Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра_____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти - другий (магістерський)

Дослідження математичних методів для вирішення задачі динамічного масштабування інфраструктури веб-застосунків в умовах обмежених ресурсів
(тема)

Виконав: студент 2 курсу, групи ІПЗм-18-2___
_____Шевченко Д.О._____
(прізвище, ініціали)

спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

Освітньо-наукової програми___
(тип програми)

Інженерія програмного забезпечення
(повна назва освітньої програми)

Керівник _____доц. Вечур В.О._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2020 р.

Харківський національний університет радіоелектроніки

Факультет_Комп'ютерних наук_____

Кафедра_Програмної інженерії_____

Рівень вищої освіти - другий (магістерський)

Спеціальність _121-Інженерія програмного забезпечення_____
(код і повна назва)

Тип програми освітньо-наукова програма_____

Освітня програма_Інженерія програмного забезпечення_____

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові _Шевченку Дмитру Олександровичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Дослідження математичних методів для вирішення задачі динамічного масштабування інфраструктури веб-застосунків в умовах обмежених ресурсів_____

затверджена наказом університету від "_27_" __03__2020 р № _473Ст_
заповнюється вручну після отримання наказу

2. Термін подання студентом роботи до екзаменаційної комісії
11 червня 2019 р.

3. Вихідні дані до роботи *електронні ресурси за обраною тематикою, вимоги до функціональності програми, методи математичного аналізу, платформа розробки Java, середовище розробки IntelliJ Idea*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, дослідження методів аналізу інфраструктури, дослідження типових інфраструктур, методи математичного аналізу, методи оптимізації інфраструктури*

5 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	доц. Вечур О.В.		11.05.20

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз проблемної області	27.01.20 – 10.02.20	виконано
2.	Постановка задачі	11.02.20 – 13.02.20	виконано
3.	Дослідження існуючих методів та моделей	14.02.20 – 19.02.20	виконано
4.	Розробка математичних моделей та алгоритмів	20.02.20 – 3.03.20	виконано
5.	Розробка бази даних та архітектури	4.03.20 – 7.03.20	виконано
6.	Програмна реалізація	8.03.20 – 25.03.20	виконано
7.	Експериментальне дослідження розроблених алгоритмів	26.03.20 – 30.03.20	виконано
8.	Підготовка пояснювальної записки	1.04.20 – 29.04.20	виконано
9.	Підготовка презентації та доповіді	30.04.20 – 3.05.20	виконано
10.	Попередній захист	7.05.20	виконано
11.	Нормоконтроль, рецензування	4.05.20 – 8.05.20	виконано
12.	Занесення диплома в електронний архів	8.05.20	виконано
13.	Допуск до захисту у зав. кафедри	8.05.20	виконано

Дата видачі завдання _____ _____2020 р.

Студент ________ _____Шевченко Д.О._____
(підпис)

Керівник роботи _____ _____доц. Вечур О.В._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить: 70 стор., 14 рис., 4 таблиць, 15 джерел.

JAVA, ОПТИМІЗАЦІЯ ІНФРАСТРУКТУРИ, АЛГОРИТМ, ОБМЕЖЕНІ РЕСУРСИ, МАСШТАБУВАННЯ, АПРОКСИМАЦІЯ, AWS, SERVERLESS.

Метою роботи є створення алгоритму, що буде використовуватися для розрахунку мінімальної конфігурації веб-застосунку за умови заданих ресурсних обмежень.

Засобами і методами розробки є IntelliJ IDEA, Java 8.

В результаті розробки отримана програма «Infrastructure calculator», яка може використовуватися користувачами для розрахунку вартості інфраструктури. Програма має надавати можливості для розрахунку вартості та необхідної інфраструктури.

The explanatory note for the certification of the master's degree includes 70 pages, 14 figures, 4 tables, 15 sources.

JAVA, INFRASTRUCTURE OPTIMIZATION, ALGORITHM, LIMITED RESOURCES, SCALING, APROXIMATION, AWS, SERVERLESS.

The purpose of the work is to create a infrastructure calculator.

The tools and methods of development are IntelliJ IDEA, Java 8.

The result is a “ Infrastructure calculator ” software that can be used by customers to calculate infrastructure price. The program should be able to calculate price and to calculate infrastructure needed for operational purpose.

ЗМІСТ

ВСТУП

Вартість програмного забезпечення складається з багатьох факторі: кількість витрачених чоловіко-годин, унікальність програмного забезпечення та інфраструктури необхідної для його роботи. Перший фактор базується на керівницьких методах, що використовуються на проекті. Другий фактор залежить від існуючого ринку. Третій фактор залежить як від системних вимог так і від нефункціональних вимог.

Витрати на інфраструктуру є постійною частиною витрат продукту. Можливість оптимізації інфраструктури з метод зменші витрат на її підтримку без втрати ефективності самого застосунку є важливою задачею у сучасному світі.

У проектних командах на цей час існує два підходи до використання інфраструктури.

Перший підхід використовує максимально можливу інфраструктуру, котру може використовувати за бюджетом. Недоліком такого метода є те, що потужності не використовуються на повну, що зменшує корисну ефективність грошей на неї витраченої, що веде до того, що бюджет просто напросто передається власникам облачних сервісів, або витрачається на оплату струму для своїх датацентрів.

Другий підхід навпаки використовує мінімальну інфраструктуру та збільшує її потужності за необхіднюстю. Цей підхід більш нагадує раціональний, але призводить до того, що змінення інфраструктури завжди витрачає час спеціалістів. Таким чином бюджет проекту витрачається не на корисну роботу, а на постійну зміну ресурсів.

Метод цієї роботи є дослідження існуючих математичних методів для вирішення задачі динамічного масштабування інфраструктури програмного застосунку та розробка практичного алгоритма для аналізу інфраструктур та подальшої їх оптимізації.

Також для розробки алгоритму оптимізації інфраструктуру необхідно було дослідити основні варіанти конфігурації цієї інфраструктури, що

використовуються у роботі спеціалістами, а також дослідити методи, що використовуються для аналізу існуючої інфраструктури.

Розроблений алгоритм необхідно було експериментально дослідити задля доказу його ефективності на реальних проектах та розібрати можливі шляхи його покращення.

В ході атестаційної роботи магістра було:

– проведено аналіз типових компонентів інфраструктури програмного додатку;

– проведено порівняльний аналіз методів для аналізу існуючої інфраструктури;

– розроблено алгоритми вирішення задачі;

– виконано програмну реалізацію алгоритму;

– була проведена експериментальна перевірка розробленого алгоритму;

– за результатами експериментальної перевірки було розроблено рекомендації щодо використання розробленого алгоритму та знайдені шляхи для подальших дослідження задля покращення алгоритму.

За результатами атестаційної роботи магістра було розроблено презентацію (див. додаток А). Лістинг програмної реалізації наведено в додатку Б.

Була написана та подана до опублікування до наукового журналу «East European Science Journal» стаття «Evaluation and optimization of software product infrastructure», що наведена у додатку В. Усі матеріали наведено на диску (див. додаток Г).

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

Швидкість наукового прогресу з кожним роком збільшується – збільшується і швидкість змінення інформаційних систем. Але відомо і те, що існуючу та реально використовувану систему важко, довго, дорого та часто нездійсненно змінити. В результаті інформаційні системи часто стають гальмом для розвитку підприємства. Основним джерелом проблем майже завжди стає трансформація та зміни у структурі. Сучасны технології настільки розвинуті, що майже неможливо знайти інформаційну систему, яку не можна створити. Однак, з огляду на вимоги бізнесу, засновані на показниках різних бізнес-оцінок, виникають додаткові складнощі, вирішення яких зводиться до забезпечення ефективного підходу до процесу проектування, реалізації й подальшого використання інформаційних систем. Зважаючи на це, можливо однозначно вважати, що обране архітектурне рішення є одним з найважливіших показників ефективності створюваної інформаційної системи, а, отже, і успішності бізнесу.

Архітектура підприємства являє собою процес збору та поширення інформації про те, як організація використовує і повинна використовувати інформаційні технології в своїй діяльності. За своєю ідеєю, архітектура інформаційних систем компанії відображає корпоративну структури компанії. Безпосередньо архітектура підприємства не описує конкретні технічні рішення окремих інформаційних систем, але дозволяє отримати істотні переваги для бізнесу та організації в цілому, що пов'язано з підвищенням ступеня використання та ефективності інформаційних систем та програмних продуктів, стандартизацією і перевикористанням технологічних ресурсів, а також зниженням ризиків інвестицій в IT-сфері.

Визначити поняття "архітектура інформаційної системи" можна безліччю способів.

Загалом архітектура інформаційної системи це концепцію, що визначає модель, структуру, виконувані функції й взаємозв’язок компонентів інформаційної системи. На рисунку 1.1 можна побачити базову архітектуру веб-застосунку [1].

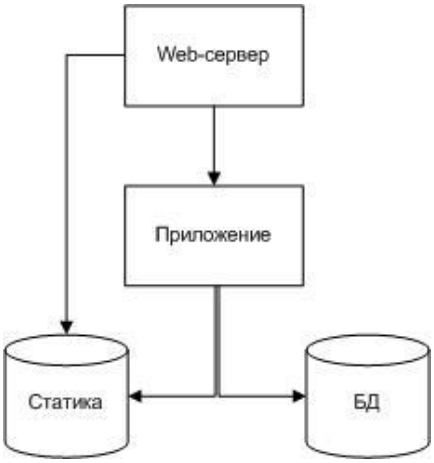

Рисунок 1.1 – Базова архітектура веб-застосунку

Концепція архітектури інформаційної системи повинна формуватися ще на етапі техніко-економічного обґрунтування (визначенні навантаження) й вибиратися такою, щоб мінімізувати вартість її використання.

Для того щоб конструктивно визначити архітектуру, необхідно відповісти на ряд запитань:

– для чого необхідна система;
– з яких складових вона зібрана;
– як вони взаємодіють;
– як розміщена інфраструктура.

Таким чином, можна вважати архітектуру інформаційної системи моделлю, що визначає вартість використання через наявну в даній системі інфраструктуру.

Чому питання формування IT-архітектури стають особливо актуальними саме зараз? Пояснення може грунтуватися на цілій сукупності факторів, основні з яких пов'язані з:

– змінами в самому бізнесі і індустріальному суспільстві;

– зміною ролі інформаційних технологій в бізнесі і суспільстві;

– зміною корпоративної культури та стилю управління в бізнесі;

– а також з об'єктивним збільшенням IT-бюджетів компаній.

До числа характерних змін бізнесу, які мають істотний вплив на використання інформаційних технологій, відносяться перш за все:

– глобалізація бізнесу, пов'язана з необхідністю об'єднання різних національних процесів, даних і персоналу;

– динаміка злиттів і поглинань, що призводить до об'єктивно необхідної інтеграції різних інформаційних систем, об'єднання IT-служб і, що є найбільш складним, інтеграції різних корпоративних культур;

– поява адаптивного стилю бізнесу - перехід від моделі, заснованої на наявній лінійці продуктів (т.зв. "make-and-sell"), до моделі, заснованої на гнучкому реагуванні на потреби ринку - ("sense-and-respond"). Цей стиль пов'язаний з визнанням неминучості і непередбачуваності змін у зовнішньому середовищі. Компанії, які взяли таку модель, пов'язують досягнення успіху зі здійсненням таких перетворень в бізнес-процесах і організаційній структурі, які могли б оперативно і адекватно підлаштовуватися під зміни, що відбуваються;

– скорочення характерних тривалостей бізнес-процесів і віртуалізація бізнесу.

Методика є інструментом для створення широкого спектра різних архітектур. Методика, зазвичай, включає в себе опис методів проектування архітектури та опис того, як різні ланки проекту пов'язані між собою, набір інструментів для опису елементів архітектури та загальний словник використовуваних термінів.

Методики також можуть містити список рекомендованих стандартів і сумісних продуктів, які можуть використовуватися для реалізації елементів архітектури. Важливо розуміти, що методики не тільки задають набір документів і планів, необхідних для опису підприємства, а й визначають, як всі ці елементи опису пов'язані між собою. Методики описують, як визначаються і документуються основні елементи архітектури підприємства.

Вони дозволяють вирішити проблему поганої коммунікації між залученими в цей процес людьми, оскільки задають якийсь загальний та однаково зрозумілий набір понять і моделей для опису елементів архітектури в інтересах різних категорій зацікавлених сторін. Розробка одних методик була ініційована державними структурами, а інших - приватними компаніями та представниками індустрії. Різні методики, як правило, орієнтовані на різні аудиторії потенційних користувачів і відрізняються широтою охоплення проблеми, увагою до певних галузей, хоча тенденція полягає в поступовій уніфікації визначень, пов'язаних з архітектурою. Методики або концентруються на певних секторах індустрії, або більш вдаються у чітке документування, або взагалі приділяють більшу увагу процесу переходу від сьогоднішнього в майбутній стан архітектури.

Існують індустріальні стандарти на опис архітектури підприємства, прийняті такими організаціями, як Інститут інженерів електрики і електроніки (IEEE - Institute of Electrical and Electronics Engineers), міжнародна організація стандартизації (ISO - International Organization for Standardization), The Open Group і т.д. Але жоден з цих стандартів не займає домінуючого положення та використовується в залежності від бажання архітектора проекту.

Більш того, жоден з них, взятий окремо, не дає групам, відповідальним за розробку архітектури, всіх інструментів, необхідних з методичної точки зору і з точки зору шаблонів, які використовуються для опису архітектури. Однак цей накопичений арсенал методик і стандартів надає архітекторам широкі можливості вибору архітектурних моделей, прикладів і досвіду різних індустрій.

При цьому треба чітко розуміти, по-перше, відмінність методики опису архітектури від самої архітектури як такої, а по-друге те, що одна й та сама методика може призводити до створення абсолютно двох несхожих між собою архітектур підприємства через відмінності в бізнесі і області діяльності організації, наявності певного набору успадкованих систем і т.д.

Важливим для розуміння методик є використовувані в них моделі, різні уявлення (view) або доменів архітектури.

Опис IT-архітектури служить детальним керівництвом, яке визначає основні, стандартні або типові елементи IT-систем, їх взаємозв'язку, а також процеси управління інформаційними системами. Що хотілося б отримати від такого документа? Можна сформулювати такі, частково суперечливі, вимоги:

– висока деталізації для практичного використання фахівцями в області інформаційних технологій при розробці нових систем;

– простоту для розуміння бізнес стороною проекту;

– динаміку розгляду (тобто "Архітектура як є" - "Короткострокові та середньострокові завдання" - "Стратегічні плани");

– можливість адаптації за новими вимогами бізнесу та врахування можливостей реалізації незапланованих проектів.

Таким чином з ростом важливості формального опису IT-архітектури зростає і важливість розуміння чи можлива реалізація архітектури заданого проекту при відомому кошторисі. Дана проблема частіше вирішується за рахунок завищення вартості проекту або понаднормової праці робітників IT-сфери.

На рисунку 1.2 можна побачити тенденцію зросту зарплати IT-фахівців, що без зміни розміру команди свідчить про збільшення середнього кошторису проекту. Таким чином важливо зменшити кошторис проекту за рахунок оптимізації конфігурації інфраструктури, що є також і необхідною частиною для розуміння процесу розробки. Важливо відмітити, що оптимізація інфраструктури може також зменшити потреби у operation частині команди, що також зменшить кошторис проекту.

При дослідженні методів аналізу інфраструктури було визначено декілька підходів.

Перший підхід аналізує інфраструктуру відно коду програмного застосунку: експерт аналізує функціональні вимоги програмного застосунку, існуючую програмну реалізацію та засновуючись на цих речах, а також своєму експертному баченні висуває вердикт щодо необхідної інфраструктури у разі першого деплою застосунку або оптимізації у разі аналіза уже існуючого застосунку.

Недоліками данного метода є суб’єктивність досвіду експерта, неоднорідність опису функціональних та нефункціональних вимог та недоліки програмної реалізації.

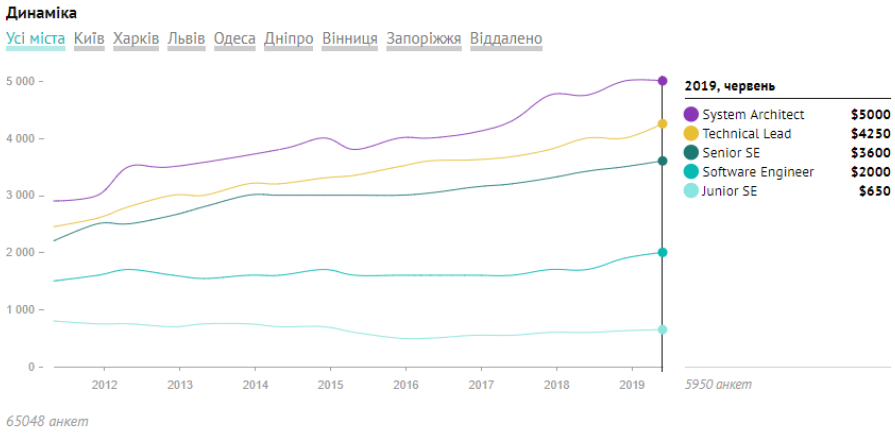

Рисунок 1.2 – Графік заробітньої платні фахівця

Неоднорідність вимог призводить до складності їх розуміння, та також потребує наявності будь-якої проектної документації, що на практиці не завжди відображає реальну картину: на практиці реалізації функціональних вимог випереджає написання технічної документації, а в зв’язку з використанням Agile-методологій призводить до того, що в гонитьбі за розробкою нової функціональності для клієнта часто необхідно жертвувати зручністю підтримки застосунку.

Недоліки програмної реалізації проекта призводять до того, що при однакових вимогах можливе перевищення необхідного числа кількості комунікації між частинами застосунку: часто в застосунках існують проблему з комунікацією сервера застосунку з сервером баз даних через неоптимізовану кількість запитів або їх структуру. Такі проблеми у кодовій базі додатку призводять до неочікуваних просадок у ефективності, що призводить до збільшення вимог до інфраструктури.

Суб’єктивність експерта можу запросто призвести до того, що замовник підтримує інфраструктуру більш потужну лише через те, що експерту вирішив збільшити відмовостійкісти додатку більше, ніж це необхідно. Це також пов’язано

з тим, що часто компанії у договорах акцентують увагу на фінансовій стороні, а не інфраструктурній.

За основу другого підходу береться робота з програмним продуктом як з «чорним ящиком»: людина, що аналізує програмний продукт, не знає нічого про те, як він працює всередині, а аналізє лише його вхідні та вихідні дані. Одним з таких методів є тестування навантаження.

Тестування навантаження складається з виконання ряду функціональних тестів відносно вже існуючого додатку та аналізує такі складові як час відповіді та коректність обробки запиту. Також під час проведення тестування навантаження слідкуючи за станом додатку можна швидко виявити не лише те, що при збільшені кількості запитів зростає час відповіді, але й те з якого моменту запити до бази даних перестають оброблятися або те, що застосунок спіймав deadlock.

Серед переваг цього метода для аналізу інфраструктури можна виділити його універсальність: за наявность функціональних вимог до програмного застосунки можливо провести тестування навантаження та знайти слабкі точки роботи застосунку.

Недоліком цього метода є той факт, що тестування може показати проблеми з інфраструктурою, а не конкретні проблеми з реалізацією функціональних вимог. Варто відмітити, що під час мониторингу також необхідно знайти досвідченого експерта, котрий зможе знайти проблеми у коді, які призводять до функціональних недоліків.

Облачні сервіси вже мають сервіси, що допомагають оптимізовувати інфраструктуру існуючих додатків. Наприклад Azure Advisor аналізує існуючу конфігурацію та телеметрію додатка та, засновуючись на цих даних, надає пропозиції щодо оптимізації інфраструктури. Такий підхід дозволяє зробити висновки щодо ефективності використання існуючою інфраструктури, але ігнорує функціональні та нефункціональні вимоги. AWS Trusted Advisor використовує аналогічні підходи задля аналізу інфраструктури. Також обидва облачні сервіси надають підтримку від technical account manager задля детальної оптимізації та економії.

1.2 Постановка задачі

Масштабованість – здатність пристрою збільшувати свої можливості шляхом нарощування числа функціональних блоків, виконуючих однакові завдання. Іншими словами, у випадку, коли віртуальний сервер не справляється з навантаженням, йому додаватимуться додаткові об’єми потужностей у вигляді додаткових ядер процесора, контейнерів, віртуальних машин, мегабайт оперативної пам’яті, тощо або піднімуть ще один віртуальний сервер для паралелізації навантаження [2]. Автоматичне масштабування виключає з цього процесу участь людини, що значно заощаджує час, ресурси і як наслідок – кошти [3]. Метою роботи є розробка алгоритму автоматичного масштабування ресурсів.

2 РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ

2.1. Аналіз типової інфраструктури

Переходячи до масштабування системи, перш за все варто визначити, який з шарів є «вузьким місцем» - тобто працює повільніше за решту системи. Для початку можна скористатися банальними утилітами типу top (htop) для оцінки споживання процесора та пам'яті і df, iostat для оцінки споживання диска. Однак, бажано виділити окремий хост, з емуляцією бойового навантаження (за допомогою AB або JMeter), на якому можна буде профілювати роботу програми за допомогою таких утиліт як xdebug, oprofile і так далі. Для виявлення вузьких запитів до БД можна скористатися утилітами типу pgFouine (зрозуміло, що робити це краще на основі балок з бойового сервера).

Зазвичай «вузьке місце» залежить від архітектури додатку, але найбільш ймовірними кандидатами на «вузьке місце» в загальному випадку є база даних і код. Якщо ваш додаток працює з великим об'ємом призначених для користувача даних, то «вузьким місцем», відповідно, швидше за все буде зберігання статики.

Як вже говорилося раніше, найчастіше вузьким місцем в сучасних додатках є база даних. Проблеми з нею поділяються, як правило, на два класи: продуктивність і необхідність зберігання великої кількості даних.

Знизити навантаження на БД можна поділивши її на декілька хостів. При цьому гостро встає проблема синхронізації даних між ними, вирішити яку можна шляхом реалізації схеми master/slave з синхронною або асинхронною репликацією. У випадку з PostgreSQL реалізувати синхронну реплікацію можна за допомогою Slony-I, асинхронну - PgPool-II або WAL(9.0). Ілюстрацію механізму можна побачити на рисунку 2.1. Вирішити проблему поділу запитів читання і запису, а так само балансування навантаження між наявними slave'ами, можна за допомогою налаштування спеціального прошарку доступу до БД (PgPool-II).

Проблему зберігання великого обсягу даних в разі використання реляційних баз даних можна вирішити за допомогою механізму партіціювання ("partitioning" в

PostgreSQL), або розгортаючи БД на розподілених файлових системах типу Hadoop DFS. Ілюстрацію цього механізму можна побачити на рисунку 2.2.

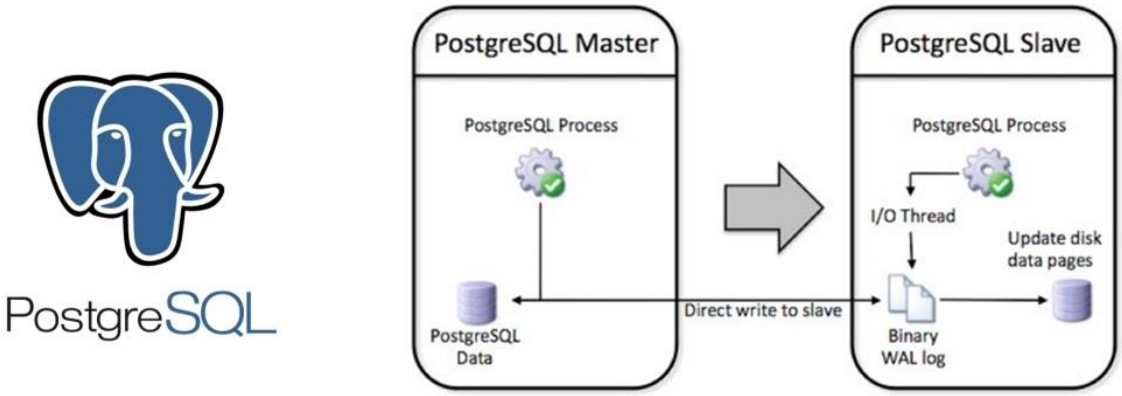

Рисунок 2.1 – Реплікація у PostgreSQL

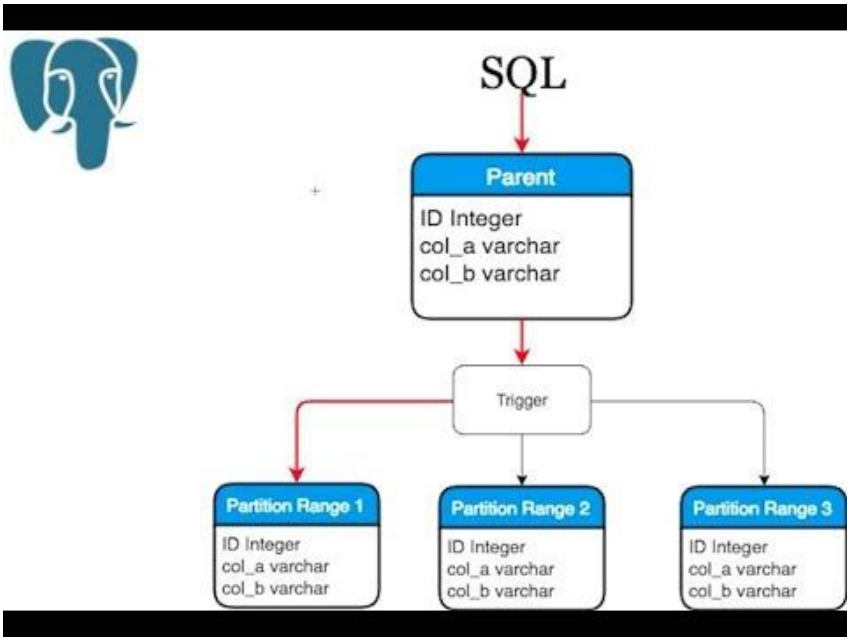

Рисунок 2.2 – Партиціонування у PostgreSQL

Однак, для зберігання великих обсягів даних найкращим рішенням буде «шардування» (sharding) даних, механізм якого є вбудованою перевагою більшості

NoSQL баз даних (наприклад, MongoDB). Ілюстрацію можна побачити на рисунку 2.3.

Крім того, NoSQL бази даних в загальному працюють швидше своїх SQL-братів за рахунок відсутності overhead'a на розбір/оптимізацію запитів та перевірки цілісності структури даних і т.д. Тема порівняння реляційних і NoSQL БД так само досить обширна і заслуговує на окрему статтю.

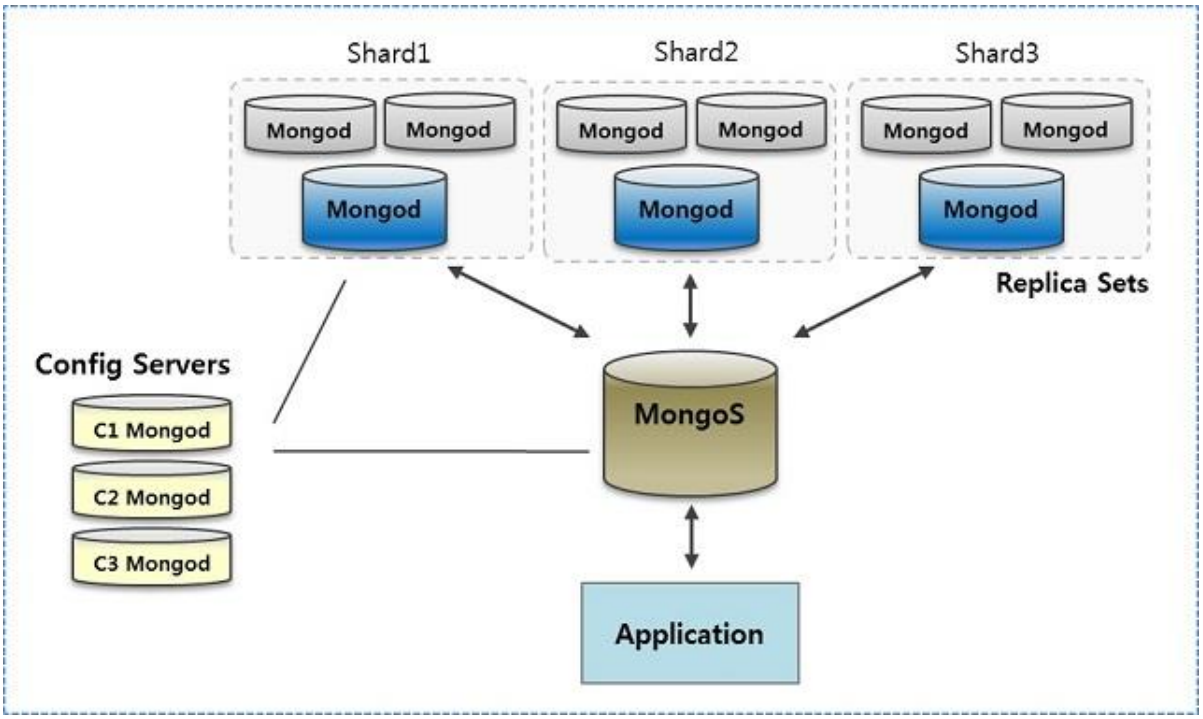

Рисунок 2.3 – Шардінг на прикладі Mongo

Окремо варто відзначити досвід Facebook, який використовують MySQL без JOIN-вибірок за рахунок денормалізації бази даних. Така стратегія дозволяє їм значно легше масштабувати БД, переносячи при цьому навантаження з БД на код, який, як буде описано нижче, масштабується простіше БД.

Складнощі з масштабуванням коду залежать від того, скільки поділюваних ресурсів необхідно хостам для роботи вашого додатка. Чи будуть це тільки сесії, або буде необхідне використання загального кешу та файлів?

У будь-якому випадку в першу чергу необхідно запустити репліки програми на декількох інстансах з однаковим оточенням.

Наступним кроком потрібно зробити так, що б файли статики, кеш та сесії web-додатку були доступні на кожному з хостів. Для сесій можна використовувати сервер, який працює через мережу (наприклад, memcached). Як сервер кеша цілком розумно використовувати той самий memcached, але, природно, на іншому хості.

Файли статики можна змонтувати з якогось загального файлового сховища по NFS / CIFS, використовувати розподілену ФС (HDFS, GlusterFS, Ceph), або використовувати хмарне сховище (S3, Google Cloud Storage).

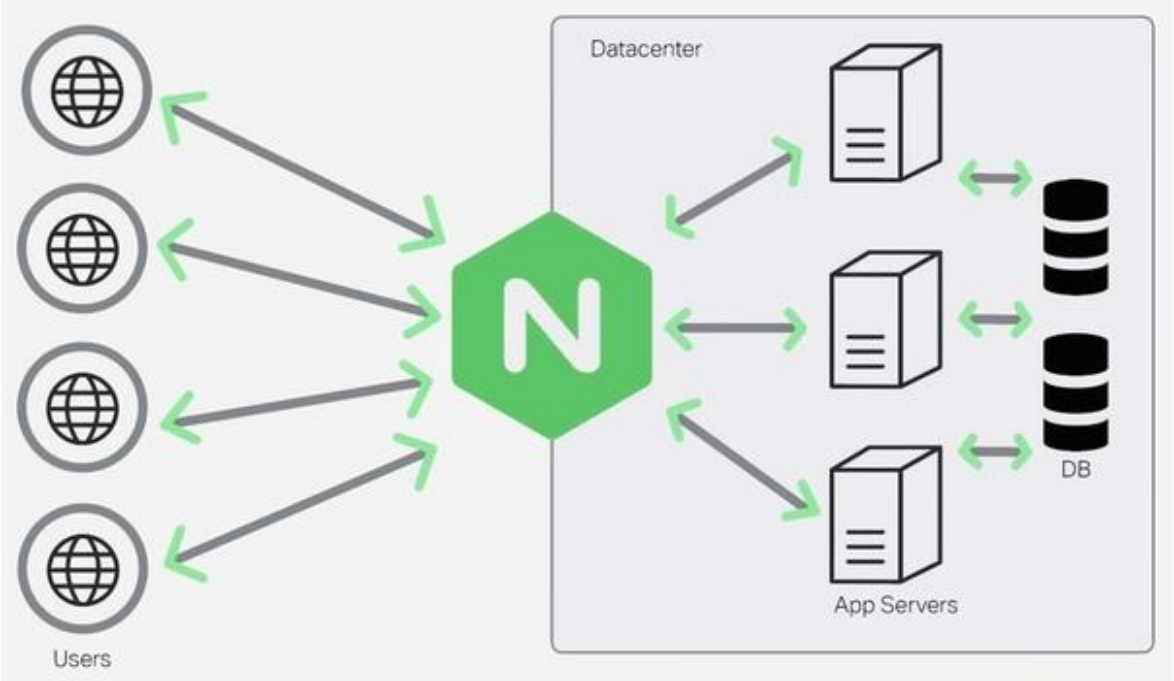

Рисунок 2.4 – Використання nginx для балансування навантаження

Так само можна зберігати файли в базі даних у бінарному вигляді (наприклад, Mongo GridFS), вирішуючи тим самим проблеми доступності і масштабованості (з урахуванням того, що для NoSQL БД проблема масштабованості вирішена за рахунок шардінга).

Окремо варто відзначити проблему деплоймента на декілька хостів. Як зробити так, щоб споживач, натискаючи «Оновити», не бачив різні версії додатка? Найпростішим рішенням, на мій погляд, буде виключення з конфіга балансувальника навантаження (web-сервера) не оновлених хостів, та послідовного

їх включення по мірі відновлення. Так само можна прив'язати користувачів до конкретних відслідковувати переходи по посиланню cookie або IP. Якщо ж оновлення вимагає значних змін в БД, найпростіше, взагалі тимчасово закрити проект.

При необхідності зберігання великого обсягу статики можна виділити дві проблеми: брак місця і швидкість доступу до даних. Як вже було сказано раніше, проблему з нестачею місця можна вирішити як мінімум чотирма шляхами: розподілена файлова система, зберігання даних в БД з підтримкою шардінга і організація шардінга «вручну» на рівні коду та хмарні сховища.

Наприклад можливо використовувати хмарні файлові сховища, як Amazon S3 або Google Cloud Storage.

Amazon Simple Storage Service (Amazon S3) - це сервіс зберігання об'єктів. Це означає, що клієнтами можуть бути компанії будь-яких розмірів і з будь-яких областей діяльності. Вони можуть використовувати цей сервіс для зберігання і захисту будь-яких обсягів даних в різних ситуаціях, наприклад для забезпечення роботи сайтів, мобільних додатків, для резервного копіювання та відновлення, архівації, корпоративних додатків, пристроїв IoT і аналізу великих даних. Amazon S3 пропонує прості у використанні інструменти адміністрування, які дозволяють організувати дані і точно налаштувати обмеження доступу відповідно потребами вашого бізнесу або законодавчими вимогами. Amazon S3 забезпечує надійність +99,999999999% (тут 11 дев'яток) і зберігає дані мільйонів додатків в інтересах компаній з усього світу.

Google Cloud Storage забезпечує високоміцне зберігання об’єктів у всьому світі, яке збільшує обсяг даних. Ви можете отримати доступ до даних миттєво з будь-якого класу зберігання, інтегрувати сховище у свої програми за допомогою єдиного уніфікованого API та легко оптимізувати ціну та ефективність.

При цьому варто розуміти, що видача статичних даних теж не найпростіше завдання, коли мова йде про високі навантаженнях. Тому дуже важливо мати безліч серверів призначених для видачі статичних даних. При цьому, якщо ми маємо загальне сховище даних (розподілена файлова система або база даних), при

збереженні файлу ми можемо зберігати його ім'я без урахування хоста, а ім'я хоста підставляти випадковим чином при формуванні сторінки (випадковим чином балансуючи навантаження між серверами, що роздають статику) . У разі, коли шардінг реалізується вручну (тобто, за вибір хоста, на який будуть залиті дані, відповідає логіка в коді), інформація про хости заливки повинна або обчислюватися на основі самого файлу, або генеруватися на підставі інших даних (інформація про користувача, кількості місця на дисках-сховищах) і зберігатися разом з ім'ям файлу в БД.

Таким чином у нас є безліч факторів для того, щоб проводити оцінку системи у реальному часі, але на момент розробки єдиним фактичним показником, котрий можливо брати до уваги є трафік, котрий і вирішено було використовувати для статичного аналізу ресурсної вартості системи.

2.2. Експериментальне порівняння типових інфраструктур

Було вирішено провести експеримент задля аналізу двох найбільш типових інсталяцій.

Для аналізу інфраструктури тестуванням навантаження було обрано два застосунки: розгорнуту на віртуальних машинах та застосунок на хмарних функціях. Під час підготовки до експерименту було реалізовано два застосунки за одним списком функціональних вимог. Це було пов’язано з тим, що варіанти розгорнутої інфраструктури невідривно пов’язані з програмною реалізацією. Таким чином було отримано java-застосунок з базою даних MongoDB, що розгортався на віртуальній машині та набор функцій AWS Lambda написаний мовою python з використання бази даних DynamoDB.

Перший застосунок було встановлено на двух віртуальних машинах t2.mini у AWS хмарі. Другий застосунок теж було розгорнуто в AWS. На цьому етапі

важливо окремо зупинитись та розглянути різницю між підходами до розгортання застосунку.

Розгортання застосунку на віртуальних машинах має в своїй основі підтримки деякої кількості віртуальних серверів з копіями коди застосунку, розгорнутою базою та її репліками. Такий підхід є стандартом у світі програмних продуктів.

Переваги цього метода складаються у том, що цей підхід вивірений часом та більшість спеціластів вміють з ним працювати. Недоліками ж такого підходу є складність масштабування. Для масштабування таких застосунків існує два підходи: горизонтальне та вертикальне масштабування. Вертикальне масштабування збільшує кількість ресурсів за рахунок збільшення ресурсів на одній віртуальній машині. Горизонтальне масштабування ж за свою основу бере балансування навантаження та збільшення ресурсів за рахунок збільшення кількості віртуальних машин. Вертикальне масштабування має свій ліміт, з боку хмарних сервісів, а горизонтальне масштабування потребує детальної розробки архітектури, відсутність якої на ранньому етапі роботи може призвести до ускладнення рефакторингу та подальшої роботи додатку.

Використання хмарних функцій за свою основу бере простоту масштабування. Для виконання кожного запиту виконується виклик «функції» - незалежного фрагмента коду, котрий виконує лише необхідну задачу. Розгортається контейнер у якому виконується код. На кожний запит створюється новий контейнер або використовується вільний, вже виконавший функцію для обробки іншого запиту.

Це призводить нас до того, що ми не лише виконуємо код, але й чекаємо створення контейнера, що додає час на обробку незалежного запита. Цей час назвивається часов холодного старта. Час холодного старта залежить від розміру функції та її мови [4].

Під час підготовки до наступної фази досліду було проведено експеримент для з’ясування цього часу. Результати можно побачити у таблиці 2.1. Виявилося, що для java час холодного старту в середньому складає 500 мілісекунд, а для python

– 200 мілісекунд. Що свідчить про деякі недоліки serverless підходу перед типовою інсталяцією на віртуальних машинах, оскільки при розгортанні звичайного сервера таких проблем не спостерігається.

Таблиця 2.1 – Час холодного старту хмарної функції

Виділена RAM / Мова	128 Mb	1024 Mb	3008 Mb
Java	650 ms	567 ms	480 ms
Python	253 ms	223 ms	189 ms
Node JS	199 ms	212 ms	210 ms
Golang	350 ms	320 ms	300 ms

Таким чином розібравши основну різницю між методами розгортання застосунку зберемо дані. Задля збору даних було проведено ряд тестів, що тестували навантаження, кількість запитів в секунду поступово збільшувалася від 100 до 1500. Результат можна побачити на рисунку 2.5 та в таблиці 2.2.

Під час аналізу отриманих даних можна помітити, що зі зростанням кількості запитів час відповіді хмарних функцій майже не змінювався та тримався біля 250 мілісекунд, що відповідає отриманим раньше даним про 200 мілісекунд холодного старта та часу виконання самої функції. Регулюючи кількість одночасно існуючих контейнерів можна знати оптимальную комбінацію ефективності.

Ефективністи ж інсталяції на віртуальній машині бажає кращого. Проаналізувавші дані було зроблено висновок, що проблема ефективності було пов'язана з тим, що Tomcat може обробляти близько 350 транзакцій в секунду [5].

Проаналізувавши отримані в ході тестування навантаження результати було спроектовано метод для оптимізації інфраструктури розгорнутої на віртуальни машинах. Суть метода в тому, що навантаження реального додатку зводиться до

математичної моделі та аналізується після чого засновуючись на отриманих даних ми проводимо автоматичне збільшення чи зменшення потужностей протягом доби.

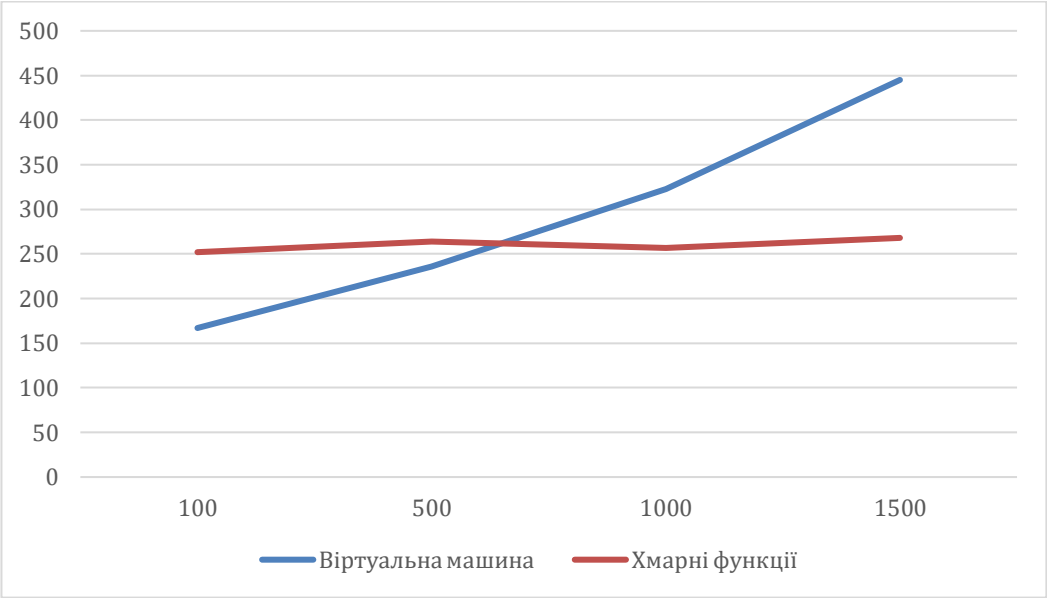

Рисунок 2.5 – Максимальний час відповіді (мс) від кількості запитів в секунду

Таблиця 2.2 – Максимальний час відповіді застосунку

Число запитів / Час відповіді	100	500	1000	1500
Віртуальна машина	167 мс	236 мс	323 мс	445 мс
Хмарні функції	252 мс	264 мс	257 мс	268 мс

В першу чергу за розробки нового веб-застосунку трафік є досить простою мірою, оскільки відсутні реальні дані, тому для аналізу вважалося б, що функція навантаження незмінна і задача вирішувалася б переведенням функції навантаження у функцію вартості, котра теж не змінювалася б з часом

2.3. Опис розробленого метода

Для існуючого додатку картина зовсім інша – в нього вже є трафік. Тому цей трафік необхідно проаналізувати та вже на основі цього аналізу давати рішення відносно вартості проекту. Приблизний вид добового графіку можна побачити на рисунку 2.6.

Рисунок 2.6 – Добовий графік трафіку

Схему розробленого алгоритму можна побачити на рисунку 2.7.

Перед обробкою таких даних вирішено було провести попереднє згладжування даних задля полегшення подальшого їх аналізу.

Для того, щоб згладити отримані дані було вирішено використати метод ковзного середнього.

Ковзне середнє – це інструмент, що використовується для згладжування часових рядів, головним чином застосовуваний для відображення змін біржових котирувань акцій, цін на сировину і так далі. Ковзне середнє – один з найстаріших

і найбільш поширених інструментів технічного аналізу. Ковзне середнє показує середнє значення ціни за певний період часу.

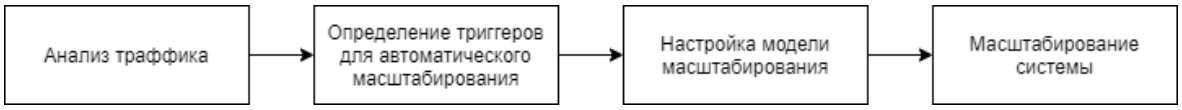

Рисунок 2.7 – Схема алгоритму

Просте ковзне середнє (англ. Simple Moving Average – SMA) – є одними з найбільш простих і популярних індикаторів в технічному аналізі. SMA є звичайним середнім арифметичним від цін за певний період. SMA відноситься до класу індикаторів, які слідують за трендом, воно допомагає визначити початок нової тенденції і її завершення, за його кутом нахилу можна визначити силу (швидкість руху). Іноді ковзне середнє називають лінією тренда [6].

Формула простого ковзного середнього:

$$SMA = \frac{\sum_{i=1}^{n} P_i}{n} \qquad (2.1)$$

де P_i – трафік,

n – основний параметр – довжина згладжування або період SMA (кількість цін що входять у розрахунок ковзного). Іноді цей параметр називають порядком змінного середнього.

Просте ковзне середнє є звичайним середнім арифметичним від даних за певний період. SMA являє собою якийсь показник даних рівноваги за певний період, чим коротше SMA, тим за менший період береться рівновага. Усереднюючи дані, воно завжди слідує за головною тенденцією трафіку, фільтруючи дрібні коливання. Чим менший параметр SMA (коротке ковзне середнє), тим швидше воно визначає нову тенденцію, але й одночасно робить більше помилкових коливань, і навпаки чим більший параметр (довге ковзне середнє), тим повільніше визначається новий тренд, але надходить менше помилкових коливань.

При використанні методу SMA для аналізу трафіку запізнювання на вході і на виході з тренда як правило дуже значне, тому в більшості випадків втрачається велика частина трендового руху.

Один з найбільш серйозних недоліків методу SMA, полягає в тому що воно надає однакові ваги як новим даним, так і більш старим, хоча логічніше було б припустити, що нові дані важливіші, тому що відображають більш близьку ситуацію з трафіком на поточний момент.

На рисунку 2.8 можна побачити результати згладжування методом ковзного середнього.

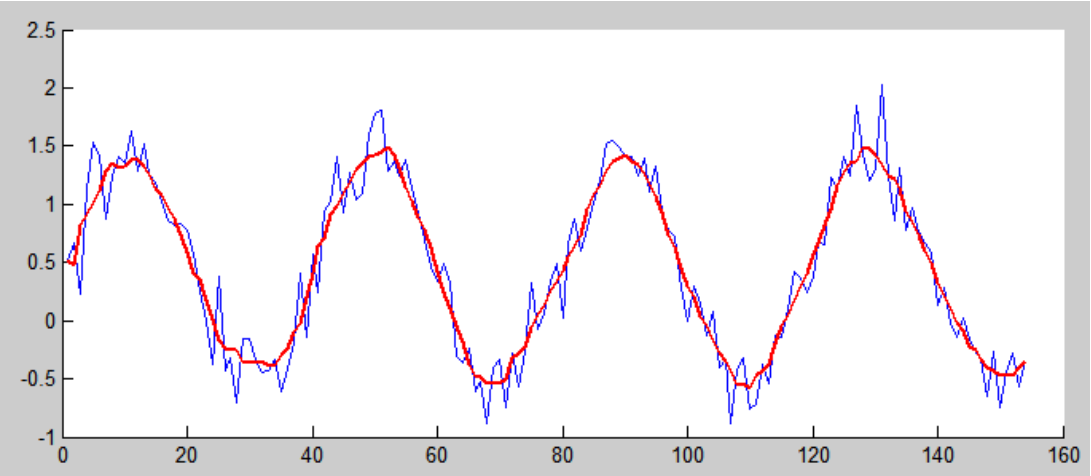

Рисунок 2.8 – Результати згладжування

Маючи згладжені дані було вирішено провести їх апроксимацію для подальшого математичого аналізу.

Для апроксимації було проведено аналіз декількох методів.

Лінійна регресія придатна при моделюванні характеристик, значення яких збільшуються чи спадають з постійною швидкістю. Це найбільш проста в побудові модель досліджуваного процесу.

Поліноміальна лінія тренду корисна для опису характеристик, що мають кілька яскраво виражених екстремумів (максимумів і мінімумів). Вибір ступеня поліному визначається кількістю екстремумів досліджуваної характеристики. Так,

поліном другого ступеня може добре описати процес, що має тільки один максимум чи мінімум; поліном третього ступеня – не більше двох екстремумів; поліном четвертого ступеня – не більше трьох екстремумів і т.д.

Логарифмічна лінія тренду з успіхом застосовується при моделюванні характеристик, значення яких спочатку швидко міняються, а потім поступово стабілізуються.

Степенева лінія тренду дає гарні результати, якщо значення досліджуваної залежності характеризуються постійною зміною швидкості росту. Прикладом такої залежності може служити графік рівноприскореного руху автомобіля. Якщо серед даних зустрічаються нульові чи негативні значення, використовувати статистичну лінію тренду не можна [7].

Експонентну лінію тренду варто використовувати в тому випадку, якщо швидкість зміни даних монотонно зростає. Для даних, що містять нульові чи негативні значення, цей вид наближення також не застосовуємо.

Було вирішено використати поліноміальну регресію з попереднім поділенням даних на проміжки для спрощення побудови поліному.

Використання методів виключення інтервалів, що розглядалися у попередньому підрозділі, накладає єдину вимогу до досліджуваної функції: вона повинна бути унімодальною. Для цього необхідно поділити функцію, що була отримана на попередніх кроках на проміжки. При цьому вказані методи можна використовувати для аналізу як неперервних так і перервних функцій, а також у випадках, коли змінні приймають значення з дискретної множини. Логічна структура пошуку за допомогою методів виключення інтервалів базується на простому порівнянні значень функції у двох пробних точках. Крім того, при такому порівнянні до розрахунку приймається тільки відношення порядку на множині значень функції і не враховується величина різниці між значеннями функції.

У даному підрозділі розглядаються методи пошуку, що дозволяють врахувати відносні зміни значень функції і як наслідок у ряді випадків є більш ефективними, ніж методи виключення інтервалів. Але виграш по ефективності

досягається за рахунок введення додаткової вимоги, відповідно якої функції повинні бути достатньо гладкими.

Основна ідея методів пов’язана з можливістю апроксимації гладкої функції поліномом та послідуючого використання поліному апроксимації для оцінювання координати точки оптимуму. Необхідними умовами ефективної реалізації такого підходу є унімодальність та неперервність функції, що досліджується. Відповідно до теореми Вейєрштраса, якщо функція неперервна на деякому інтервалі, то її з будь-яким ступенем точності можна апроксимувати поліномом достатньо високого порядку.

Таким чином, якщо функція унімодальна і знайдено поліном, який достатньо точно її апроксимує, то координату точки оптимуму функції можна оцінювати шляхом обчислення координати точки оптимуму поліному. Відповідно до теореми Вейєрштраса, якість оцінювання координати точки оптимуму, які отримуються за допомогою поліному апроксимації можна підвищити двома способами: використанням поліному більш високого порядку та звуженням інтервалу апроксимації.

Другий спосіб взагалі кращий, оскільки побудова поліному апроксимації вище третього порядку є надто складною процедурою.

Найпростішим варіантом поліноміальної інтерполяції є квадратична апроксимація, що заснована на тому факті, що функція, що має мінімум на деякому інтервалі, повинна бути хоча б квадратичною. Таким чином, під час реалізації методу оцінювання з використанням квадратичної апроксимації припускається, що в обмеженому інтервалі можна апроксимувати функцію квадратичним поліномом, а потім використовувати побудовану схему для оцінювання координати дійсного мінімуму функції.

Таким чином побудувавши поліноміальну функцію для заданого проміжку часу можна провести її аналіз. Візьмемо похідну функцію і знайдемо її точки екстремуму. На основі цього розрахуємо необхідні конфігурації для кожної з точок екструмуму.

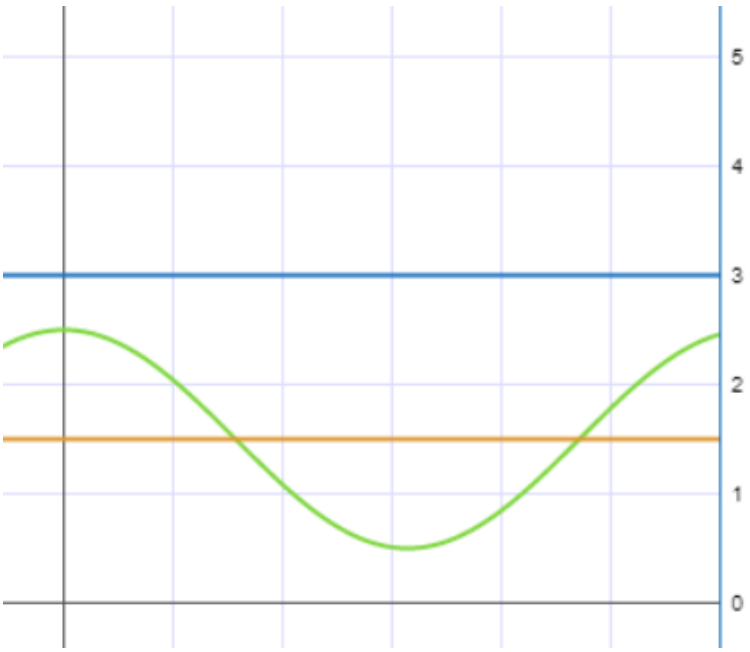

Рисунок 2.9 – Графік навантаження та необхідних конфігурацій

Для статичного аналізу необхідно відмітити точки зміни конфігурації. Це вирішується за рахунок вирішення нерівності де функція повинна мати більше значення, ніж конфігурація для точки локального мінімуму.

$$f(x) > P(x_{min}) \tag{2.2}$$

де $f(x)$– функція трафіку,

$P(x_{min})$ – мінімальна конфігурація для точки локального мінімуму функції.

Після вирішення цієї нерівності отримаємо функцію ресурсів і зможемо оцінити її вартість.

Розрахувавши вартість ресурсів для такого трафіку можливо дати інфраструктурну оцінку вартості проекту. Також, на основі цих даних можливо задати конфігурацію для динамічної зміни конфігурації у реальному часі.

Використаємо дані відносно трафіку для визначення швидкості зростання функції у моментах, коли ми змінюємо конфігурацію. Розрахуємо значення похідної у цих точках.

Знаючи дані у точках переходу ми маємо приблизну швидкість зростання функції у момент, коли необхідно змінювати конфігурацію. Збережемо ці дані для динамічної зміни конфігурації.

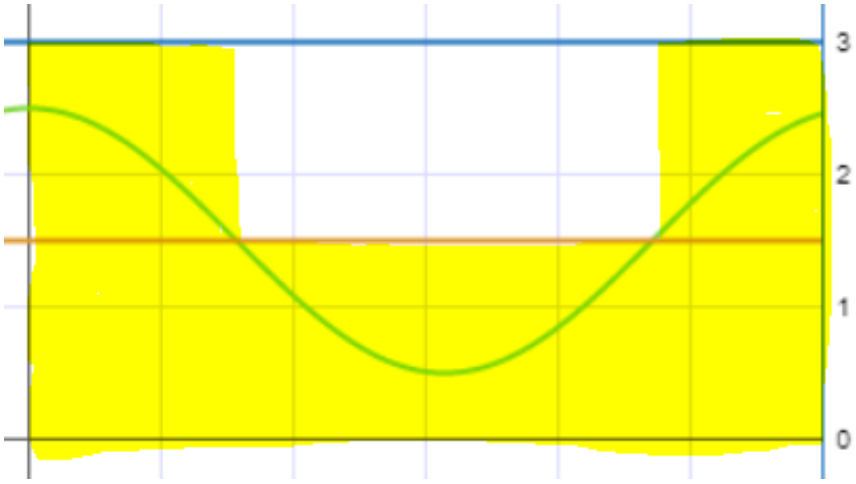

Рисунок 2.10 – Графік необхідних ресурсів

Вирішено, що запланований програмний продукт відслідковуватиме навантаження різних значущих показників веб-застосунку і, в залежності від результатів моніторингу, вносить зміни до інфраструктури підвищуючи її стійкісь. Значущими параметрами для моніторингу є наступні параметри:

– CPU;
– RAM;
– трафік.

Навантаження на CPU (Central Processor Unit - центральний процесор сервера) - це процентне відображення використання ресурсів центрального процесору сервера. [1]

Оперативна пам’ять (англ. Random Access Memory, RAM, пам'ять з довільним доступом) або оперативний запам’ятовуючий пристрій – це

енергозалежна частина системи комп’ютерної пам’яті, в якій під час роботи комп’ютера зберігається виконуваний код, а також вхідні, вихідні та проміжні дані, що обробляються ядрами процесорами.[2]

Трафік є важливою складовою моніторингу оскільки, якщо сервера не зможуть обробити трафік користувача, то це може принести збитків компанії.

В першу чергу буде запускатись крок дослідження середовища – скільки віртуальних ядер у контейнері, яка кількість оперативної пам’яті, та з якою швидкістю змінюється трафік.

CPU навантаження не має досягати значення 100%. Щодо параметру операційної пам'яті бажано не допускати перевищення реального значення навантаження більше ніж 70-90%. Саме ці значення будуть взяті до уваги при розробці програмного застосунку.

Наступним кроком є послідовні циклічні перевірки і порівняння основних значущих параметрів з реальним показниками продуктивності контейнерів. Якщо швидкість зростання трафіку буде зростати і наближатися до швидкості зміни трафіку у точці зміни конфігурації, то відбуватимуться зміни в деплоймент файл (якщо показники вийшли за межі встановленої норми; якщо ж ні – то зміни не відбуваються), який зберігається у json форматі. Після збереження змін згідно з деплоймент файлом відбуваються відповідні зміни інфраструктури, яка знаходиться на моніторингу, для адаптації її до нових навантажень. Якщо, внаслідок автоматичного масштабуваня, кількість контейнерів або серверів буде перевищувати один, то навантаження буде балансуватись за допомогою серверу nginx задля того, щоб уникнути надмірного навантаження однієї з нод веб-застосунку.

3 ПРОЕКТУВАННЯ ПРОГРАММНОГО ЗАБЕЗПЕЧЕННЯ

Після проведення дослідної роботи було вирішено розробити прототип статичного аналізатору.

Після загрузки csv-файлу з трафіком користувач повинен загрузити csv-файл з цінами на ресурси.

Виклик функції можливий з параметрами:

– -t(--traffic) – шлях до файлу з трафіком;

– -p(--price) – шлях до файлу з вартістю;

– -c(--calculate) – розрахувати мінімальну вартість;

– -s(--speed) – розрахувати швидкості зміни трафіку.

На рисунку 3.1 можна побачити приклад роботи програми.

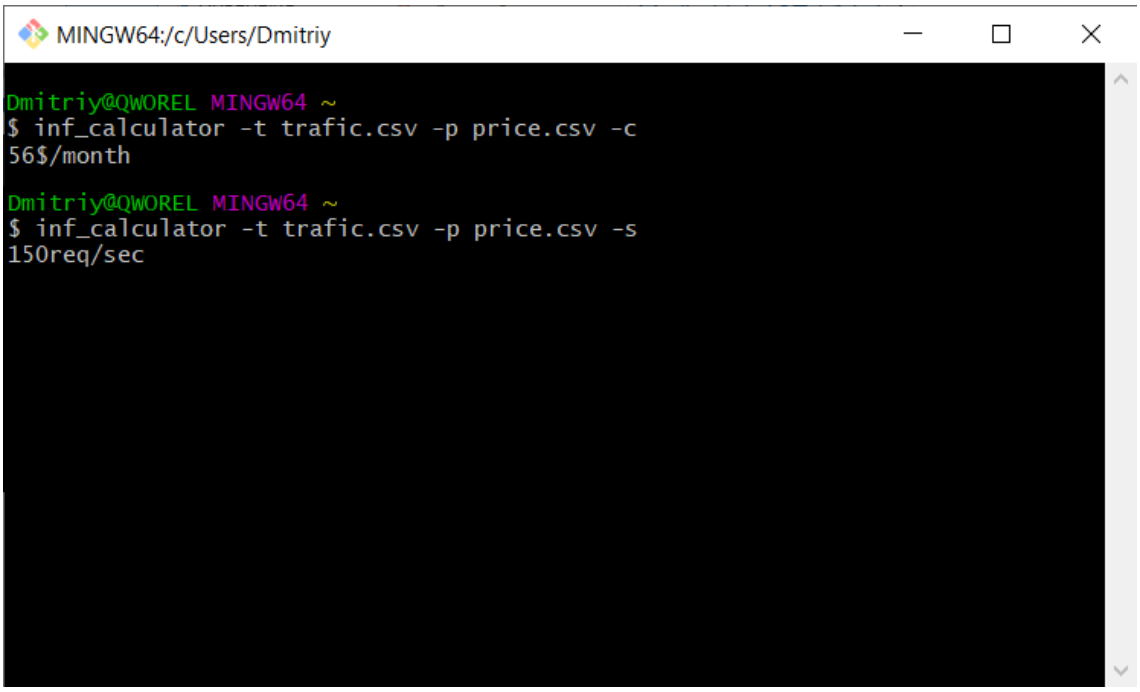

Рисунок 3.1 – Приклад роботи програми

Під час запуску програма проводить згладжування даних з traffic.csv, апроксимує їх та знаходить відповідну конфігурацію.

У подальшому можливо додати виведення результатів обробки даних у форматі csv, оскільки для тестових даних вийшло уникнути складних результатів з декількома точка переходу функції.

Також необхідно змінити алгоритм розділення трафіку на проміжки для спрощення роботи та проаналізувати інші методи, що використовуються для згладжування та апроксимації, оскільки це може збільшити ефективність додатку.

4 ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА АЛГОРИТМУ

Проаналізувавши дані отримані на етапі аналізу типових інфраструктур та маючи програмну реалізацію розробленого алгоритму було вирішено провести експериментальну перевірку його ефективності.

Для аналізу візьмемо трафік одного невеликого додатку. Сервер розгорнуто на AWS на віртуальній машині c5.large. Побачити його навантаження можно на рисунку 4.1 та в таблиці 4.1.

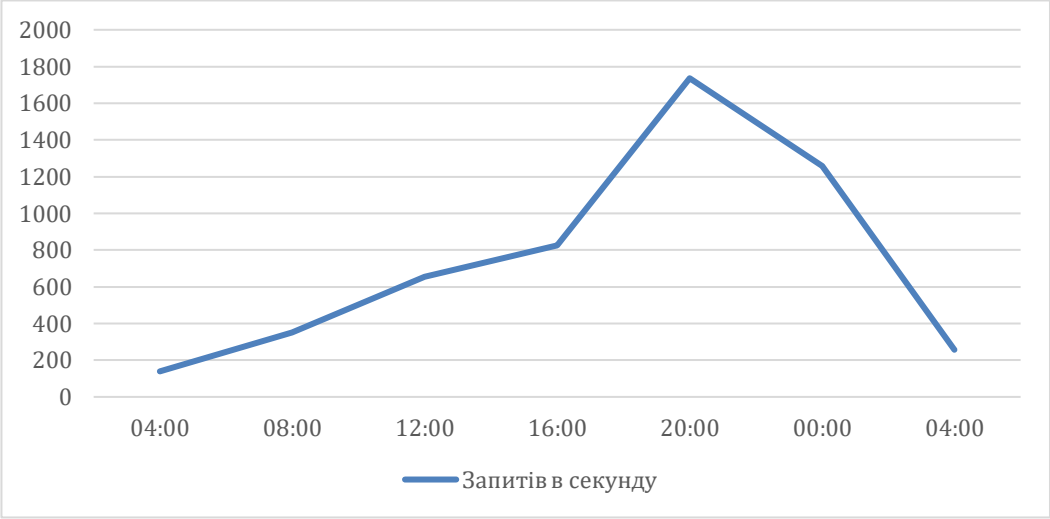

Рисунок 4.1 – Кількість запитів в секунду протягом доби

Таблиця 4.1 – Добове навантаження додатку

Час доби	4:00	6:00	9:00	12:00	15:00	18:00	21:00	00:00	2:00
Кількість запитів в секунду	139	226	421	653	763	1236	1672	1257	805

Поділимо навантаження на проміжки та проведемо згладжування методом ковзного середнього. Після цього проведемо апроксимацію кожного з проміжків за допомогою поліноміальної регресії та знайшовши похідні отримаємо швидкість з

якою збільшується кількість запитів у моменти, коли час обробки запиту стає більшим за 200 мілісекунд для одного сервера [11].

В рамках попереднього аналіза інфраструктури було визначено, що це значення складає 350 запитів. Знайдемо значення похідної для усіх множин з значень з кроков в 350 запитів. Так ми отримаємо дані щодо швидкості зростання кількості запитів в секунду, які будемо використовувати у якості тригера для зміни конфігурації.

Налаштуємо load-balancer, у нашому випадку сервер nginx, так, щоб при збільшені кількості запитів він підіймав нові віртуальні машини, а при зменшені – зменшував їх кількість. Застосувавши даний метод до існуючого продукту отримаємо результати відносно часу відповіді які можна побачити на рисунку 4.2 та таблиці 4.2.

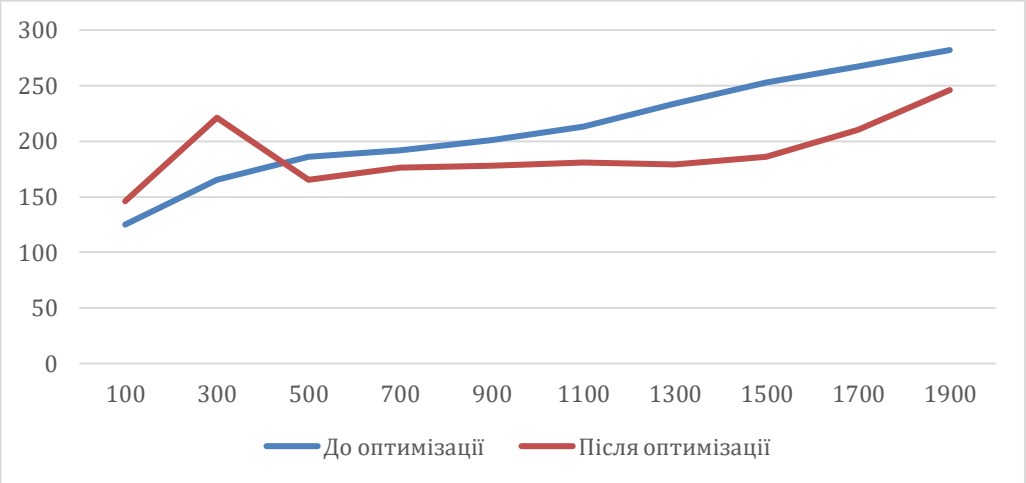

Рисунок 4.2 – Максимальний час відповіді від кількості запитів до і після оптимізації

Проаналізувавши отримані дані ми бачимо, що зі збільшенням росту запитів максимальний час відповіді додатку до оптимізації поступово зростає. Після оптимізації ми можемо помітити, що на малій кількості запитів час їх обробки більше, ніж у інсталяції до оптимізації. Це може пояснюватись тим, що задля оптимізації інфраструктури були використані інстанси t2.small, котрі за своїми характеристиками програють c5.large, які використовувалися до оптимізації. Пік на

300 запитах обумовлено тим, що тригер, який було налаштовано задля підняття нових віртуальніх машин ще не відпрацював і других інстанс для обробки запитів не був піднятий. В подальшому динамічне маштабування дозволило втримати час відповіді у зручній для користувача границі.

Таблиця 4.2 – Результати оптимізації

Кількість запитів в секунду	100	300	500	700	900	1100	1300	1500	1700	1900
Час відповіді до оптимізації, мс	125	165	186	192	201	213	234	253	267	282
Час відповіді після оптимізації, мс	146	221	165	176	178	181	179	186	210	246

Збільшення часу відповіді після 1500 запитів говорить про те, що даний тригер погано реагує на непердбачене навантаження, котре більше за дані на основі яких тригер було побудовано, що ставить під сумнів його корисність та призводить до необхідность аналізу інших тригерів для зміни інфраструктури.

З фінансової точки зору до оптимізації вартість інфраструктури склада 72 долари на місяці: рахуємо, що сервер c5.large використовувася кожного дня протягом місяця за вартості 0,097 долара в час. Інфраструктура після оптимізації коштувала 64 доларів: протягом експеримента вийшло, що одна віртуальна машина була активна 24 години протягом місяця, ще дві – 16 годин, і ще дві – 8 годин за вартості в 0,027 доларів за одну віртуальні машину за годину. Це пов’язано з тим, що хмарні сервіси спонукають ефективно використовувати ресурси ща рахунок своїх цінових політик.

Використання великої кількості тимчасових віртуальніх машин (у нашому експерименті від 1 до 5) вийшло вигіднішим не лише з точки зору користувацького досвіду, але й з фінансової сторони питання. Проаналізувавши дані більш скурпульозно було з’ясовано, що при збільшені вартості віртуально машини для оптимізованої інфраструктури хоча б на цент призвело б до програшу з боку фінансів, а саме вартість складала б 86 доларів на місяць.

ВИСНОВКИ

В результаті атестаційної роботи магістра було запропоновано алгоритм для оптимізації інфраструкти, що був заснований на методах математичного аналізу:

– згладжуємо дані про навантаження існуючої інфраструктури;

– проводимо апроксимацію;

– знаходимо критичні точки отриманої функції і використовуємо їх у якості тригерів для масштабування.

Реалізація проекту відбувалася за допомогою мови програмування Java, CSV як формат вхідних даний, Під час роботи використовувалися InteliJ Idea, nginx та AWS.

В ході виконання роботи були закріплені знаня та навички відносно аналізу архітектури програмного застосунку, аналізу та розробки функціональних та нефункціональних вимог, навички роботи з хмарними сервісами та мовами Java та Python.

Було проведений аналіз проблемної галузі, існуючих рішень проблеми, типових варіантів конфігурації та аналізу інфраструктур, існуючих методів оптимізації.

В ході атестаційної роботи магістра було:

– проведено аналіз типових інфраструктур з метод створення метода для їх аналізу;

– розроблено метод для аналізу інфраструктур;

– розроблено алгоритм для оптимізації інфраструктури;

– виконано програмну реалізацію алгоритму;

– проведені експериментальні дослідження розробленого алгоритму.

За результатами атестаційної роботи було розроблено презентацію (див. додаток А).

Була подана стаття до опублікування до наукового журналу «East European Science Journal» стаття «Evaluation and optimization of software product infrastructure». Статтю наведено у додатку В.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Цваліна К., Абрамс Б. Інфраструктура програмних проектів [Текст] / Цваліна К., Абрамс Б. М. : Вильямс, 2011.-418 с.

2. Michael A Murphy, Brandon Kagey, Michael Fenn, and Sebastien Goasguen. Dynamic provisioning of virtual organization clusters. In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pages 364–371. IEEE Computer Society, 2009.

3. Fangzhe Chang, Jennifer Ren, and Ramesh Viswanathan. Optimal resource allocation in clouds. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pages 418–425. IEEE, 2010.

4. Nathan Malishev. AWS Lambda Cold Start Language Comparisons, 2019 edition. URL: https://levelup.gitconnected.com/aws-lambda-cold-start-language-comparisons-2019-edition-%EF%B8%8F-1946d32a0244

5. Peter Lin. So You Want High Performance / URL: https://tomcat.apache.org/articles/performance.pdf

6. Вирт Н. Алгоритмы и структуры данных. [Текст] / Н.Вирт. – М.:, Издат-во "Вильямс", 1998г

7. Герасимов Б.М. Человеко-машинные системы принятия решений с элементами искусственного интеллекта / Б.М. Герасимов, В.А. Тарасов, И.В. Токарев. – Киев: Наукова думка, 1993. – 286 с.

8. Еккель Б., Бек К., Брант Дж. Thinking in Java [Текст] / Еккель Б., Бек К., Брант Дж. - Спб.: БХВ-Питер, 2015. - 1159 с.

9. Мартин Р. Чистий код: створення, аналіз та рефакторінг [Текст] / Мартин Р. Спб.: БХВ-Питер, 2010.-466 с.

10. Шилдт Г., Холмс Дж. Искусство программирования на Java [Текст] / Шилдт Г., Холмс Дж. - М. : Вильямс, 2005. - 336 с.

11. Jordan Kasteler. What Is Time To First Byte, And How To Improve It/ URL: https://www.searchenginepeople.com/blog/16081-time-to-first-byte-seo.html

12. Vechur O.V., Shevchenko D.O. Evaluation and optimization of software product infrastructure//East European Science Journal. 2020 №57.

13. Chalyi, S., Leshchynskyi, V. and Leshchynska I. (2019), “Designing explanations in the recommender systems based on the principle of a black box”, Advanced Information Systems, Vol. 3, No. 2, pp. 47–51/ URL: https://doi.org/10.20998/2522-9052.2019.2.08

14. Frank Doelitzscher, Markus Held, Christoph Reich, and Anthony Sulistio. Viteraas: Virtual cluster as a service. In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pages 652–657. IEEE, 2011.

15. ДСТУ 3008-95. Державний стандарт України. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення [Текст] ДСТУ 3008-95. – Чинний з 01.01.2006. – К.: Держспоживстандарт України, 2005, 38с.