2.3 – Результати тестування реляційних СКБД

Очевидно, що переможцем тут є PostgreSQL. Найменший розмір, найменше використання процесора та пам’яті, найвища швидкість запису та найшвидша швидкість читання. Друге місце займає TimescaleDB.

В іншій статті[19] було порівняно PostgreSQL та MongoDB, як представників СКБД різних типів (рисунок 2.4)

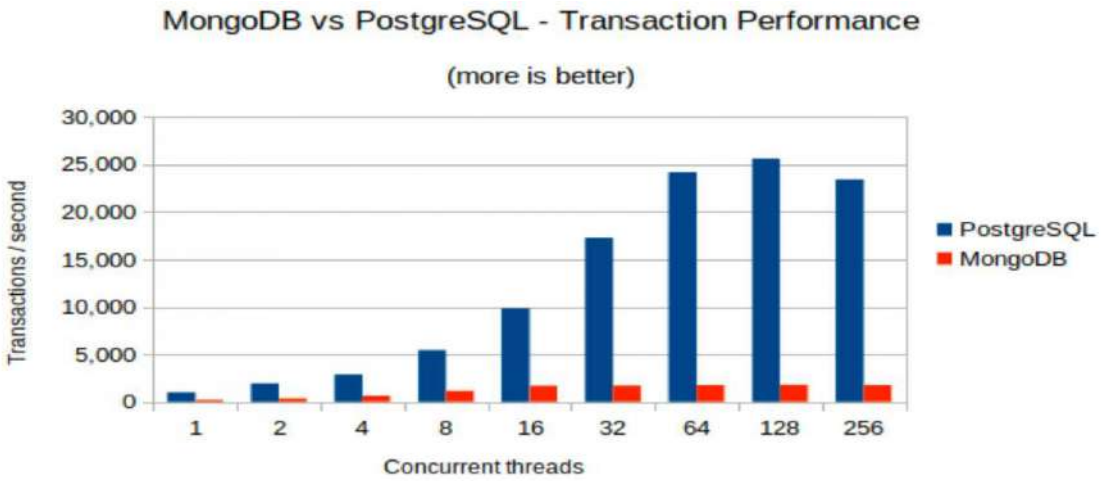

Рисунок 2.4 – Результати тестування MongoDB та PostgreSQL

Система керування базами даних (СКБД) Postgres виявилася в 4-15 разів швидше, ніж MongoDB, під час тестування продуктивності транзакцій, проведеного OnGres, компанією, що спеціалізується на наданні програмного забезпечення та послуг баз даних та спонсорованої EnterpriseDB. Цікаво, що Postgres продемонстрував перевагу в продуктивності тесту онлайн-аналітичної обробки

(OLAP) на основі JSON (JavaScript Object Notation), розробленому спеціально для роботи з даними на основі документів, що є передбачуваною силою MongoDB. Додаткове тестування було здійснено для робочих навантажень оперативної обробки транзакцій (OLTP). Для тесту OLTP використовувався стандартний галузевий тест sysbench, щоб показати, що Postgres працює в середньому втричі швидше, ніж MongoDB. Протягом усього тестування, яке вимірює продуктивність при різних робочих навантаженнях, Postgres незмінно демонструвала більшу продуктивність, ніж MongoDB.

2.3 Загальна інформація про мови програмування

2.3.1 Java

Java[20] - це високорівнева, об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (пізніше придбана компанією Oracle). Вона була випущена в 1995 році і швидко стала однією з найпопулярніших мов програмування у світі.

Java розроблена з орієнтацією на переносимість і розширюваність програм. Її код компілюється в проміжний байт-код, який може бути виконаний на будь-якому комп'ютері з встановленою віртуальною машиною Java (JVM - Java Virtual Machine). Це дозволяє програмам на Java працювати на різних платформах без необхідності перекомпіляції.

Основні характеристики Java:

- Java підтримує об'єктно-орієнтований стиль програмування, що дозволяє організувати програми в окремі об'єкти, що взаємодіють між собою;
- байт-код Java може бути виконаний на будь-якій платформі, на якій присутня віртуальна машина Java (JVM), що робить Java дуже переносимою мовою;
- Java використовує систему збору сміття, що дозволяє автоматично вивільняти пам'ять, що була використана об'єктами, але більше не потрібна;

- Java має вбудовану підтримку багатопотокового програмування, що дозволяє виконувати кілька завдань одночасно.

2.3.2 PHP

PHP[21] (PHP: Hypertext Preprocessor) - це скриптова мова програмування, яка призначена для розробки веб-додатків і динамічних веб-сторінок. Вона була розроблена Расмусом Лердорфом у 1994 році і сьогодні використовується на багатьох веб-сайтах і веб-програмах.

Основні характеристики PHP:

- PHP виконується на стороні сервера, що означає, що код PHP обробляється на веб-сервері перед тим, як відповідь надсилається до клієнта. Клієнт отримує результат вже у вигляді звичайної HTML-сторінки;
- PHP має широкі можливості роботи з різними системами управління базами даних (наприклад, MySQL, PostgreSQL, Oracle і багатьма іншими). Це дозволяє легко створювати веб-додатки, що взаємодіють з базою даних;
- PHP має простий і зрозумілий синтаксис, що дозволяє швидко вивчити мову і розробляти додатки без великого зусилля;
- PHP має велику кількість розширень, які додають нові функціональні можливості до мови. Це дозволяє розробникам використовувати готові рішення і збільшити продуктивність своїх додатків.

2.3.3 Node.js

Node.js[22] - це вільне, відкрите середовище виконання JavaScript на стороні сервера, побудоване на базі двигуна V8 Chrome. Воно дозволяє розробникам використовувати JavaScript для створення веб-додатків і серверних програм.

Основні характеристики Node.js:

- Node.js побудоване на подієвій моделі, що дозволяє обробляти багато

запитів одночасно без блокування виконання коду. Воно використовує асинхронний ввід/вивід, що дозволяє ефективно працювати з мережевими операціями;

- Node.js дозволяє використовувати JavaScript на стороні сервера, що забезпечує єдино образність мови програмування від клієнтської до серверної частини додатка;

- Node.js має багатий набір модулів і пакетів, доступних через систему керування пакетами npm. Це дозволяє розробникам швидко розширювати функціональність своїх додатків за допомогою сторонніх бібліотек;

- Node.js побудоване на швидкому двигуні V8 Chrome, що забезпечує високу продуктивність виконання JavaScript-коду;

- Node.js добре підходить для створення масштабованих додатків. Воно має побудову, що дозволяє обробляти багато запитів одночасно і легко масштабуватись на багатоядерних системах.

2.3.4 Golang

Go (або Golang)[23] - це високорівнева, компільована мова програмування, розроблена в компанії Google. Вона була представлена в 2009 році і швидко набула популярності серед розробників завдяки своїм простоті, ефективності та здатності працювати з багато потоковими програмами.

Основні характеристики Golang:

- Golang має чистий і простий синтаксис, що дозволяє легко вивчити мову та розробляти програми. Вона прагматична і позбавлена зайвих конструкцій.

- Golang була спроектована з метою надати високу продуктивність. Вона має швидке виконання завдяки ефективній роботі з пам'яттю, компіляції в машинний код і оптимізаціям виконання.

- Golang надає вбудовану підтримку багато потокового програмування. Вона має легкий синтаксис для створення і керування багато потоковими програмами, що дозволяє використовувати потоки для паралельного виконання за-

вдань.

- Golang надає потужні і прості інструменти для розробки мережевих додатків. Вона включає в себе пакети для створення серверів, роботи з HTTP, TCP, UDP і іншими протоколами.
- Golang підтримує багато платформ, включаючи Windows, macOS, Linux і інші. Код, написаний на Golang, може бути компільований для різних операційних систем без необхідності внесення змін у вихідний код.

2.3.5 Аналіз продуктивності мов програмування

Для порівняння було обрано декілька основних орієнтирів, які порівнюють загальну продуктивність HTTP-сервера в цих серверних середовищах[24]. Звісно треба мати на увазі, що існує багато факторів, які впливають на продуктивність всього наскрізного шляху запиту/відповіді HTTP, і представлені тут цифри - це лише кілька прикладів, які були зібрані разом, щоб дати основне порівняння.

Для кожного з цих середовищ було написано відповідний код для читання у файлі розміром 64 КБ з випадковими байтами, запустив хеш SHA-256 N разів.

Запустивши 2000 ітерацій з 300 одночасними запитами і лише одним хешем на запит (N = 1) дає нам наступне:

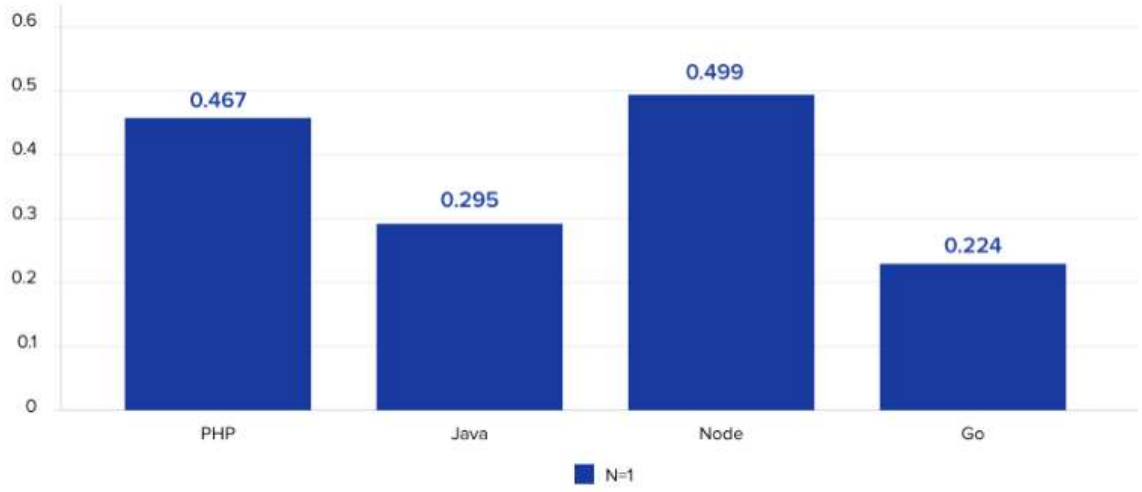

Рисунок 2.5 – Кількість мілісекунд для виконання запиту серед усіх одночасних

запитів

Важко зробити висновок лише з цього графіка, але здається, що з такою кількістю зв'язків та обчислень ми спостерігаємо часи, які більше пов'язані із загальним виконанням самих мов, особливо після вводу-виводу. Зверніть увагу, що мови, які вважаються "скриптовими мовами" (вільний набір тексту, динамічна інтерпретація), працюють повільніше.

Але що станеться, якщо ми збільшимо N до 1000, все ще з 300 одночасними запитами – таке ж навантаження, але в 100 разів більше хеш-ітерацій, а значить збільшимо навантаження на процесор(рисунок 2.6).

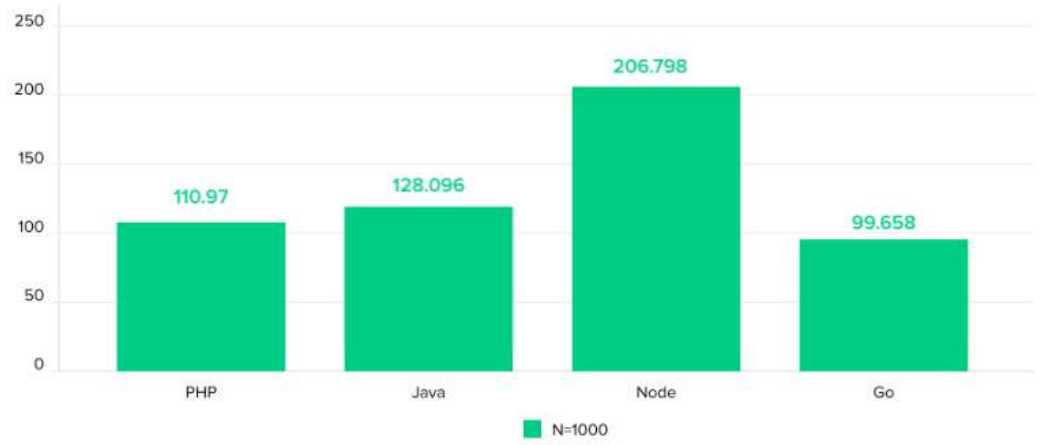

Рисунок 2.6 – Кількість мілісекунд для виконання запиту серед усіх одночасних запитів

Раптово продуктивність Node значно падає, оскільки інтенсивні операції процесора в кожному запиті блокують один одного. І що цікаво, продуктивність PHP стає набагато краще (в порівнянні з іншими) і перевершує Java в цьому тесті. (Варто зазначити, що в PHP реалізація SHA-256 написана на C, і шлях виконання займає набагато більше часу в цьому циклі, оскільки зараз ми робимо 1000 хеш-ітерацій.).

Далі було проведено тест 5000 одночасних з'єднань (з N = 1) - або настільки близькими до цього, наскільки я міг би вийти. На жаль, для більшості цих середовищ рівень відмов не був незначним. Результати тесту можна побачити на рисунку (рисунок 2.7).

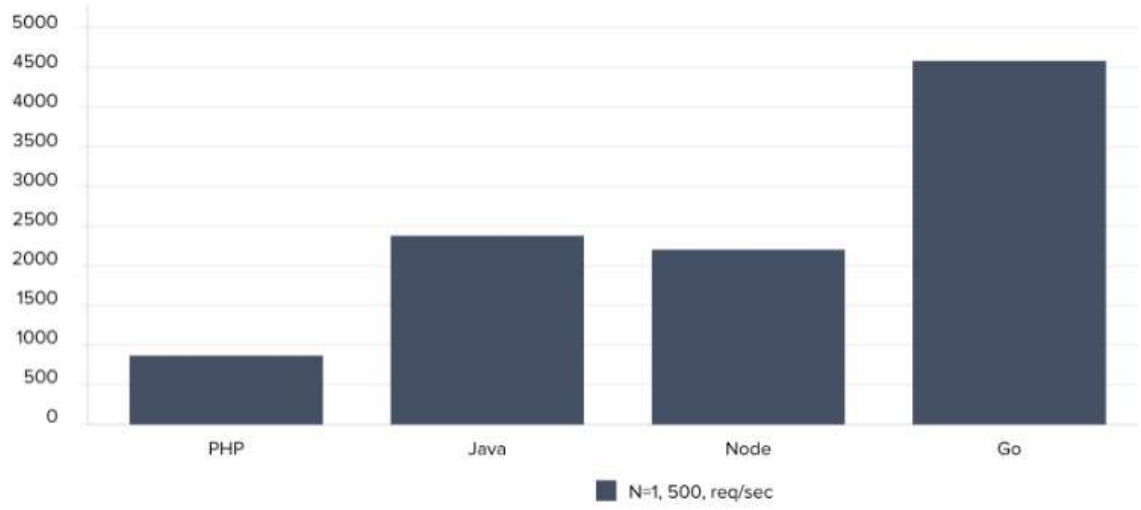

Рисунок 2.7 – Загальна кількість запитів в секунду

Це припущення, але схоже, що при великому обсязі підключень накладні витрати на з'єднання, пов'язані зі створенням нових процесів і пов'язана з ним додаткова пам'ять в PHP + Apache, стає домінуючим фактором і знижує продуктивність PHP. Очевидно, що Go є переможцем, за ним йдуть Java, Node і, нарешті, PHP.

Незважаючи на те, що факторів, які впливають на вашу загальну пропускну здатність, багато і сильно варіюються від програми до програми, чим більше ви розумієте суть того, що відбувається під капотом і пов'язані з цим компроміси, тим краще вам буде.

Заради справедливості, і PHP, і Java, незважаючи на описи, мають неблокуючі реалізації вводу-виводу, доступні для використання, але вони не так поширені, як описані вище підходи, і при використанні таких підходів потрібно враховувати супутні операційні витрати на утримання серверів. Не кажучи вже про те, що ваш код повинен бути структурований таким чином у спосіб роботи з такими середовищами; ваш "звичайний" веб-додаток PHP або Java зазвичай не буде працювати без істотних змін у такому середовищі.

Для порівняння (таблиця 2.3), якщо врахувати кілька важливих факторів, які впливають на продуктивність, а також простоту використання, ми отримаємо наступне:

Таблиця 2.3 – Порівняння мов програмування за обраними факторами

Мова	Потоки проти процесів	Неблокуючий ввід/вивід	Простота использования
PHP	Процеси	Ні	
Java	Потоки	Доступний	Потрібні зворотні виклики
Node.js	Потоки	Так	Потрібні зворотні виклики
Go	Потоки (Горутини)	Так	Зворотні виклики не потрібні

Потоки, як правило, набагато ефективніше використовують пам'ять, ніж процеси, оскільки вони використовують однаковий простір пам'яті, а процеси цього не роблять. Поєднуючи це з факторами, пов'язаними з неблокуючим вводом/виводос, ми бачимо, що, принаймні з факторами, розглянутими вище, коли ми рухаємося вниз по списку, загальна настройка, пов'язана з введенням-виводом, покращується

Однак на практиці вибір середовища для створення додатку тісно пов'язаний зі ззнаннями команди розробників середовища та загальною продуктивністю, якої можна досягти за допомогою нього. Тому для кожної команди може не мати сенсу просто занурюватися і починати розробку веб-додатків і сервісів на Node або Go. Дійсно, пошук розробників або знайомство з вашою внутрішньою командою часто називають основною причиною відмови від використання іншої мови та/або середовища.

2.4 Висновки до другого розділу

В ході аналізу гіпервізорів для віртуалізації було виявлено, що найкращими гіпервізорами є гіпервізори 1 типу, які встановлюються безпосередньо на апаратне устаткування, а отже не потребують витрат ресурсів серверу на обслуговування операційної системи. Після цього було проаналізовано актуальні гіпервізори 1 типу і було виявлено те, що Microsoft Hyper-V найбільше підходить для системи розглянутої в роботі, так як має невисоку вартість, порівняно з іншими гіпервізорами цього типу, а значить відповідає меті роботи, і в той же час має гарні показники віртуалізації, і не поступається продуктивність конкурентам в сфері віртуалізації.

Проаналізувавши найбільш популярні системи управління реляційними базами даних з відкритим початковим кодом, а саме PostgreSQL та MySQL було визначено що для розробки приватної хмарної системи паралельного збору та обробки інформації найбільше підходить PostgreSQL.

Після того як було обрано СКБД, було проаналізовано мови програмування веб серверів, і спираючись на такі чинники як продуктивність, можливість працювати з багатьма потоками чи навіть ядрами процесора, споживання невеликих об’ємів оперативної пом’яті, а також простоти мови. Зваживши всі переваги та недоліки було обрано Golang, для розробки веб серверу.

3 РОЗРОБКА ХМАРНОЇ СИСТЕМИ РОЗПОДІЛЕНОГО ОБЧИСЛЕННЯ

3.1 Структура хмарної системи розподіленого обчислення

Після проведення аналізу та проектування архітектури, була запропонована наступна структура системи (рисунок 3.1).

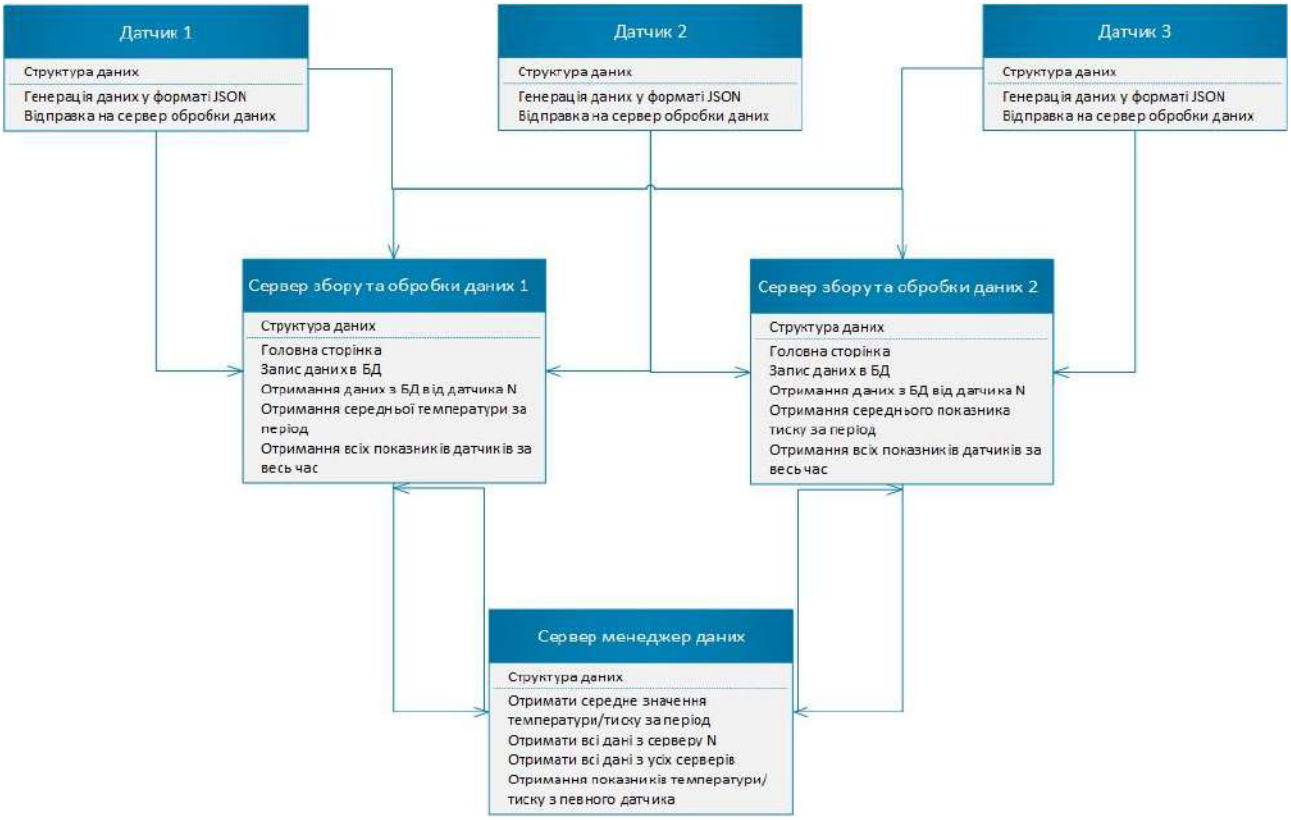

Рисунок 3.1 – Модель хмарної системи розподіленого збору та обробки інформації

Як видно на рисунку, принцип роботи системи такий: є n-на кількість датчиків (для демонстрації системи було обрано три датчики), в розробленій системі в ролі датчика будуть виступати віртуальні машини з наступними налаштуваннями:

- Windows 10;
- 3 Гб;
- Intel Core I5-3230m;

- 1 віртуальний процесор;
- Мережева карта: Microsoft Hyper-V Network Adapter.

Окрім трьох віруальних машин які емулюють потоки даних від температурних датчів є 3 віртуальні машини серверів збору та обробки даних з датчиків з наступними характеристиками:

- Windows 10;
- 4 Гб ОЗУ;
- Intel Xeon E-2224G;
- 1 віртуальний процесор;
- Мережева карта: Microsoft Hyper-V Network Adapter.

Алгоритмічна модель роботи датчиків виглядає наступним чином (рисунок 3.2)

Рисунок 3.2 – Алгоритмічна модель роботи датчиків температури

Алгоритмічна модель роботи серверу обробки даних виглядає наступним чином (рисунок 3.3)

Рисунок 3.3 – Алгоритмічна модель роботи серверу обробки даних

Алгоритмічна модель роботи серверу менеджеру даних виглядає наступним чином (рисунок 3.4)

Рисунок 3.4 – Алгоритмічна модель роботи серверу-менеджеру збору даних

3.2 Розробка серверної частини

Після розробки структури та написання алгоритму роботи кожного з компонентів системи було розпочато роботу з написання програмного забезпечення. Так як для написання ПЗ було обрано Golang, як мову програмування та PostgreSQL, як систему керування базами даних, наступним кроком треба було обрати програмну платформу (фреймворк) для розробки, але через те що мова досить нова в порівнянні з іншими представниками розробки веб-серверів в Golang, кількість таких платформ невелика, тому було прийнято рішення писати бекенд стандартними бібліотеками без використання фреймворків.

У ході розробки було використано такі бібліотеки, як:

- database/sql;
- encoding/json;
- fmt;
- github.com/lib/pq;
- log;
- net/http;

Database/sql – це бібліотека яка забезпечує загальний інтерфейс навколо баз даних SQL (або подібних до SQL). Пакет sql потрібно використовувати разом із драйвером бази даних.

Github.com/lib/pq – це чистий драйвер Go Postgres для бази даних.

Fmt – це пакет, який реалізує форматований ввід-вивід за допомогою функцій, аналогічних printf і scanf в мові C. Формат синтаксису походить від C, але є простішим.

Log – це пакет, який реалізує простий пакет логування. Він визначає тип, Logger, з методами для форматування виводу. Він також має попередньо визначений «стандартний» Логер, доступний через допоміжні функції.

Encoding/json – це пакет, який реалізує кодування та декодування JSON, як визначено в RFC 7159 [25]. Зіставлення між значеннями JSON і Go описано в документації для функцій Marshal і Unmarshal.

Net/http – це пакет, який забезпечує реалізацію клієнта та сервера HTTP.

Основою перевагою реалізації серверів в Golang, є те, що, наприклад, в пакеті net/http кожен запит на сервер за замовчуванням обробляється в окремій горутині, а отже надає дуже гарну продуктивність завдяки паралелізму потоків обробки даних, малому використанню пам’яті, так як на забезпечення роботи однієї горутини потрібно лише 4 Кб пам’яті, а також через те що горутини реалізовані таким чином, що навантаження на сервер іде рівномірне, а отже «простоїв» процесору не буде.

Для зберігання даних, які будуть надаватися датчиками, а також для збереження інформації про показники датчиків отриманих з бази даних було вико-

ристано структури мови Golang (лістинг 3.1).

Лістинг 3.1 – Структура даних показників датчика

```
type Indication struct {
SensorID   int    `json:"sensorid"`
Indication int    `json:"indication"`
Date       string `json:"date"`
}
```

Для емуляції потоків даних розглядалося два варіанти: генерація випадкових чисел та відправка на сервер або зчитування показників з файлу з подальшою їх відправкою на сервер. Зваживши всі переваги та недоліки кожного з способів, для забезпечення більшої правдоподібності було обрано другий спосіб.

Згідно алгоритму роботи та обраного варіанту реалізації програма (лістинг 3.2), яка відповідає за роботу датчика, спочатку зчитує дані з файлу, які містять показники температури за останні 5 років, далі формує пакет даних у форматі JSON і після цього відправляє їх на сервер обробки даних.

Лістинг 3.2 – Функція генерації показника датчика в форматі JSON

```
file, err := os.Open("data1.csv")
if err != nil {
     fmt.Println("Error opening CSV file:", err)
     return
}
defer file.Close()

reader := csv.NewReader(file)

var indications []Indication

for {
     row, err := reader.Read()
     if err == io.EOF {
          break
     }
     if err != nil {
          fmt.Println("Error reading CSV row:", err)
          return
     }
```

```
                indication := Indication{
                SensorID:    row[0],
                Date:        row[1],
                Temperature: row[2],
           }


           indications = append(indications, indication)
     }

     jsonData, err := json.Marshal(indications)
     if err != nil {
           fmt.Println("Error marshaling JSON:", err)
           return
     }

     url := "http://10.20.77.3:8080/indication/add"
     req,    err    :=    http.NewRequest(http.MethodPost,    url,
strings.NewReader(string(jsonData)))
     if err != nil {
           fmt.Println("Error creating HTTP request:", err)
           return
     }
     req.Header.Set("Content-Type", "application/json")

     client := http.DefaultClient
     resp, err := client.Do(req)
     if err != nil {
           fmt.Println("Error sending HTTP request:", err)
           return
     }
     defer resp.Body.Close()

           if resp.StatusCode != http.StatusOK {
           fmt.Println("Server     returned     non-OK     status:",
resp.Status)
           return
     }

     fmt.Println("Data successfully sent to the server.")}
```

Для забезпечення обробки запитів до серверу існує функція main (лістинг 3.3), яка містить «роутери», які запускають функцію відповідно до запиту, який надходить на сервер. Функція яка забезпечую цей зв’язок, а саме http.HandleFunc, має дуже гарну особливість, яка робить продуктивність серверу ще більшою. Ця особливість полягає в тому, що кожен запит, який надходить до серверу поміщається в окрему горутину, а це означає, що всі запити об-

роблюються паралельно одне одному.

Лістинг 3.3 – Головна функція серверу обробки даних

```
     func main() {
     http.HandleFunc("/", mainPageHandler)
     http.HandleFunc("/indication/add", addIndicationHandler)
     http.HandleFunc("/indications/sensor",
getIndicationBySensorHandler)
     http.HandleFunc("/average/date",
averageValueIndicatorsByDAteHandler)
     http.HandleFunc("/indication/all", getAllIndicationsHandler
     fmt.Println(http.ListenAndServe(":8080", nil))}
```

Для перевірки працездатності серверу було написано функцію для отримання головної сторінки (лістинг 3.4), після запиту до серверу буде отримано повідомлення про те що сервер працює

Лістинг 3.4 – Функція обробника запиту на отримання головної сторінки

```
     func mainPageHandler(w http.ResponseWriter, r *http.Request)
{
     w.Header().Set("Content-Type", "application/json")
     serverStatus := []byte("Server is working")
     w.Write(serverStatus)
     }
```

Після того, як дані з датчика було відправлено, сервер отримує запит на запис інформації в базу даних (лістинг3.5), після з’єднання з базою даних обробник запиту декодує JSON та записавши їх у структуру (лістинг 3.1) переходить до циклу в якому функція запису в базу даних спочатку отримує підготовлений запит, який призначений для захисту від SQL-ін’єкцій, а також дані з структури (лістинг 3.1), а вже після цього записує дані, і повертає відповідний статус, або у разі невдачі, повертає помилку.

Лістинг 3.5 – Функція обробника запиту на запис даних у форматі JSON отриманих від датчика в базу даних

```
     func     addIndicationHandler(w     http.ResponseWriter,     r
*http.Request) {
        db, err := sql.Open("postgres", "host=localhost port=5432
user=postgres password=1 dbname=weathersensor sslmode=disable")
```

```
        if err != nil {
          fmt.Println("Error connecting to the database:", err)
           return
   }
   defer db.Close()

        err = db.Ping()
        if err != nil {
           log.Fatal("Error pinging the database:", err)
        }

   if r.Method != http.MethodPost {
           w.WriteHeader(http.StatusMethodNotAllowed)
      return
   }

        var indications []Indication
        err = json.NewDecoder(r.Body).Decode(&indications)
        if err != nil {
           w.WriteHeader(http.StatusBadRequest)
      fmt.Fprintf(w, "Error decoding JSON: %v", err)
           return
   }

   stmt, err := db.Prepare(`INSERT INTO indications ("sensorID",
temperature, "date") VALUES ($1, $2, $3)`)
        if err != nil {
           fmt.Println("Error preparing statement:", err)
           return
   }
   defer stmt.Close()

   for _, indication := range indications {
           _,    err    =    stmt.Exec(stmt,    indication.SensorID,
indication.Indication,                              indication.Date)
      if err != nil {
              log.Fatal(err)
      }
   }

      w.WriteHeader(http.StatusCreated)
   fmt.Fprintln(w,   "Data   successfully   inserted   into   the
database.")
     }
```

Після того, як дані були записані в базу даних можна проводити деякі операції з ними. Саме для того, щоб можна було отримати всі показники з якогось конкретного датчика було написано функцію для отримання інформації за його ідентифікаційним номером (лістинг 3.6). Як і в функції запису в базу да-

них алгоритм роботи приблизно однаковий, і виглядає приблизно так, сервер отримує запит в якому міститься ідентифікатор сенсору, передає цю інформацію потрібному обробнику, який в свою чергу формує SQL запит, який разом з ідентифікатором передається в базу даних, яка в свою чергу передає дані серверу, а саме записує їх в структуру (лістинг 3.1) після чого з цих даних формує JSON і відправляє його або ж помилку у разі некоректної роботи.

Лістинг 3.6 – Функція обробника запиту на отримання всіх показників від якогось датчика.

```
     func getIndicationBySensorHandler(w http.ResponseWriter, r
*http.Request) {
     db, err := sql.Open("postgres", "host=localhost port=5432
user=postgres password=1 dbname=weathersensor sslmode=disable")
     if err != nil {
          fmt.Println("Error connecting to the database:", err)
          return}
     defer db.Close()
          err = db.Ping()
     if err != nil {
          fmt.Println("Error pinging the database:", err)
          return}
     if r.Method != http.MethodPost {
          w.WriteHeader(http.StatusMethodNotAllowed)
          return}
     stmt, err := db.Prepare(`SELECT * FROM indications WHERE
"sensorID" = $1 ORDER BY date`)
     if err != nil {
          fmt.Println("Error preparing statement:", err)
          return}
     defer stmt.Close()
     sensorID := r.URL.Query().Get("sensorid")
     rows, err := stmt.Query(sensorID)
     if err != nil {
          fmt.Println("Error querying data:", err)
          return}
     defer rows.Close()
          for rows.Next() {
          var indication Indication
          err := rows.Scan(&indication.SensorID,
&indication.Indication, &indication.Date)
          if err != nil {
          w.WriteHeader(http.StatusInternalServerError)
          fmt.Fprintf(w, "Error scanning row: %v", err)
          return}
          indications = append(indications, indication)}
     jsonData, err := json.Marshal(indications)
```

```
     if err != nil {
          w.WriteHeader(http.StatusInternalServerError)
          fmt.Fprintf(w, "Error marshaling JSON: %v", err)
          return}
     w.Header().Set("Content-Type", "application/json")
     w.WriteHeader(http.StatusOK)
     w.Write(jsonData)
     }
```

Однією із операцій, які можна проводити над показниками температури є пошук середнього її значення за якийсь період, цю функцію як одну з важливих було реалізовано і в дипломній роботі (лістинг 3.7). Алгоритм роботи схожий на функцію обробника запиту на отримання всіх показників від якогось датчика, але окрім передавання обробнику ідентифікатору датчика, в цей «роутер» запиту треба передати ще й дату початку і кінця періоду за який Ми хочемо отримати дані.

Лістинг 3.7 – Функція обробника запиту на отримання середньої температури за певний період часу

```
     func                    averageValueIndicatorsByDAteHandler(w
http.ResponseWriter, r *http.Request) {
     db,  err  :=  sql.Open("postgres",  "host=localhost  port=5432
user=postgres password=1 dbname=weathersensor sslmode=disable")
     if err != nil {
          fmt.Println("Error connecting to the database:", err)
          return}
     defer db.Close()
     err = db.Ping()
     if  err  !=  nil  {fmt.Println("Error  pinging  the  database:",
err)
          return}
     if r.Method != http.MethodGet {
          w.WriteHeader(http.StatusMethodNotAllowed)
          return}
     stmt, err := db.Prepare(`SELECT temperature FROM indications
WHERE "sensorID" = $1 AND date >= $2 AND date <= $3`)
     if err != nil {fmt.Println("Error preparing statement:", err)
          return}
     defer stmt.Close()
     sensorID := r.URL.Query().Get("sensorid")
     startDate := r.URL.Query().Get("start_date")
     endDate := r.URL.Query().Get("end_date")
     rows, err := stmt.Query(sensorID, startDate, endDate)
     if err != nil {fmt.Println("Error querying data:", err)
          return}
```

```
     defer rows.Close()
     var sum float64
     var count int
     for rows.Next() {
          var value float64
          err := rows.Scan(&value)
          if err != nil {fmt.Println("Error scanning row:", err)
          return}
          sum += value
          count++}
     var average float64
     if count > 0 {average = sum / float64(count)}
     data := Data{AvgValue: average,}
     jsonData, err := json.Marshal(data)
     if err != nil {
          fmt.Println("Error marshaling JSON:", err)
          return}
     sendAverage(w, r, jsonData)}
     func  sendAverage(w  http.ResponseWriter,  r  *http.Request,
jsonData []byte) {
     w.Header().Set("Content-Type", "application/json")
     w.WriteHeader(http.StatusOK)
     w.Write(jsonData)
     fmt.Println("JSON sent successfully.")
     }
```

Існує багато сервісів, які надають можливість працювати з тим чи іншим сервером за допомогою інтерфейсу прикладного програмування (англ. Application Programming Interface або скорочено API), наша система не є виключенням, саме через це було створено функцію (лістинг 3.8.), яка надає можливість отримати дані всіх показників за весь час.

Лістинг 3.8 – Функція обробника запиту на отримання всіх показників за весь час у форматі JSON

```
     func   getAllIndicationsHandler(w   http.ResponseWriter,   r
*http.Request) {
     db,  err  :=  sql.Open("postgres",  "host=localhost  port=5432
user=postgres password=1 dbname=weathersensor sslmode=disable")
     if err != nil {
          fmt.Println("Error connecting to the database:", err)
          return}
     defer db.Close()
     err = db.Ping()
     if err != nil {
          fmt.Println("Error pinging the database:", err)
          return}
```

```
    if r.Method != http.MethodGet {
        w.WriteHeader(http.StatusMethodNotAllowed)
        return}
    query := "SELECT * FROM indications "
    args := make([]interface{}, 0)
    rows, err := db.Query(query, args...)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        fmt.Fprintf(w, "Error querying data: %v", err)
        return}
    defer rows.Close()
    for rows.Next() {var indication Indication
        err          :=          rows.Scan(&indication.SensorID,
&indication.Indication, &indication.Date)
        if err != nil {
            w.WriteHeader(http.StatusInternalServerError)
            fmt.Fprintf(w, "Error scanning row: %v", err)
            return}
        indications = append(indications, indication)}
    jsonData, err := json.Marshal(indications)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        fmt.Fprintf(w, "Error marshaling JSON: %v", err)
        return}
    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusOK)
    w.Write(jsonData)}
```

3.3 Тестування системи

Для початку роботи системи треба спочатку запустити віртуальні машини, які емулюють датчики температури і запустити програму емуляції відправки даних на сервер. Результат роботи програми можна побачити на рисунку (рисунок 3.5).

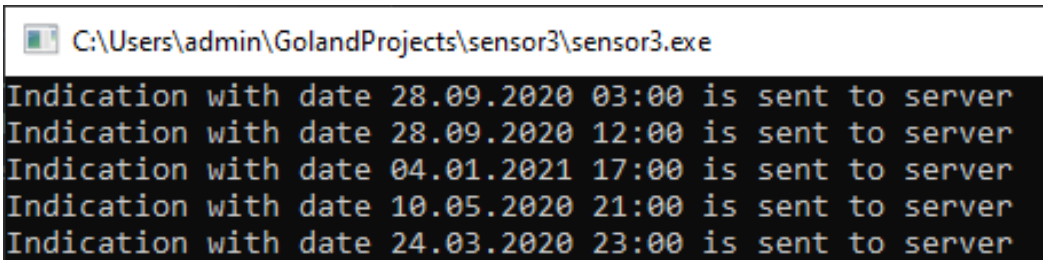

Рисунок 3.5 – Результат роботи програми емуляції відправлення датчиків на сервер

Щоб протестувати роботу серверу було вирішено використати програму

Postman для того, щоб облегшити формування запиту до серверу.

Спочатку спробуємо отримати від серверу головну сторінку (рисунок 3.6)

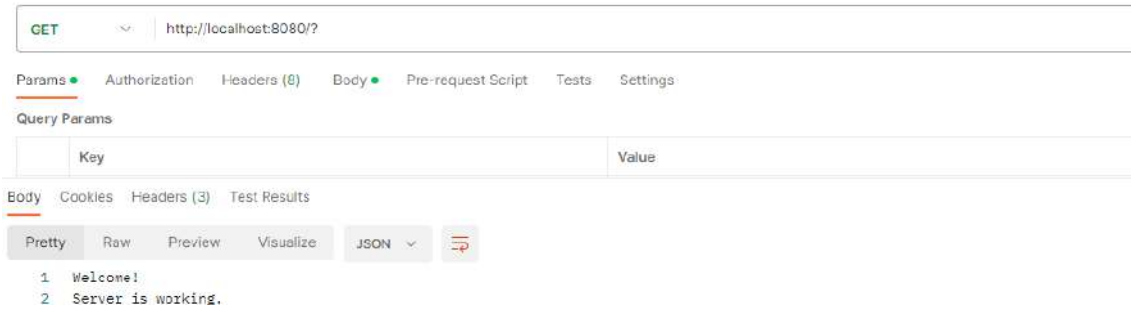

Рисунок 3.6 – Результат відповіді на запит про отримання головної сторінки

Після отримання головної сторінки зробимо запит на отримання всіх показань температури від усіх датчиків (рисунок 3.7)

Рисунок 3.7 – Результат відповіді на запит про отримання всіх показників від всіх датчиків

Далі було створення запиту на отримання середнього значення від певного датчика за певний період (рисунок 3.8)

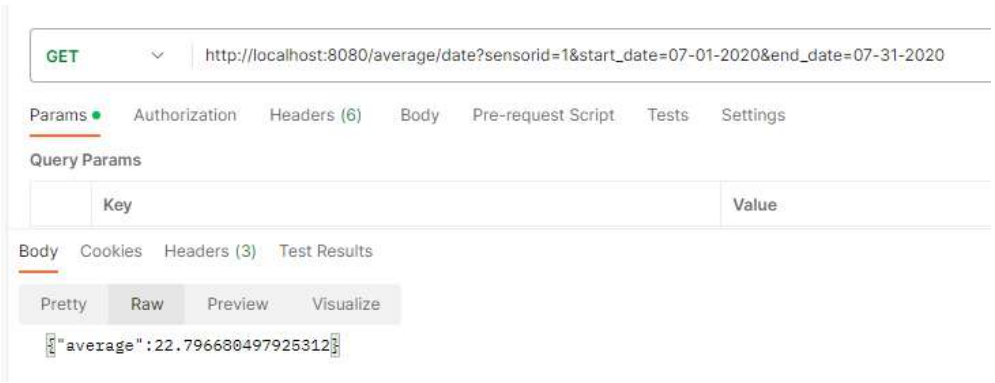

Рисунок 3.8 – Результат відповіді на запит про отримання середнього значення від певного датчика за певний період

ВИСНОВКИ

У даний час хмарні системи стали однією з найбільш важливих платформ для зберігання та обробки даних, що забезпечують розширення масштабів зберігання та обробки даних, зниження вартості інфраструктури, а також забезпечення гнучкості і ефективності використання ресурсів.

Також зараз все більше і більше систем, які збирають та оброблюють інформацію. Починаючи зі звичайних сайтів та комп'ютерних ігор, закінчуючи метеорологічними системами, інтернетом речей та навіть системи ситуаційної обізнаності.

В роботі було докладно проаналізована галузь хмарних обчислень, були розглянуті паралельні та розподілені обчислення. В другому розділі було розглянуто і порівняно інструменти, якими можна побудувати хмарні системи, а саме декілька гіпервізорів різних типів. Так, як наша система обов'язково має кудись зберігати дані, було розглянуто існуючі бази даних, а також проведено їх порівняння за певними показниками. І наостанок, для написання коду вебсерверу було розглянуто мови програмування для написання серверів, а саме було проаналізовано, які мови можна використати для подібних задач, а потім було порівняно за їх можливостями, яка з мов краще підходить для нашої системи.

Підсумовуючи все вищесказане, можна з впевненістю заявити, що хмарні системи розподіленого обчислення – це системи, які будуть ставати тільки популярнішими, а отже дослідження таких систем є дуже важливим напрямком.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach Nearly $600 Billion in 2023[Електронний ресурс] – https://www.gartner.com/en/newsroom/press-releases/2023-04-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023

2) Mell, P., Grance, T. (2011). The NIST definition of cloud computing. NIST special publication.

3) Всі там будемо: що таке хмарні сервіси і чому вони так стрімко розвиваються https://businessviews.com.ua/ru/tech/id/hmari-dlja-biznesu-2003/

4) Kovalenko, A., Liashenko, O., & Yaroshevych, R. (2021). Comparative analysis of the organization of cloud infrastructure. Advanced Information Systems, 5(2), 108–113.

5) Rizvi, Mohd Ahsan Kabir. (2016). Simulation of Parallel and Distributed Computing: A Review. IOSR Journal of Computer Engineering. 18. 5-11.

6) Ткачев В. Н. Метод предотвращения возникновения коллизий при параллельной обработке данных в децентрализованных вычислительных системах / В. Н. Ткачев, В. Е. Саваневич, А. Б. Анненков, А. Б. Брюховецкий // Науково-виробничий збірник "Наукові записки УНДІЗ". - № 2 (22). - 2012. - С. 110-116.

7) Zhang, Qi & Cheng, Lu & Boutaba, R.. (2010). Cloud Computing: State-of-the-art and Research Challenges. Journal of Internet Services and Applications.

8) VirtualBox vs. VMWare vs. Hyper-V: What's the Best Virtual Machine? [Електронний ресурс] – https://www.makeuseof.com/tag/virtualbox-vs-vmware-vs-hyper-v/

9) Офіційний сайт VMware – https://www.vmware.com/

10) Офіційний сайт Oracle VM VirtualBox [Електронний ресурс] – https://www.virtualbox.org/

11) Hyper-V Technology Overview [Електронний ресурс]

https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/hyper-v-technology-overview

12) Vagrant web development - is VMware better than VirtualBox? [Електронний ресурс] – https://www.jeffgeerling.com/blogs/jeff-geerling/vagrant-vmware-7-vs-virtualbox-5-benchmarks

13) Hyper-V vs VMware: Which One to Choose? [Електронний ресурс] – https://www.nakivo.com/blog/hyper-v-vmware-complete-comparison/

14) Офіційний сайт MongoDB [Електронний ресурс] – https://www.mongodb.com/

15) Офіційний сайт MySQL [Електронний ресурс] – https://www.mysql.com/

16) Офіційний сайт Oracle RDBMS [Електронний ресурс] – https://www.oracle.com/database

17) Офіційний сайт PostgreSQL [Електронний ресурс] – https://www.postgresql.org

18) Benchmark databases in Docker: MySQL, PostgreSQL, SQL Server [Електронний ресурс] – https://itnext.io/benchmark-databases-in-docker-mysql-postgresql-sql-server-7b129368eed7

19) New Benchmarks Show Postgres Dominating MongoDB in Varied Workloads [Електронний ресурс] – https://www.enterprisedb.com/news/new-benchmarks-show-postgres-dominating-mongodb-varied-workloads

20) Офіційний сайт Java[Електронний ресурс] – https://www.java.com/en/

21) Офіційний сайт PHP[Електронний ресурс] – https://www.php.net/

22) Офіційний сайт Node.js [Електронний ресурс] – https://nodejs.org/en

23) Офіційний сайт Golang [Електронний ресурс] – https://go.dev/

24) Server-side I/O Performance: Node vs. PHP vs. Java vs. Go [Електронний ресурс] – https://www.toptal.com/back-end/server-side-io-performance-node-php-java-go

25) The JavaScript Object Notation (JSON) Data Interchange [Електронний ресурс] – Formathttps://www.rfc-editor.org/info/rfc7159