Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
Міністерство освіти та науки України

Кваліфікаційна наукова
праця на правах рукопису

КАРПЕНКО АНДРІЙ СЕРГІЙОВИЧ

УДК 004.65.056:004.75

ДИСЕРТАЦІЯ

МЕТОДИ ТА ЗАСОБИ ЗАБЕЗПЕЧЕННЯ КІБЕРБЕЗПЕКИ ГЛОБАЛЬНО-РОЗПОДІЛЕНИХ РЕПЛІКОВАНИХ СИСТЕМ ЗБЕРІГАННЯ ДАНИХ З КОНТРОЛЬОВАНОЮ УЗГОДЖЕНІСТЮ

125 Кібербезпека та захист інформації
(шифр і назва спеціальності)
12 Інформаційні технології
(галузь знань)
Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ А. С. Карпенко
(підпис, ініціали та прізвище здобувача)

Науковий керівник Горбенко Анатолій Вікторович,
доктор технічних наук, професор

Харків – 2023

АНОТАЦІЯ

Карпенко Андрій Сергійович. Методи та засоби забезпечення кібербезпеки глобально-розподілених реплікованих систем зберігання даних з конрольованою узгодженістю. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 125 Кібербезпека та захист інформації. – Національний аерокосмічний університет імені М. Є. Жуковського «Харківський авіаційний інститут», Харків, 2023.

Дисертаційна робота присвячена забезпеченню кібербезпеки глобально-розподілених реплікованих систем зберігання даних й підвищенню їхньої стійкості до загроз порушення готовності та узгодженості в умовах наявного протиріччя між цими властивостями за рахунок використання відповідних методів та засобів.

Об’єктом дослідження є процеси побудови та забезпечення безпеки глобально-розподілених систем зберігання даних в умовах кіберзагроз..

Розроблено комплекс нових та удосконалених моделей для глобально-розподілених систем зберігання даних, які забезпечують: 1) деталізацію загроз кібербезпеки ГРРС; 2) формалізацію опису патернів розгортання ГРРС у хмарному середовищі з урахуванням доменів готовності за допомогою апарату теоретико-множинного представлення; 3) оцінювання готовності та зменшення часу обслуговування в умовах кібератак завдяки використанню механізму надлишкових читань за допомогою гібридного імітаційного підходу.

Вперше запропоновано метод динамічного керування рівнем узгодженості ГРРС, який, на відміну від відомих, базується на побудові доменів змішаного робочого навантаження та дозволяє підвищити готовність системи, гарантуючи при цьому строгу узгодженість даних для підвищення стійкості до DDoS атак.

Удосконалено метод надлишкових читань ГРРС, який ґрунтується на використанні надлишковості щодо встановленого рівня узгодженості операцій читання та дозволяє зменшити екстремальні часові затримки й підвищити

готовність при встановленому обмеженні на час обслуговування або цілісність в умовах кіберзагроз порушення даних та відмов в обслуговуванні.

Ключові слова: великі дані, бази даних, нереліаційні бази даних, nosql, реплікація, цілісність, узгодженість, готовність, цілісність інформації, функційна безпечність, хмарні технології, безпека, конфіденційність, кібербезпека, модель загроз, вразливість, кіберзагроза, контрзахід.

Список публікацій здобувача:

1. A. Gorbenko, O. Tarasyuk, and A. Karpenko, «Analysis of Trade-offs in Fault-Tolerant Distributed Computing and Replicated Databases», in Proceedings of 11th International Conference on Dependable Systems, Services and Technologies, DESSERT, Kyiv, 2020, pp. 1-6. DOI: 10.1109/DESSERT50317.2020.9125078 (стаття у працях конференції, Scopus)

2. A. Gorbenko, A. Karpenko, and O. Tarasyuk, «Performance evaluation of various deployment scenarios of the 3-replicated Cassandra NoSQL cluster on AWS», Radioelectronic and computer systems, no. 4 (100), pp. 157-165, 2021. DOI: 10.32620/reks.2021.4.13 (наукове фахове видання категорії А, Scopus, Q3)

3. А. Карпенко, О. Тарасюк, і А. Горбенко, «Дослідження узгодженості та продуктивності у нереляційних реплікованих баз даних», Сучасні інформаційні системи, т. 5, №3, pp. 66-75, 2021. DOI: doi.org/10.20998/2522-9052.2021.3.09 (наукове фахове видання категорії Б)

4. О. Тарасюк, А. Горбенко, і А. Карпенко, «Розвиток архітектур, теорем та моделей властивостей розподілених систем зберігання даних», Вимірювальна та обчислювальна техніка в технологічних процесах, №2, pp. 5-13, 2022. DOI: 10.31891/2219-9365-2022-70-2-1 (наукове фахове видання категорії Б)

5. J. Ahmed, A. Karpenko, O. Tarasyuk, A. Gorbenko, and A. Sheikh-Akbari «Consistency issue and related trade-offs in distributed replicated systems and databases: a review», Radioelectronic and computer systems, no. 2 (106), pp. 171-179, 2023. DOI: 10.32620/reks.2023.2.14 (наукове фахове видання категорії А, Scopus, Q3)

ANNOTATION

Karpenko Andrii. Methods and tools of ensuring cyber security of globally distributed replicated data storage systems with controlled consistency. – Manuscript copyright.

Thesis on competition of scientific degree of Doctor of Philosophy by specialty 125 Cybersecurity and information protection. – National Aerospace University "Kharkiv Aviation Institute", Kharkiv, 2023.

The thesis is devoted to ensuring the cyber security of globally distributed replicated data storage systems and increasing their resistance to threats of availability and consistency in the conditions of the existing contradiction between these properties due to the use of appropriate methods and means.

The object of research is the processes of building and ensuring the security of globally distributed data storage systems under conditions of cyber threats.

A complex of new and improved models for globally distributed data storage systems has been developed, which provide: 1) detailing of threats to the cyber security of globally distributed replicated data storage systems; 2) formalization of the description of the deployment patterns of the globally distributed replicated data storage systems in the cloud environment, taking into account the domains of readiness using the apparatus of the theoretical-multiple representation; 3) assessment of readiness and reduction of service time in the conditions of cyber attacks due to the use of the mechanism of redundant readings with the help of a hybrid simulation approach.

For the first time, a method of dynamic control of the consistency level of globally distributed replicated data storage systems was proposed, which, unlike the known ones, is based on the construction of mixed workload domains and allows to increase the system readiness, while guaranteeing strict data consistency to increase resistance to DDoS attacks.

The method of redundant reads of globally distributed replicated data storage systems has been improved, which is based on the use of redundancy with respect to the set level of consistency of read operations and allows to reduce extreme time delays and

increase readiness with the set limit on service time or integrity in the conditions of cyber threats of data breaches and denials of service.

Key words: big data, databases, non-relative databases, NoSQL, replication, integrity, consistency, availability, integrity of information, functional safety, cloud technology, security, privacy, cybersecurity, threat models, vulnerability, cyberthreat, countermeasure.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ГРРС	– Глобально-розподілена реплікована система зберігання даних.
ІБ	– Інформаційна безпека.
ІТ	– Інформаційна технологія.
ТМО	– Теорія масового обслуговування.
ACID	– Atomicity, consistency, isolation, durability.
AWS	– Amazon web services.
BASE	– Basically available, soft state, eventually consistent.
Dos/DDos	– (Distributed) Denial-of-service, атака на відмову в обслуговуванні системи.
FW	– Frameworks
CAP	– Consistency, availability, partition tolerance.
NoSQL	– Not only SQL, no SQL.
STRIDE	– Spoofing, tampering, repudiation, information disclosure, denial of service, evaluation of privilege.
YCSB	– Yahoo cloud serving benchmark.
PaaS	– Platform-as-a-service

ВСТУП

Обґрунтування вибору теми дослідження. На сьогоднішній день інформаційні технології впливають на всі сфери життєдіяльності суспільства. Популярність відомих інформаційних сервісів таких як Google, Twitter, Instagram, Facebook, дали поштовх розвитку глобально-розподілених систем зберігання великих даних. Розподілені інформаційні системи [1], компоненти яких зв’язані через глобальну мережу Інтернет, також активно використовуються у критично важливих застосунках, відмови в яких можуть впливати на бізнес-процеси та життя людей. Зокрема, йдеться про системи зберігання банківських або медичних даних, системи підтримки безпеки польотів, системи військового призначення та інші важливі галузі життєдіяльності.

За своєю природою глобально-розподілені системи, що працюють через публічну мережу Інтернет, особливо схильні до відмов компонентів і каналів зв’язку, а також кіберзагроз, спрямованих на порушення готовності (DDoS атаки) та цілісності даних завдяки компрометації вузлів цих систем.

У той час як сучасні інформаційні технології мають широкий спектр засобів криптографічного захисту, методів ідентифікації та аутентифікації, які успішно вирішують завдання забезпечення конфіденційності даних, одночасне забезпечення високої готовності та узгодженості інформації – двох інших важливих властивостей кібербезпеки, у глобально-розподілених системах зберігання даних ускладнено через наявність протиріч між ними [2, 3]. Крім того, можлива неузгодженість між даними, що зберігаються на різних вузлах-репліках розподілених систем, є додатковою загрозою до цілісності інформації.

Традиційні механізми підвищення готовності в умовах інформаційних вторгнень та DDoS-атак, зокрема, резервування компонентів та реплікація даних, вимагають забезпечення синхронізації між репліками. Від цього також залежить і цілісність даних, яка на системному рівні перетворюється на необхідність забезпечення узгодженості між вузлами-репліками в умовах можливої компрометації деяких з них [4]. Однак, для глобально-розподілених систем,

компоненти яких знаходяться віддалено один від одного та об’єднані через глобальну мережу Інтернет, оновлення не можуть поширюватись миттєво [5, 6, 7, 8]. Це безумовно ускладнює забезпечення узгодженості між компонентами системи та підтримки готовності системи в цілому. Наприклад, очікування результату від усіх реплік для забезпечення сильної узгодженості, особливо в умовах DDoS-атак може призвести до того, що відповідь від системи буде отримано лише після закінчення встановленого часу очікування (тайм-ауту), тобто, матиме місце порушення готовності. У той же час зниження рівня узгодженості підвищує ризик порушення цілісності інформації при компрометації окремих реплік. В цих умовах підвищення продуктивності системи та зниження часових затримок дозволить, з одного боку, підвищити її готовність та забезпечити більшу стійкість до DDoS-атак, а з іншого, – дає змогу збільшити рівень узгодженості без зниження готовності.

Отже, забезпечення кібербезпеки глобально-розподілених інформаційних систем, зокрема властивостей готовності та цілісності, слід розглядати у комплексі з іншими властивостями таких систем відповідно до теорії гарантоздатності, з урахуванням наявних протиріч та компромісів між ними. Вагомий внесок у розвиток теорії гарантоздатності розподілених систем, яка поєднує методи та механізми забезпечення й оцінки надійності, продуктивності та інформаційної безпеки внесли А. Авіженіс, Ж. Лапрі [4], В. Харченко, А. Романовський [5], А. Горбенко [9] та інші. Однак, відомі дослідження не повною мірою враховують вплив продуктивності розподілених систем на кібербезпеку, зокрема на властивості готовності та узгодженості, а також наявність протиріччя між цими двома властивостями.

Таким чином, актуальною науково-прикладною задачею є розроблення та удосконалення методів та засобів для забезпечення кібербезпеки глобально-розподілених реплікованих систем (ГРРС) зберігання даних й підвищення їхньої стійкості до загроз порушення готовності та узгодженості даних в умовах наявного протиріччя між цими властивостями ГРРС.

Об’єкт дослідження – процеси побудови та забезпечення безпеки глобально-розподілених систем зберігання даних в умовах кіберзагроз.

Предметом дослідження – моделі, методи та засоби підвищення кібербезпеки глобально-розподілених реплікованих систем зберігання даних.

Мета і завдання дослідження. Метою дослідження є забезпечення кібербезпеки глобально-розподілених реплікованих систем зберігання даних, шляхом підвищення готовності, контролю та керування узгодженістю даних.

Для досягнення мети дослідження необхідно вирішити наступні завдання:

– дослідити та удосконалити моделі загроз для ГРРС;

– виконати аналіз методів і засобів забезпечення кібербезпеки ГРРС;

– розробити теоретико-множинну модель патернів розгортання ГРРС з урахуванням доменів готовності хмарних провайдерів в умовах кібератак;

– розробити метод динамічного керування рівнем узгодженості ГРРС для забезпечення кібербезпеки;

– розробити метод надлишкових читань для підвищення готовності (доступності даних) ГРРС в умовах кібератак.

Методи дослідження. У дисертаційній роботі використовувались методи математичного моделювання, теорії ймовірності та математичної статистики, теорії систем масового обслуговування при розробці та дослідженні методу динамічного керування рівнем узгодженості операцій читання і запису для глобально-розподілених систем зберігання даних в умовах кібератак, методу надлишкових читань та гібридної імітаційної моделі для оцінювання ефективності методу надлишкових читань для забезпечення кібербезпеки.

Наукова новизна отриманих результатів:

– **розроблено комплекс нових та удосконалених** *моделей* для глобально-розподілених систем зберігання даних, які забезпечують: 1) деталізацію загроз кібербезпеки ГРРС; 2) формалізацію опису патернів розгортання ГРРС у хмарному середовищі з урахуванням доменів готовності за допомогою апарату теоретико-множинного представлення; 3) оцінювання готовності та зменшення часу

обслуговування в умовах кібератак завдяки використанню механізму надлишкових читань за допомогою гібридного імітаційного підходу;

– **вперше запропоновано** *метод* динамічного керування рівнем узгодженості ГРРС, який, на відміну від відомих, базується на побудові доменів змішаного робочого навантаження та дозволяє підвищити готовність системи, гарантуючи при цьому строгу узгодженість даних для підвищення стійкості до DDoS атак;

– **удосконалено** *метод* надлишкових читань ГРРС, який ґрунтується на використанні надлишковості щодо встановленого рівня узгодженості операцій читання та дозволяє зменшити екстремальні часові затримки та підвищити готовність при встановленому обмеженні на час обслуговування або цілісність в умовах кіберзагроз порушення даних та відмов в обслуговуванні.

Особистий внесок здобувача полягає у розробці методів, моделей та інструментальних засобів, які забезпечують вирішення поставлених задач, описаних вище. Всі основні результати отримані автором особисто та опубліковано у роботах [10, 11, 12, 13, 14].

У працях, які опубліковані у співавторстві, автору належать: теоретика-множинна модель опису патернів розгортання [11]; модель загроз для глобально-розподілених систем зберігання даних [10]; метод динамічного керування рівнем узгодженості ГРРС [12]; метод надлишкових читань глобально-розподілених систем зберігання даних [10]; гібридної імітаційної моделі для оцінки ефективності методу надлишкових читань.

Апробація матеріалів дисертації. Основні положення та ідеї дисертаційної роботи доповідалися та обговорювалися на міжнародному науково-технічному семінарі «Критичні комп’ютерні технології та системи» (КриКТехС) на кафедрі «Комп’ютерних систем, мереж та кібербезпеки» Національного аерокосмічного університету ім. М.Є. Жуковського «ХАІ» (Харків, 27 жовтня 2023 р.), на науково-технічному семінарі «Гарантоздатні Інформаційні Технології» (ГІТ) на кафедрі «Комп’ютерних систем, мереж та кібербезпеки» Національного аерокосмічного

університету ім. М.Є. Жуковського «ХАІ» (Харків, 28 жовтня 2020 р.), а також на конференції «Dependable Systems, Services and Technologies» (Київ, 2020).

Зв'язок з науковими програмами, планами, темами. Дисертаційна робота виконана у Національному аерокосмічному університеті ім. М.Є. Жуковського «Харківський авіаційний інститут» відповідності з державними програмами та планами НДР:

– НДР «Методологічні засади та технології оцінювання та забезпечення безпеки (захисту) критичних інформаційних інфраструктур» (№ ДР 0119U100979, 2019-2021);

– НДР «Методи, моделі та інформаційні технології підвищення надійності та безпечності складних ІТ-систем на етапах розроблення та впровадження» (№ Д/Р 0121U113842, 2021-2023).

Роль автора у зазначених НДР в яких автор був безпосереднім виконавцем, полягає у розробці методів та засобів забезпечення кібербезпеки глобально-розподілених реплікованих систем зберігання даних.

Практичне значення отриманих результатів. Практичні результати полягають у доведенні теоретичних положень дисертаційної роботи до конкретних алгоритмів, інструментальних засобів та рекомендацій для побудови глобально-розподілених систем зберігання даних та забезпечення стійкості до DDoS атак і загроз порушення даних. Результати дисертаційної роботи впроваджено у додатку В:

– ТОВ «Софтсерв Інновації» (акт впровадження від 28 серпня 2023) при проектуванні системи підтримки асинхронних запитів для системи електронної комерції;

– у навчальному процесі Національного аерокосмічного університету ім. М. Є. Жуковського «Харківський авіаційний інститут» (акт впровадження від 16 березня 2023);

– у навчальному процесі Leeds Beckett University при формуванні курсу «Хмарних обчислень» (акт впровадження від 20 липня 2022).

Структура та обсяг дисертації. Дисертація складається із вступу, чотирьох розділів, висновку, списку виконаних джерел і додатків. Загальний обсяг дисертації складає 172 сторінок, з яких анотація на 5 сторінках, зміст на 3 сторінках, перелік умовних позначень на 9 сторінці, основний текст на 133 сторінках, список використаних джерел із 100 найменувань на 10 сторінках, додатки на 39 сторінках. Робота містить 30 таблиць та 50 малюнків.

Публікації. За темою дисертаційної роботи було опубліковано 5 наукових праць, серед яких: 2 статті у наукових фахових виданнях України категорії Б; 2 статті у англомовних журналах категорії А, що індексовані у базі даних Scopus (квартіль Q3); 1 публікація в працях міжнародної конференції, матеріали якої включено у базу даних Scopus.

РОЗДІЛ 1
АНАЛІЗ РОЗВИТКУ, ОСОБЛИВОСТЕЙ ТА ПРОБЛЕМ КІБЕРБЕЗПЕКИ ГЛОБАЛЬНО-РОЗПОДІЛЕНИХ СИСТЕМ ЗБЕРІГАННЯ ВЕЛИКИХ ДАНИХ

1.1. Еволюція характеристик великих даних

«Великі дані» (Big Data) – це структуровані та неструктуровані дані, які мають велику різноманітність, надходять з великою швидкістю та у великих обсягах, що перевищує можливості зберігання на окремому сервері. На сьогоднішній день, накопичувач SSD має ємність 30,72 Терабайти, чи можна це вважати великими даними? З цього приводу дослідники додають, що «великі дані» потрібно розглядати ширше, як соціально-економічний феномен, який дозволяє аналізувати величезні об’єми даних за допомогою нових технологій та робити відповідні висновки. Сама концепція «великих даних» почала зароджуватися у 60-70 роках минуло століття та пов’язана з появою перших центрів обробки інформації та розробкою реляційних баз даних. Вже на початку 2000-х років з ростом популярності Facebook, Twitter, YouTube об’єми даних почали рости. У 2005 році було розроблено фреймворк з відкритим кодом для зберігання та аналізу даних Hadoop. У наступні роки обсяг даних різко зріс, що додало поштовх у розвитку NoSQL (Non SQL, Not only SQL) баз даних. «Великі дані» дозволяють, базуючись на великому об’ємі даних, приймати правильні рішення з високою ймовірністю довіри до них, але вони не позбавлені проблем. По-перше, великий обсяг даних треба зберігати, приблизно кожен рік обсяг даних збільшується вдвічі і тому треба йти у ногу з часом та використовувати новітнє програмне забезпечення. По-друге, зберігати дані не достатньо, їх потрібно підготувати, а саме провести «очищення» даних (clean data), перш ніж їх можна буде використати. [15]

Використання або аналітика великих даних дає змогу отримати звіти, які відображають тенденції у різних сферах життєдіяльності на які вона зорієнтована. Однак для аналітики великих даних використовується багато міркувань, зазвичай їх називають характеристиками або V`s великих даних. У 2001 році у статті

«3D Data Management: Controlling Data Volume, Velocity and Variety» написаною Д. Лені, були запропоновані основні характеристики великих даних: обсяг, швидкодія та різноманітність [16]. Але на даний час, з'явилося багато міркувань, які також треба враховувати, і тому запропонований 3V термін, який визначає атрибути великих даних, набув подальшого розвитку. Станом на зараз можна знайти 4V, 5V, 8V та навіть 42V. Додаткові властивості дають змогу до розширення аналітики та пов'язаних з нею процесами. Важливо зауважити, що подальший розвиток V`s розширює попередній термін [17]:

– обсяг (volume) – великий об'єм даних, який генерується з різних джерел, наприклад, соціальних мереж, мобільних застосунків тощо;

– швидкодія (velocity) – можливість швидкісного (майже у реальному часі) аналізу, обробки та зберігання великого об'єму даних;

– різноманітність (variety) – це типи даних, які можуть мати різну структуру в залежності від джерела, а саме структуровані, напівструктуровані та неструктуровані;

– достовірність (validity) – це можливість аналізу отриманих даних з огляду до її відповідності та повноти;

– цінність (value) – це можливість отримати з даних, корисну інформацію та з більшою ймовірністю прийняти правильне рішення;

– візуалізованість (visualization) – це можливість інтерпретації даних для її подачі у вигляді графіків, схем з метою швидкого розуміння;

– вагомість (volatility) – це можливість, яка дає змогу відповісти наскільки дані правильні для свого призначення та має певну схожість з достовірністю.

1.2. Еволюція властивостей систем зберігання даних

На сьогоднішній день ми живемо у світі інформаційних технологій, які проникли у всі можливі сфери життєдіяльності людства. Нещодавній розвиток систем управління базами даних співпав з досягненнями у технологіях паралельного обчислення. З огляду на цей факт, з'явився новий клас сховищ даних, а саме глобально-розподілені нереляційні системи управління базами даних (QCon), і саме вони зараз широко застосовуються у Twitter, Instagram, Facebook, Google та інших сучасних розподілених інформаційних системах для збереження та обробки величезних обсягів даних.

Системи управління базами даних пройшли певний шлях еволюції від архітектури мейнфреймів до глобально-розподілених нереляційних сховищ даних. Здебільшого ця еволюція була зумовлена появою нових вимог до збереження даних та забезпечення доступу до них. Це пов'язано з ростом обсягів інформації, зростанням швидкості її надходження та необхідністю одночасного обслуговування величезної кількості різноманітних споживачів інформації.

Вперше акронім ACID [18] був сформований А. Рейтером і Т. Хардером базуючись на дослідженнях Д. Грейа. Теорема ACID (Atomicity, Consistency, Isolation, Durability) представляє собою набір властивостей, які повинна мати реляційна база даних для забезпечення достовірності даних, що зберігаються:

– атомарність (Atomicity) – властивість, яка гарантує, що кожна транзакція буде виконана в повному обсязі або не виконається взагалі, не допускаючи проміжного стану;

– узгодженість (Consistency) – властивість, яка гарантує, що кожна транзакція не порушує цілісність даних у ній;

– ізольованість (Isolation) – властивість, яка гарантує, що будь-яка операція читання або запису не буде змінена іншими операціями читання або запису;

– довговічність (Durability) – властивість, яка гарантує, що зміни внесені до бази даних, які були успішно завершені, збережуться назавжди, навіть у разі відмови системи.

BASE (Basically Available, Soft state, Eventually consistent) [19] є більш гнучкою моделлю, ніж ACID. Вона забезпечує ряд переваг перед ACID-сумісними базами даних, які за своїм дизайном не є у повній мірі розподіленими системами і не можуть забезпечити ефективну роботу в умовах відмов каналів зв'язку. Проте, властивості BASE не надають жорстких гарантій щодо узгодженості даних на відміну від моделі ACID:

– основна доступність (basically available) – властивість, яка не гарантує узгодженість при основних операціях читання або запису, при цьому існує ймовірність того, що при запиті на читання повертаються не останні дані або при операції запису дані не зберігаються;

– м'який стан (soft state) – властивість, яка говорить, що стан системи може змінюватися з часом при зміні кінцевої узгодженості;

– кінцева узгодженості (eventually consistency) – властивість, яка говорить, що система після деякого проміжку часу досягне бажаного рівня узгодженості для операцій запису та матиме актуальні дані.

Модель BASE надала поштовх для розвитку нереляційних баз даних NoSQL. Властивості, передбачені цією моделлю, забезпечують: високу масштабованість сховищ даних; велику продуктивність, швидкість виконання операцій читання та запису; простоту реалізації механізмів реплікації; можливість для зберігання та аналізу дійсно великих даних, так званих Big Data.

Гіпотеза CAP (акронім утворений з англомовних іменувань властивостей розподілених інформаційних систем: Consistency, Availability, Partition tolerance) вперше була озвучена у 2000 році Е. Брюером [2]. Згодом у 2002 році С. Гілберт та Н. Лінч довели гіпотезу та трансформували її у теорему.

Теорема CAP стверджує, що для будь-якої розподіленої інформаційної системи неможливо одночасно забезпечити виконання більш ніж двох із зазначених властивостей, а саме:

– узгодженість (consistency) – властивість, яка гарантує, що кожна операція читання повертає актуальні дані;

– доступність (availability) – властивість, яка не гарантує, що репліка має актуальні дані, репліка відповідає без помилок;

– стійкість до розподілення (partition tolerance) – властивість, яка гарантує функціонування системи навіть при втраті повідомлень між репліками або їх відмови.

У разі відмови комунікаційних каналів і втрати зв’язку між вузлами в розподілених реплікованих системах баз даних неможливо одночасно підтримувати, як доступність так і узгодженість даних. Тобто або вся операція читання повинна бути скасована для збереження узгодженості, або система буде продовжувати підтримувати доступність, а отже, нехтується узгодженість.

Наступним етапом розвитку CAP стала теорема PACELC [3]. Теорема PACELC була вперше описана та сформульована Д. Абаді у 2012 році. Вона базується на основі CAP теореми та має аналогічні властивості, але дає більш ширшу картину з іншого боку. Якщо у мережі присутні розділи, то можна вибрати між доступністю та узгодженістю в іншому випадку можна робити вибір між затримкою та узгодженістю. PACELC йде далі і стверджує, що існує додатковий компроміс: між затримкою та узгодженістю, навіть за відсутності розділів, забезпечуючи таким чином більш повне зображення потенційних компромісів узгодженості для розподілених систем. Під затримками слід вважати час, за який клієнт отримає відповідь при встановленому рівні узгодженості.

1.3. Аналіз розвитку та проблем архітектур систем зберігання великих даних

1.3.1. Клієнт-серверна архітектура

Клієнт-серверна модель – це розподілена архітектура, яка відокремлює постачальників інформаційно-обчислювальних ресурсів, тобто серверів обробки та збереження інформації, від ініціаторів запитів на обробку – клієнтів (див. рис. 1.1) [20].

Рисунок 1.1. Клієнт-серверна архітектура

Доступ до даних у клієнт-серверній архітектурі забезпечується наступним чином: 1) клієнт формує та надсилає запит до сервера бази даних; 2) сервер обробляє запит і отримує необхідну інформацію з бази даних, а у разі необхідності – виконує маніпуляції з інформацією, що зберігаються у базі даних; 3) після завершення обробки запиту сервер формує відповідь та надсилає її клієнту. Таким чином, ресурси клієнтського комп'ютера не приймають участь у фізичному виконанні запиту [21].

1.3.2. Архітектура централізованого реплікованого кластеру

Традиційна клієнт-серверна архітектура з одним сервером має певні обмеження, які пов'язані з доступністю та продуктивністю. Так, у разі відмови сервера всі клієнти втрачають доступ до інформації. Крім того, існує певна межа продуктивності, після досягнення якої сервер не може впоратися з обслуговуванням нових користувачів, або ж значно зростає час обслуговування

запитів. Вирішенням перелічених вище недоліків може бути реплікація даних поміж декількох серверів. Це дозволяє розподілити зростаюче навантаження між серверами-репліками, а також забезпечити стійкість до відмов серверів, тобто підвищити готовність системи.

Таким чином, наступним етапом розвитку розподілених систем була архітектура реплікованого клієнт-серверного кластеру з репліками, розташованими у межах локальної мережі (див. рис. 1.2. та 1.3.) [22]. Передумовами виникнення такої архітектури була необхідність підвищення надійності, доступності та продуктивності.

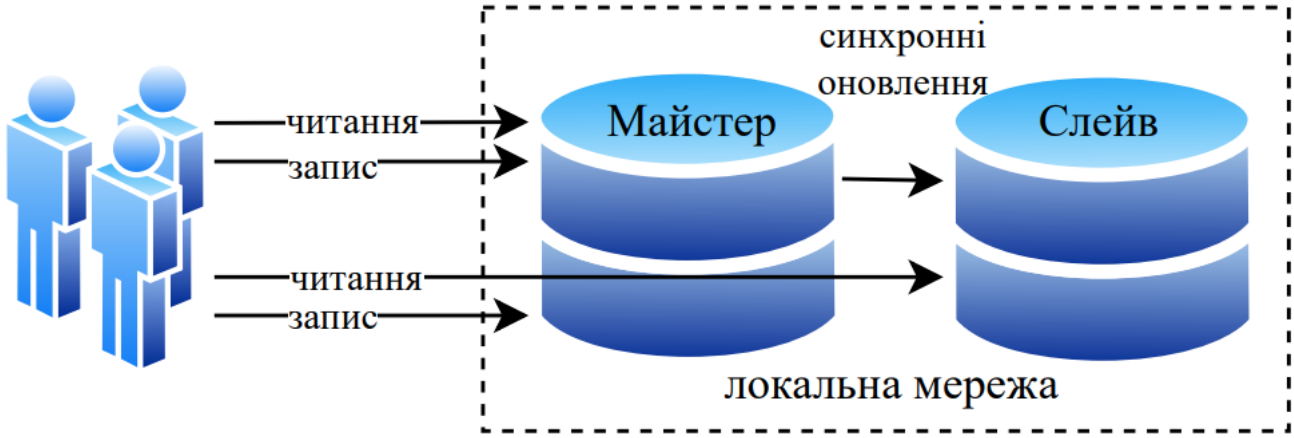


Рисунок 1.2. Архітектура централізованого кластеру майстер-слейв

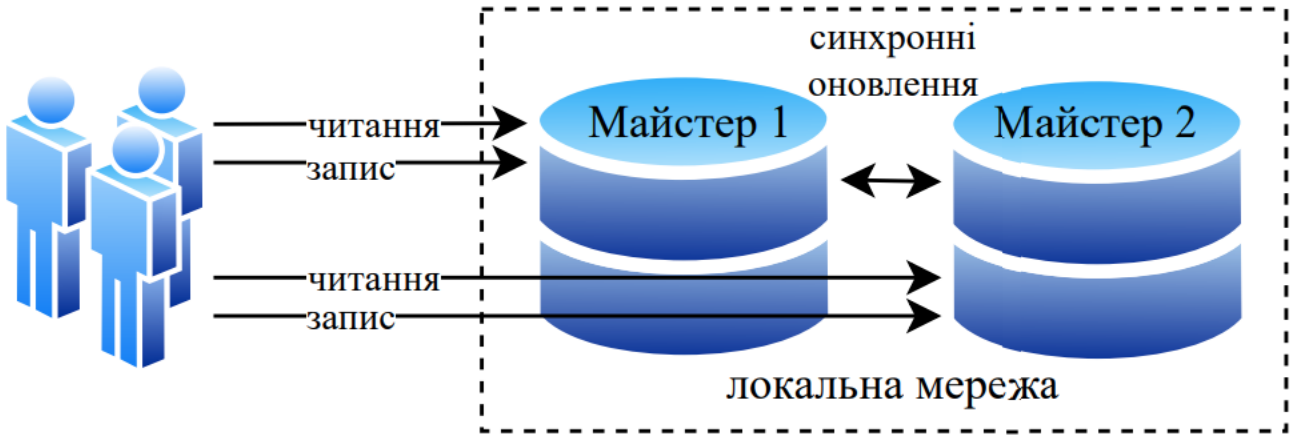


Рисунок 1.3. Архітектура централізованого кластеру майстер-майстер

Існують два різновиди реплікованого кластеру: майстер-слейв (master-slave) та майстер-майстер (master-master або multi-master). У першому випадку (рис. 1.2.) всі репліки можуть обробляти запити на читання інформації, проте будь-яка операція,

яка вимагає оновлення даних, повинна проходити та оброблятися майстер-сервером, який розповсюджує оновлення для всіх підлеглих серверів.

Архітектура майстер-слейв досить проста у реалізації, але до її недоліків можна віднести наявність єдиної точки відмови (майстер-сервера) та проблеми продуктивності при зростанні кількості операцій запису [23]. З цієї причини реплікація майстер-слейв переважно використовується в інтенсивних для читання додатках. Якщо майстер виходить з ладу, один із підлеглих серверів може прийняти на себе роль майстра. Однак на час вибору нового майстра, кластер не може обслуговувати запити на оновлення інформації.

Наступним етапом розвитку баз даних стала поява архітектури кластеру майстер-майстер (рис. 1.3.), яка є більш складною у реалізації, але вирішує недоліки попередньої схеми. Однак, можливість оновлення бази даних декількома серверами одночасно викликає складність у підтримки узгодженості даних. Конфлікт може виникнути, якщо кілька серверів намагаються одночасно оновити один і той самий об’єкт у базі даних. Для виявлення та вирішення таких конфліктів використовуються спеціальні алгоритми та протоколи [24].

Треба також зазначити, що у централізованому кластері всі репліки поряд розміщені у локальній мережі, а синхронізація між майстром та іншими репліками є синхронною. Тобто, операція запису чи оновлення даних вважається успішною тільки після завершення оновлення інформації на всіх серверах-репліках. Також, всі репліки призупиняють обслуговування запитів до закінчення синхронізації даних між ними використовуючи механізм двофазних транзакцій (two-phase commit protocol). Оскільки, всі репліки об'єднані локальною мережею, синхронізація виконується дуже швидко і блокування роботи системи не займає багато часу.

1.3.3. Архітектура глобального-розподіленого реплікованого кластеру

Архітектура централізованого реплікованого кластеру ефективно обслуговує клієнтів, що розташовані географічно поруч зі серверами. Однак, при обслуговуванні географічно віддалених клієнтів виникають суттєві затримки, пов'язані зі зростанням часу передачі результатів обробки через глобальну мережу Інтернет. Також централізований кластер не захищений від відмов за загальними причинами, тобто не забезпечує катастрофостійкості.

Таким чином, для обслуговування географічно-віддалених користувачів виникла необхідність у побудові глобально-розподілених інформаційних систем за архітектурою майстер-майстер в яких репліки розташовані близько до груп користувачів, які вони обслуговують, а не одна до одної (рис. 1.4).

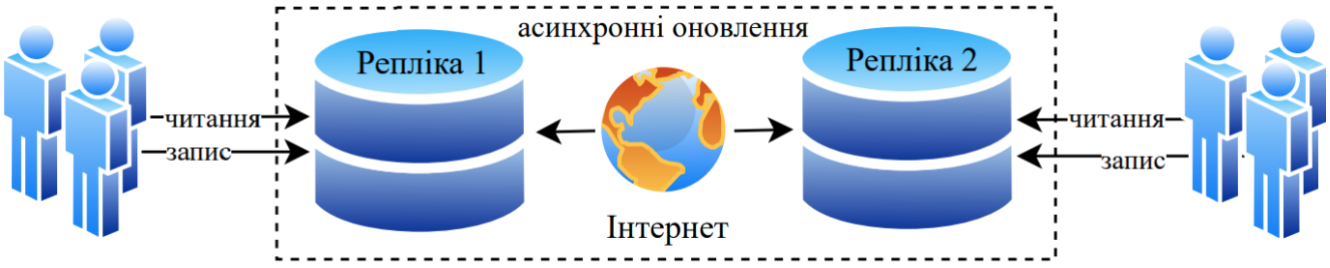


Рисунок 1.4. Архітектура глобально-розподіленого реплікованого кластеру

Однак, така архітектура фактично унеможливлює виконання синхронних оновлень між репліками, оскільки реалізація протоколу двофазних транзакцій між глобально-розподіленими репліками призводить до катастрофічного зниження продуктивності усієї системи. Тому, у таких системах використовується механізм асинхронних реплікацій у якому зміни інформації в одній репліці розповсюджуються на інші репліки у фоновому режимі через деякий час. Таким чином, асинхронна реплікація призводить до появи тимчасової неузгодженості даних між вузлами-репліками. Фактично, це обумовлює перехід від моделі ACID зі сильною узгодженістю даних, до моделі BASE, яка забезпечує тільки кінцеву узгодженість (Eventual Consistency). Як наслідок, деякі клієнти можуть отримати

застарілі дані. Безумовно, у деяких додатках послаблення вимог до узгодженості може спричинити катастрофічні наслідки, що є неприйнятним. Однак для систем, які можуть собі дозволити послаблення узгодженості, наприклад, у соціальних мережах, впровадження асинхронної реплікації дозволяє досягти високої продуктивності та зробити систему високо-масштабованою [24].

1.4. Аналіз та класифікація розподілених нереляційних баз даних

Акронім NoSQL є поняттям, яке об’єднує альтернативні варіанти систем зберігання на відмінну від традиційних реляційних SQL баз даних. Особливістю цих систем є модель даних, які мають різні структури на противагу класичній моделі рядків та стовбців, що використовуються в реляційних системах керування базами даних. Враховуючі модель даних NoSQL можна виділити наступні види [25]:

– документо-орієнтована система керування базами даних (document database) зберігає дані у вигляді JSON, BSON або XML об’єктів, що дає змогу переробляти структуру об’єктів в залежності від вимог;

– система керування базами даних «ключ-значення» (key-value store) є найпростішим варіантом NoSQL баз даних та є доволі схожою на реляційну базу даних, яка має тільки два стовбця: ключ та значення;

– стовбцево-орієнтована система керування базами даних (column-oriented database) організована у вигляді набору стовбців, на відміну від реляційних баз даних, які зберігають дані у рядках і читає рядок за рядком. Це дає змогу не чіпати не бажані дані і прочитати необхідний набір стовбців;

– графово-орієнтована система керування базами даних (graph databases) організована безпосередньо на зв’язках між елементами даних, де кожен елемент зберігається як вузол, а зв’язки між вузлами – відносинами.

У таблиці 1.1 представлені приклади NoSQL баз даних з огляду на модель даних [26].

Таблиця 1.1 – Моделі даних у NoSQL базах даних

Модель даних	Представники
Ключ-значення	Redis, Oracle NoSQL, Amazon DynamoDB
Документ-орієнтовані	MongoDB, Couchbase, Amazon SimpleDB
Стовбцево-орієнтовані	Cassandra, Google Bigtable, HBase
Графо-орієнтовані	Neo4j, Amazon Neptune, FlockBD, OrientDB

1.4.1. Apache Cassandra

Apache Cassandra – це NoSQL база даних з відкритим кодом, яка була спочатку розроблена за допомогою мови програмування Java компанією Facebook (зараз Meta) у 2008 році і вже в наступному році була передана фонду Apache Software Foundation. В основі Cassandra лежить гібридна модель даних, а саме стовбцево-орієнтована з концепцією «ключ-значення» [27].

Cassandra має кільцеву архітектуру на основі майстер-майстер реплікації, яку можна розгорнути у двох та більше центрів обробки даних (див. рис. 1.5). Cassandra досягає горизонтальної масштабованості, розділяючи всі дані, що існують у системі за допомогою хеш функції.

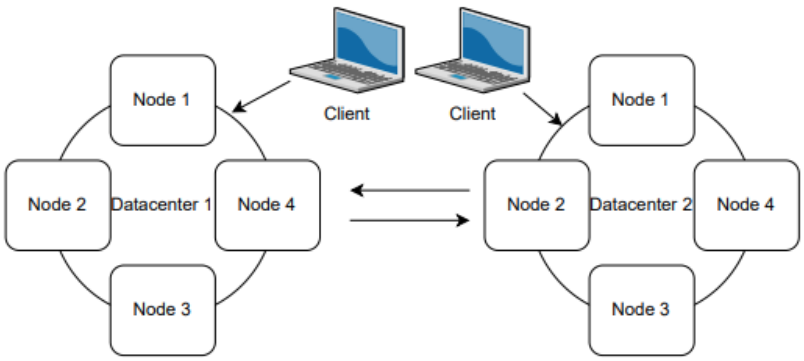


Рисунок 1.5 – Кільцева архітектура Cassandra кластеру

Cassandra має дві різні стратегії реплікації, які визначають фізичні вузли, як репліки для заданого діапазону маркерів: NetworkTopologyStrategy, SimpleStrategy. Кожен ключовий простір даних має свою власну стратегію реплікації. Стратегія NetworkTopologyStrategy вимагає певного коефіцієнта реплікації для кожного

центру обробки даних у кластері. Навіть якщо кластер використовує лише один центр обробки даних, NetworkTopologyStrategy рекомендується замість SimpleStrategy, щоб полегшити додавання нових фізичних або віртуальних центрів обробки даних до кластера пізніше, якщо це необхідно. У стратегії SimpleStrategy схема розподілу встановлює перший вузол, на якому зберігається репліка. SimpleStrategy однаково обробляє всі вузли, ігноруючи будь-які налаштовані датацентра або серверної стійки (rack) [27].

Інформація про працездатність у кластері Cassandra поширюється розподіленим способом через механізм виявлення збоїв на основі протоколу gossip. Gossip – це те, як Cassandra поширює основну інформацію про завантаження кластера, таку як приналежність до кластеру та версії мережевих протоколів між вузлами. Вузли gossip збирають стан інших вузлів, як безпосередньо до інших вузлів gossip, так і від них.

1.4.2. MongoDB

MongoDB– це NoSQL база даних, яка була розроблена у 2007 році, командою розробників DocubleClick (наразі Google). В основі MongoDB лежить модель даних «ключ-документ», яка зберігає документи у форматі JSON. Пошук даних у базі виконуються за допомогою ключа або значенням відповідного поля [27].

MongoDB має ієрархічну архітектуру, яка побудована на основі одного або декількох «mongos» (маршрутизаторів запитів) і одного або декількох шардів, які запускають «mongod». Кожен набір реплік має основний і декілька залежних вузлів, які утворюють майстер-слейв реплікацію (див. рис. 1.6).

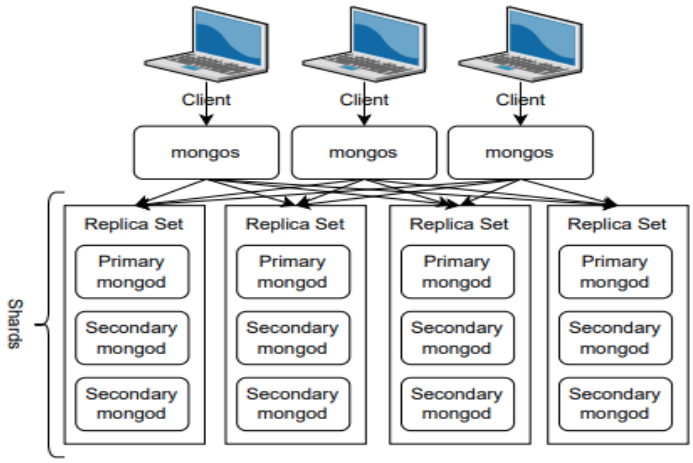


Рисунок 1.6 – Топології MongoDB

MongoDB використовує набір реплік, які є групою «mongod» екземплярів, які оперують з однаковим набором даних. Основний вузол приймає всі операції запису та записує всі зміни у журнал операцій oplog. Залежні сервери реплікують oplog основного вузла та застосовують зазначені операції до свого набору даних для узгодженості між основним і залежними вузлами. [27]

Процес відмови та відновлення у MongoDB базується на використанні допоміжних запитів перевірки працездатності, які виконуються кожні дві секунди між учасниками. Якщо один із учасників кластеру не відповідає протягом десяти секунд, то цей вузол позначається, як непрацездатний. У випадку відмови основного вузла виконується процедура вибору одного із залежних вузлів, який стане основним та почне приймати запити на запис.

1.4.3. Azure CosmosDB

Azure CosmosDB– це NoSQL база даних, яка була розроблена корпорацією Microsoft у 2017 році. Ця база даних є мультимодельною і в залежності від обраного API буде використовуватися модель, яка є сумісною. Для CosmosBD можна обрати: Cassandara API, MongoDB API, Gremlin API, Table API [28].

База даних CosmosDB представляє PaaS (Platform-as-a-service) рішення та тільки може бути розгорнута у хмарному середовищі Microsoft Azure. CosmosDB може використовувати архітектуру майстер-слейв або майстер-майстер (з 2018 році

підтримка операції запису для різних географічних локацій), репліки якої знаходяться у центрах обробки інформації Microsoft Azure, які мають різну географічну локацію [28].

Процес реплікації у CosmosDB при мінімальних налаштуваннях має визначений шлях реплікації. Цей шлях вказує на пару «ключ-значення» у кожному елементі бази даних. Потім значення ключа розділу використовується для розподілу даних у межах регіону. Кожен контейнер в Azure CosmosDB розповсюджує дані, використовуючи значення ключа розділу, на різні фізичні розділи в одному регіоні. Але фізичний розділ насправді не є окремою фізичною машиною чи пристроєм. Фактично, реалізація фізичного розділу є набір реплік. Набір реплік – це група обчислювальних ресурсів, які можуть динамічно збільшуватися та зменшуватися відповідно до потреб бази даних. Кожен набір реплік матиме інші географічно віддалені набори реплік, які керують тими ж ключами розділів, якщо дані розподіляються глобально [28].

1.5. Аналіз та класифікація моделей узгодженості

Моделі узгодженості даних визначають гарантії, що надаються сховищем даних користувачам, щодо результатів операцій запису та читання при наявності конкуруючих запитів від інших користувачів.

Існує багато різних моделей узгодженості, які використовуються у розподілених обчислювальних системах і системах зберігання даних, які узагальнені на рис. 1.7. Умовно їх можна поділити на дві основні групи [29]. Перша група забезпечує певний стан сховища даних для всіх користувачів (моделі орієнтовані на дані); друга надає гарантії лише для окремого користувача (моделі орієнтовані на користувача) у той час як інформація, що повертається у відповідь на той же самий запит читання, отриманий одночасно від декількох користувачів, може відрізнятися від користувача до користувача [10].

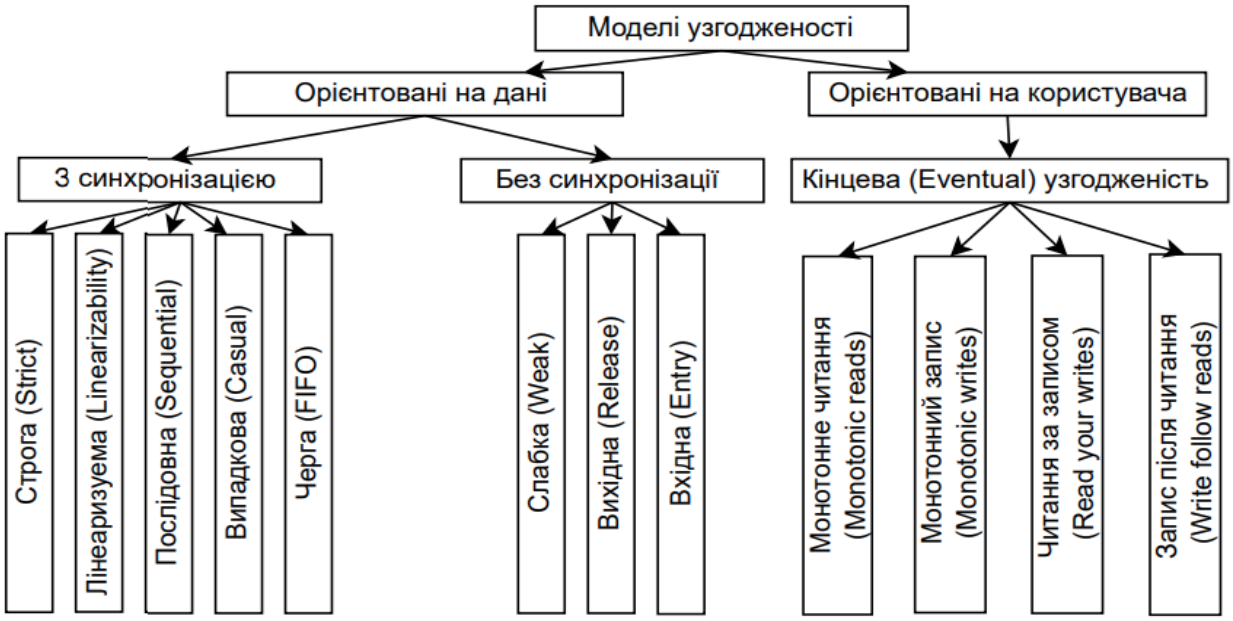


Рисунок 1.7 – Моделі узгодженості даних

Сильна узгодженість не може бути ефективно досягнута у реплікованих глобально-розподілених системах. Таким чином, багато сучасних нереляційних сховищ даних реалізують різні види моделей послабленої узгодженості.

На жаль, на сьогодні не існує загальних правил для послаблення узгодженості і, виходячи з цього, різні постачальники баз даних реалізують різні моделі узгодженості, які несумісні між собою та які важко порівняти.

Наприклад, Apache Cassandra, яка підтримує архітектуру майстер-майстер, визначає дискретний набір рівнів узгодженості, вказуючи кількість реплік, що викликаються для виконання кожної операції читання та запису (див. табл. 1.2). Сильна узгодженість, яка гарантує, що читання завжди відображатиме найбільш актуальні дані, досягається коли сума вузлів, що викликаються при операціях запису та читання є більшою за коефіцієнт реплікації даних [10].

Таблиця 1.2 – Модель узгодженості Apache Cassandra

Рівень узгодженості	Визначення та гарантії узгодженості
ONE	Дані мають бути записані до журналу транзакцій та у пам’ять принаймні однієї репліки перш ніж підтвердити успішність операції запису клієнту; під час зчитування даних Cassandra запитує та повертає відповідь з однієї репліки (використовується найближча репліка з найменшою затримкою); сильна узгодженість забезпечується, коли сума реплік, що викликаються для читання та запису даних більша за коефіцієнт реплікації.
TWO	Дані повинні бути записані принаймні на дві репліки; операція читання повертає найбільш актуальний запис із двох найближчих реплік (найбільш актуальні дані визначаються шляхом порівняння часових міток записів, повернутих цими двома репліками).
THREE	Подібно до TWO але для трьох реплік.
QUORUM	Кворум (тобто більшість) вузлів-реплік повинен підтвердити запис або повернути відповідь на запит читання.
ALL	Дані повинні бути записані на всі вузли репліки в кластері перед підтвердженням успішності операції; запит на читання завжди повертає найбільш актуальні дані після того, як отримання відповіді від усіх реплік; операція читання/запису не буде вважитися успішною, якщо навіть одна репліка не відповість на запит.
EACH_QUORUM, LOCAL_QUORUM, LOCAL_ONE	Додаткові рівні узгодженості, які використовуються, якщо репліки кластера Cassandra розподілені по декількох центрах обробки даних.

Модель узгодженості MongoDB, що є системою майстер-слейв, заснована на налаштуванні параметрів w та j, як показано у табл. 1.3. Оновлення даних з первинної репліки (майстра) на вторинні (слейви) завжди виконуються асинхронно. Операція читання може виконуватися з майстра або зі слейва. Якщо операція читання виконується зі вторинних реплік то MongoDB автоматично стає системою з кінцевою (eventual) узгодженістю [10].

Таблиця 1.3 – Модель узгодженості MongoDB

Рівень узгодженості	Визначення та гарантії узгодженості
w:0, j:false	Найслабший рівень узгодженості (записи можуть бути втрачені навіть без розділу системи), що забезпечує найменшу затримку читання/запису.
w:1, j:false	Гарантовано запис на диск первинної репліки; це забезпечує досить низьку затримку, але й дуже слабку узгодженість.
w:2, j:false	Гарантовано запис на диск первинної репліки та в пам'ять однієї з вторинних реплік; це забезпечує низьку затримку та слабку узгодженість.
w:2, j:true	Гарантовано запис на дисках первинної репліки та однієї з вторинних реплік; це забезпечує середню затримку та узгодженість.
w:majority, j:false	Гарантовано запис на дисках первинної репліки та в пам’ять більшості вторинних реплік; це забезпечує середню затримку та узгодженість.
w:majority, j:true	Гарантовано запис на дисках первинної репліки та більшості вторинних реплік; це забезпечує високу узгодженість даних, але ж тягне й високі часові затримки на виконання операцій читання та запису.

Azure CosmosDB – це хмарна система зберігання даних, репліки якої можна розмістити по всьому світу у кількох регіонах Azure. CosmosDB підтримує п’ять рівнів узгодженості, характеристика яких наведена у табл. 1.4, та може бути налаштована для роботи з однією або кількома майстер-репліками.

Таблиця 1.4 – Модель узгодженості Azure CosmosDB

Рівень узгодженості	Визначення та гарантії узгодженості
STRONG (Reads: local minority; Writes: global majority)	Сильна узгодженість забезпечує гарантії лінеаризованості, тобто, наприклад, читання гарантовано повертає найбільш актуальні результати запису/оновлення даних.
BOUNDED STALENESS (Reads: local minority: Writes: local majority)	Гарантується, що читання завжди повертатиме узгоджені результати оновлення даних. Однак, допускається, що читання може відставати від запису, але не більше ніж на K оновлень або на T часовий інтервал (тобто вікно нестачі (staleness window)), залежно від того, що менше.
SESSION (Reads: single replica with session token; Writes: local majority)	Гарантується, що читання завжди повертатиме узгоджені результати оновлення даних протягом однієї клієнтської сесії; для інших клієнтів також гарантоване монотонне читання та запис, запис після читання (write-follows-reads) та читання свого запису (read-your-writes).
CONSISTENT PREFIX (Reads: single replica; Writes: local majority)	Гарантується, що читання завжди повертатиме угоджені результати оновлення даних без пробілів; наприклад, гарантується дотримання впорядкованої послідовності оновлень даних (починаючи з першої), яка зберігалася в майстер-репліці в якийсь час у минулому; натомість, допускається, що найбільш актуальні дані можуть бути відсутніми.
EVENTUAL (Reads: Single replica; Writes: local majority)	Немає гарантії порядку читання; за відсутності подальшого запису репліки зрештою сходяться; користувачі можуть читати дані, які «старші», ніж ті, що були прочитані раніше.

1.6. Модель компромісів реплікованих розподілених сховищ даних CAP/L

Властивості глобально-розподілених систем, що також мають відношення до кібербезпеки: узгодженість, готовність та затримка – тісно пов’язані. Готовність можна інтерпретувати, як ймовірність того, що кожен запит врешті-решт отримає відповідь. Однак у реальних системах надто пізня відповідь (тобто відповідь отримана за межами тайм-ауту системи) розглядається, як відмова, що порушує готовність системи.

В той час висока затримка має небажаний ефект для багатьох інтерактивних застосунків. Звіт [30] показує, що ймовірність того, що клієнт використовуватиме веб-застосунок значно зменшується, якщо час відгуку системи збільшується навіть на 100 мс.

Система або її вузол (репліка) можуть вважатися недоступними, якщо фактичний час відповіді перевищує час очікування програми, тобто розділ можна розглядати, як обмежений час відповіді вузла (репліки). Таким чином, з точки зору CAP, повільне мережеве з'єднання або вузол (репліка), що повільно відповідає (наприклад, через велику кількість запитів), може призвести до рішення про те, що систему розділено. Сьогодні архітектори систем управління розподіленими базами даних і великих веб-застосунків, таких як Twitter, Instagram, Facebook тощо, часто віддають перевагу слабкій гарантії узгодженості, запроваджуючи асинхронні оновлення даних на користь високої доступності системи та малого часу відгуку. Але найбільш перспективним підходом є спроба збалансувати ці властивості щодо бажаної затримки та необхідного рівня узгодженості.

Запропонована модель CAP/L є інтерпретацією та узагальненням теорем CAP і PACELC та відображає кількісний зв'язок та компроміси властивостей глобально-розподілених реплікованих систем зберігання даних (див. рис. 1.9). У рамках цієї моделі узгодженість визначається кількістю вузлів (реплік), які запитуються одночасно, щоб повернути визначений (узгоджений) результат до клієнтської програми. Зокрема, можливі такі рівні узгодженості [10]:

– ONE (еквівалент гарячого резервування). Відповідь з однієї репліки пересилається клієнту. Це найслабший рівень узгодженості, хоча він гарантує мінімальну затримку. У цей час є кілька варіантів цього рівня узгодженості: ONE-ONE – запитується однин вузол (репліка) (зазвичай найближча репліка з найменшою затримкою мережі), відповідь на яку повертається користувачеві; ONE-ALL – усі вузли (репліки) запитуються одночасно і користувачеві повертається найшвидша відповідь без очікування відповідей інших реплік, які ігноруються; ONE-QUARUM – запитується кворум вузлів (реплік) і користувачеві повертається найшвидша відповідь;

– ALL (еквівалентно N-модульній надлишковості) – система повинна чекати, поки всі вузли (репліки) не повернуть свої відповіді. У цьому випадку час відповіді обмежується найповільнішою реплікою, хоча забезпечується сильна узгодженість;

– QUORUM (еквівалент системи двох із трьох або більшості) – система має чекати відповідей від кворуму вузлів (реплік). Він забезпечує компроміс між ONE і ALL рівнями, що змінюють затримку та узгодженість.

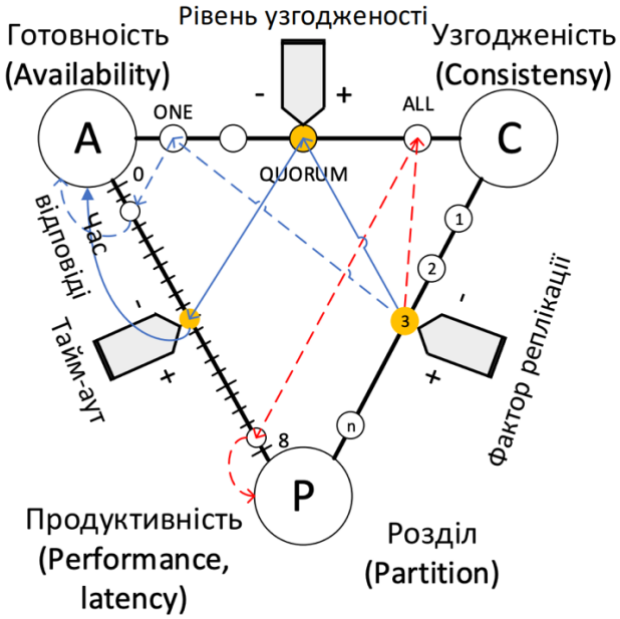


Рисунок 1.9 – Компроміси CAP/PACELC

Якщо найвищим пріоритетом є підвищення готовності та мінімізація часових затримок, користувачам слід розглянути менший рівень узгодженості, наприклад ONE та його варіації. Рівень ONE-ALL може забезпечити швидшу відповідь (швидше, ніж ONE-ONE і ONE-QUORUM) за вищих обчислювальних витрат і накладних витрат на передачу даних.

Якщо найвищим пріоритетом є узгодженість та зниження ризиків порушення цілісності (у випадку неузгодженості даних), користувачі повинні вибрати сильніший параметр узгодженості, який, однак, погіршує затримку системи. Система завжди забезпечує сильну узгодженість, гарантуючи, що читання завжди відображатиме останній запис, зберігаючи таке правило:

$$(N_{\text{запис}} + N_{\text{читання}}) > \text{фактор реплікації} \qquad (1.1)$$

Наприклад, рівень узгодженості QUORUM, який використовується, як для операцій запису, так і для операцій читання, завжди забезпечує сильну узгодженість. Сильна узгодженість читання забезпечується, коли рівень узгодженості ALL використовується для читання даних, тоді як рівень узгодженості ONE використовується для запису, і навпаки. У першому випадку перевага віддається мінімізації затримки операції запису, тоді як у другому випадку забезпечується мінімальна затримка читання.

Встановлений у системі час очікування (тайм-аут) розглядається, як межа між доступністю системи та продуктивністю (в термінах затримки/часу відповіді) [31]. Крім того, якщо репліка не відповідає до вказаного часу очікування, система вважається розділеною, що також впливає на цілісність. Таким чином, розробники систем повинні мати можливість налаштувати тайм-аути відповідно до бажаного часу відповіді системи, також маючи на увазі компроміси між цілісністю (узгодженістю), готовністю та швидкодією.

1.7. Аналіз моделі кібербезпеки глобально-розподілених реплікованих систем зберігання даних

Кібербезпека [32]– це стан захищеності комп’ютерних систем, електронних даних і мереж їх передачі від несанкціонованого доступу, та інших дій, пов’язаних з маніпулюванням, блокуванням, псуванням, руйнуванням і знищенням умисного і випадкового характеру. Модель кібербезпеки складається з готовності, цілісності та конфіденційності [33].

1.7.1. Аналіз властивості конфіденційності ГРРС

На сьогоднішній день, системи зберігання даних повинні захищати дані від несанкціонованого доступу до них, тобто протидіяти порушенням конфіденційності. Конфіденційність [32] – це властивість захищеності системи від неавторизованого доступу до інформації та спроб розкриття (отримання змісту) неавторизованими користувачами та/або процесами. Тобто, це означає, що лише

авторизовані користувачі повинні мати змогу виконувати операції, які їм дозволено виконувати у базі даних. Повноваження мають бути визначені таким чином, щоб різні користувачі мали різні права на ті самі відношення у базі даних.

Для глобально-розподілених реплікованих систем зберігання даних застосовуються різні моделі безпеки доступу, які розглянуті у [34, 35, 36, 37]. Модель Role Based Access Control (RBAC) визначає ролі відповідно до завдань та обов'язків користувачів, а авторизація доступу відповідно до ролей. Традиційна модель доступу має дві категорії: «обов'язковий контроль доступу» та «дискреційний контроль доступу». У моделі «обов'язкового контролю доступу» доступ користувачів до ресурсів контролюється відповідно до певних правил, які попередньо визначені. У моделі «дискреційного контролю доступу» користувачі можуть надавати повноваження доступу іншим користувачам у межах обмежень, призначених для них. [38]

Глобально-розподілені репліковані системи зберігання даних також широко використовуються криптографічні примітиви для забезпечення конфіденційності інформації, які розглянуті у [39, 40, 41, 42, 43]. Здебільшого, ГРРС використовують загальнодоступні канали зв'язку, які можуть прослуховуватись. Для захисту даних, які передаються через незахищену мережу, використовують шифрування. Також дані, які зберігаються на диску теж потребують використання шифрування, щоб протидіяти загрозам, наприклад, викрадення диску або несанкціонованого доступу до резервних копій.

Слід відзначити, що методи та засоби забезпечення конфіденційності досить розвинені у теоретичному та практичному сенсі та реалізовані у багатьох мережевих протоколах і широко інтегровані в існуючі розподілені систем та бази даних.

1.7.2. Аналіз властивості цілісності ГРРС

Забезпечення кібербезпеки глобально-розподілених систем зберігання даних без урахування аспектів забезпечення цілісності даних. Цілісність [32] – це

властивість, яка полягає в тому, що інформація практично не може бути змінена випадково чи навмисне неавторизованими суб'єктами (порушниками) чи об'єктами (процесами). У посібнику [44] зазначені загрози порушення цілісності, наприклад, несанкціоноване модифікування даних. Також порушення цілісності не обмежуються навмисними атаками. Це можуть бути помилки користувача системи, неправильне налаштування системи тощо. Автори посібника з кібербезпеки [44] визначили, що цілісність можна досліджувати з трьох різних сторін: запобігання неавторизованим суб'єктам від внесення змін, запобігання авторизованим суб'єктам від внесення змін та запобігання авторизованим суб'єктам від внесення неавторизованих модифікацій (помилок) і підтримка узгодженості. Коли механізм безпеки забезпечує цілісність, він забезпечує високий рівень впевненості у тому, що дані, об'єкти та ресурси не будуть неправомірно змінені. У той же час слід враховувати, що для контролю цілісності потрібні додаткові ресурси: пам'ять та час [45, 46]. Це досить добре простежується у глобально-розподілених системах зберігання даних. Оскільки, неузгодженість між вузлами-репліками породжує додаткову загрозу властивості цілісності ГРРС, а забезпечення сильної узгодженості підвищує час обслуговування і як наслідок знижує рівень готовності системи до її стійкості до атак типу відмова в обслуговуванні (DDoS).

Методи синхронної та асинхронної реплікації розглянуті у [47, 48, 49, 50, 51] наукових працях. Синхронна реплікація застосовує механізм двофазних транзакцій. Особливість цього механізму є те що при оновленні даних, виконується блокування ресурсів, що породжує великі затримки, особливо для ГРРС. При застосуванні асинхронних оновлень, спостерігається явище неузгодженості системи у певний проміжок часу, поки система не розповсюдить оновлення.

У науковій праці [52], автори пропонують використовувати надлишковість для підвищення готовності, цілісності та впливу конфіденційності операційних систем. Подібний підхід може бути пропонується застосовувано і для розподілених систем зберігання даних у якому механізм реплікації надає природну надмірність.

1.7.3. Аналіз властивості готовності ГРРС

Готовність [32] – це властивість, яка полягає в тому, що авторизований користувач і/або процес, наділений відповідними повноваженнями, може використовувати ресурс згідно відповідним правилам. Для клієнт-серверних систем готовність визначається, як властивість забезпечення відповіді клієнту до встановленого часу очікування (тайм-ауту) в заданих умовах у будь-який час [4]. Для глобально-розподілених систем додається фактор часу, тобто відповідь повинен бути отриманий в рамках часу очікування, який встановлюється там-аутом [53, 54].

Глобально-розподілені системи, що використовують глобальну мережу Інтернет, підпорядковуються законам розподілу з повільно спадним хвостом. Це означає, що найгірший час виконання може бути набагато більшим від середнього значення часу обробки запиту у системі. Це може бути зумовлене періодичними перевантаженнями у глобальній мережі Інтернет, які призводять до збільшення часу обробки запитів у системі. У подібних ситуаціях це призводить до зменшення продуктивності системи та неефективній роботі механізмів відмовостійкості, що можуть впливати на рішення про розділ системи.

Неможливість отримати відповіді від деяких вузлів протягом зазначеного часу очікування спричиняє розділ системи зберігання даних. З огляду на це, розділену систему можна розглядати, як обмеження часу відгуку вузла. Повільне мережеве з’єднання або вузол, який повільно реагує, чи неправильні налаштування тайм-ауту можуть призвести до помилкового рішення про те, що систему було розділено. Коли систем виявляє стан розділу, вона має вирішити, чи повернути клієнту неузгоджену відповідь, чи повернути помилку, що підриває доступність системи [55]. Для глобально-розподілених систем значення тайм-ауту впливає на рішення про розділ системи і в подальшому на продуктивність. Для розподілених систем значення тайм-ауту рахується, як подвійне значення найгіршого часу обробки запиту до системи (worst execution time), наприклад протокол TCP або набагато більше значення ніж це, наприклад Cassandra, яка за замовчуванням має тайм-аут

для запитів на запис 2 секунди і для запитів на читання 5 секунд. Репліковані відмовостійка система переходить у стан розділеної, коли одна з її частин не відповідає через відмову каналу зв'язку, затримку або збій вузла, що призводить до тайм-ауту. Тайм-аут глобально-розподіленої реплікованої системи зберігання даних є фундаментальною частиною усіх розподілених механізмів відмовостійкості, що працюють через глобальну мережу Інтернет та використовується, як основний механізм виявлення помилок. Також тайм-аут відіграє важливу роль, як визначальний фактор у взаємодії між готовністю та продуктивністю системи. Тайм-аут – це стан в який переходить система, коли остання намагається отримати деяку іншу інформацію з іншої системи через канал зв'язку, але це не вдається зробити раніше, ніж попередньо визначений допустимий час очікування. Важливо зауважити, що різні параметр тайм-ауту можуть впливати на доступність системи, середній час обслуговування та очікування. Якщо тайм-аут нижчий за типовий час відповіді, система, ймовірно, частіше перебуватиме у стані розділу. Проте, занадто великий тайм-аут не дозволяє вчасно виявити збій системи та ефективно застосувати механізми відмовостійкості [56]. Сучасні глобально-розподілені системи зберігання даних використовують статичне значення тайм-ауту, але перспективніший є підхід застосування динамічного значення тайм-ауту, який змінюється під час роботи системи, що дає змогу збалансувати продуктивність, готовність і підвищити ефективність механізмів відмостійкості. Тому важливо знизити найгірший час обслуговування запитів (worst execution time) для розподілених систем. Таким чином, підвищення швидкодії системи дозволяє підвищити стійкість системи до атак типу DDoS або забезпечити сильний рівень узгодженості для зниження загроз цілісності при такому ж рівні готовності системи.

1.7.4. Зв'язок моделі кібербезпеки з моделлю гарантоздатності

Останні десятиліття вчені приділяють велику увагу до інтегрованої концепції гарантоздатності (dependability) інформаційних систем. Під гарантоздатністю

будемо розуміти можливість надавати послуги, яким можна обґрунтовано довіряти. У рамках цієї інтегрованої концепції гарантоздатності виділяють наступні атрибути [32]:

– готовність (availability) – це властивість ресурсу системи, яка полягає в тому, що користувач (процес) може використовувати ресурс згідно правил та певної якості;

– надійність (reliability) – це властивість незмінності певної поведінки і результатів;

– функційна безпечність (safety) – відсутність катастрофічних наслідків для користувачів та навколишнього середовища;

– цілісність (integrity) – це властивість, яка полягає в тому, що система не може бути змінена неправомірно або випадково;

– ремонтопридатність (maintainability) – це властивість, яка полягає в тому, що система дає можливість модифікації та ремонту.

Також на ряду з атрибутами, які описані вище, виділяють атрибут інформаційної безпеки (security). У класичному вигляді інформаційна безпека опирається на атрибути: конфіденційність, цілісність та доступність. Конфіденційність – це властивість захищеності інформації з наперед заданою якістю від неавторизованого доступу до неї та спроб розкриття неавторизованими користувачами та процесами [32]. Інформаційна безпека розподілених комп’ютерних систем (серверів), особливо розподілених через глобальну мережу Інтернет, зазнає певних змін. Ці зміни відображають реальний розподілений характер зв’язків між компонентами, а також природну надмірність при використанні компонентів – реплік, що зберігають копії інформації. Виходячи з цього, властивість готовності переходить у властивість доступності, а властивість цілісності на рівні вузлів системи розширюється властивістю узгодженості для всієї системи (див. рис. 1.8).

Властивість узгодженості означає, що незважаючи до якої репліки під’єднані клієнти системи у будь-який проміжок часу вони отримують однакову інформацію, а дані записані на один вузол поширюються на інші.

Готовність є важливим аспектом інформаційної безпеки, оскільки вона гарантує, що авторизовані користувачі мають доступ до даних і систем, необхідних для виконання своєї роботи та підтримки діяльності організації. Для розподілених систем існують загрози як вторгнення, перенавантаження системи та відмови мережі. Найчастіше розподілені системи схильні до мережевий відмов або збоїв, за статистикою мережеві відмови, станом на 2020 рік, становлять 12%, а цілеспрямовані кібератаки 11% [57]. Більшість з мережевих відмов становлять атаки типу DDoS, що уражає в першу чергу готовність системи в цілому, особливо для глобально-розподілених реплікованих систем. У той самий час, загроза для готовності системи несе за собою неузгодженості системи, оскільки встановити бажаний рівень узгодженості неможливо.

Розділ (partition) – це розрив взаємодії у розподіленій системі між двома та більше репліками. Ураховуючи ймовірність розділу, глобально-розподіленої система може перебувати у доступному стані і в той же час бути розділеною або бути доступною, але розділеною. У результаті стану системи доступна, але розділена, оновлення не можуть поширюватися по реплікам, що веде за собою ймовірність отримання клієнтом застарілої або спотвореної інформації у випадку компрометації вузла системи.

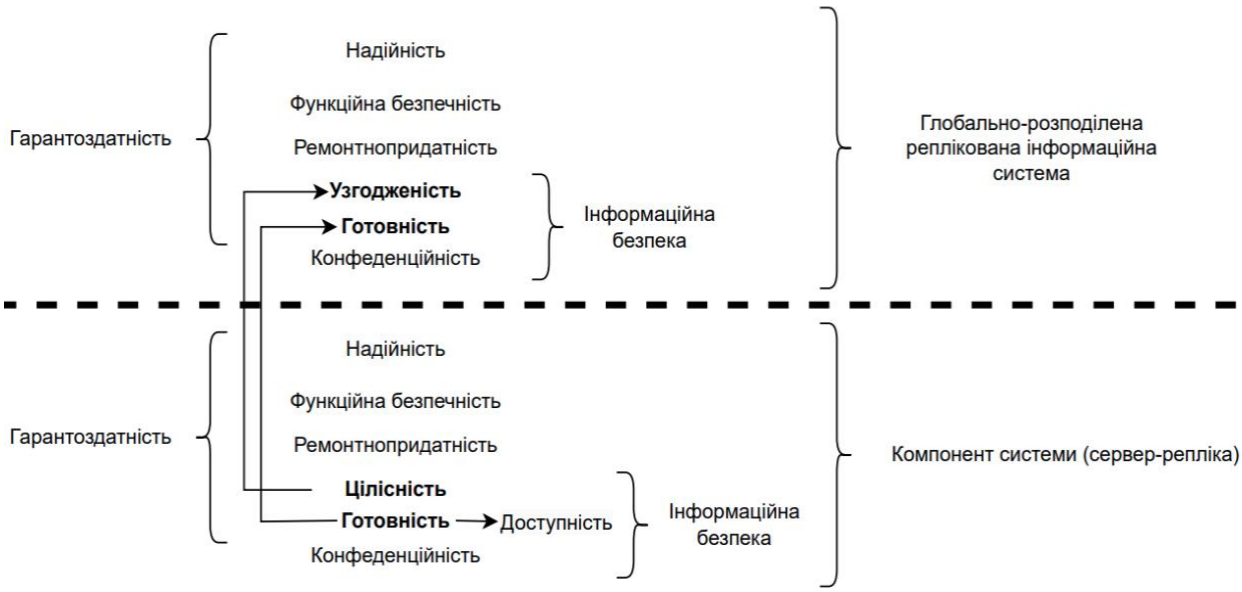


Рисунок 1.8 – Модель гарантозданості та інформаційної безпеки глобально-розподілених реплікованих систем та їх компонентів

Властивість узгодженості може брати початок з слабкої в одній крайності до сильної у іншій та мати проміжні стани, наприклад стан кінцевої або пом’якшеної узгодженості. На практиці не є можливим ефективно досягнути сильної узгодженості для глобально-розподілених реплікованих систем. Таким чином, багато таких систем реалізують різні види кінцевої або послабленої моделей узгодженості. На превеликий жаль, загальних правил щодо послаблення узгодженості не має і тому вендори впроваджують різні моделі узгодженості, які не є сумісними в більшості випадків. Баланс між властивостями: узгодженості, доступності та розділу у таких системах описує теорема CAP і в подальшому PACELC (див. розд. 1.2).

Для систем, які використовують безліч серверів для зберігання даних, розширяється традиційна модель гарантоздатності в частині інформаційної безпеки (ІБ). Властивість цілісності інформації розширяється властивістю узгодженості даних, що зберігаються на різних вузлах. При цьому ключовою особливістю є те, що ці властивості (цілісності та готовності) входять у протиріччя один з одним. Наприклад, для забезпечення сильної узгодженості даних, що повертаються клієнту, необхідно очікувати відповідь від усіх вузлів. Однак, в умовах їх глобального розподілення в Інтернеті, підвищується ймовірність того, що один або кілька вузлів виявляться недоступними, тобто, не повернуть відповідь протягом встановленого часу очікування. У цьому випадку виникає протиріччя між узгодженістю та готовністю – чи повернути клієнту потенційно неузгоджені дані (тобто порушити їх цілість) або відказати йому в обслуговуванні (тобто порушити готовність). Таким чином, у подібних системах питання швидкодії, гарантоздатності та інформаційної безпеки не можуть бути сильної відокремлені один від іншого і повинні розглядатися комплексно.

1.8. Постановка наукового завдання та обґрунтування методики досліджень

1.8.1. Загальне наукове завдання

Результати проведеного аналітичного дослідження теорем, архітектур, методів та моделей глобально-розподілених систем зберігання даних показали, що необхідно розвивати та розробляти методи та засоби кібербезпеки глобально-розподілених реплікованих систем зберігання даних, насамперед готовності та цілісності, приймаючи до уваги фундаментальні протиріччя між властивостями узгодженості, цілісності, готовності та продуктивності цих систем.

Загальним науковим завданням дисертаційного дослідження є розробка та удосконалення методів та засобів підвищення кібербезпеки ГРРС, зокрема цілісності та готовності. Для вирішення загального завдання, необхідно вирішити ряд часткових наукових та прикладних завдань:

– аналіз методів і засобів підвищення кібербезпеки глобально-розподілених систем зберігання даних;

– дослідження та удосконалення моделі загроз для глобально-розподілених систем зберігання даних;

– розробка теоретико-множинної моделі для описання патернів розгортання глобально-розподілених систем зберігання даних;

– розробка та дослідження методу динамічного керування рівнем узгодженості ГРРС для підвищення стійкості до DDoS атак при виключені загроз цілісності через неузгодженість даних;

– розробка та дослідження методу надлишкових читань глобально-розподілених систем зберігання даних для підвищення готовності та цілісності в умовах кіберзагроз порушення даних та відмов в обслуговуванні;

– розробка гібридної імітаційної моделі для оцінки ефективності методу надлишкових читань щодо підвищення готовності.

1.8.2. Вибір математичного апарату та методології дослідження

Методологія дисертаційного дослідження ґрунтується на використанні принципів системного аналізу [58, 59, 60] при постановці та вирішенні задач дисертаційного дослідження і використовується в:

– визначенні етапів вирішення поставлених задач та логічній послідовності їх виконання;

– виборі адекватного математичного апарату, методів досліджень та їх співвідношень із завданнями окремих етапів;

– формальному представленні структури, станів та властивостей ГРРС;

– декомпозиції ГРРС на компоненти та дослідженні їх взаємозв’язків у задачах синтезу, аналізу та підвищення готовності та швидкодії ГРРС;

– встановленні та дослідженні взаємозв’язків між властивостями готовності, цілісності, узгодженості та оперативності ГРРС з урахуванням специфіки створення та експлуатації.

Процес вирішення наукової проблеми декомпозований на кілька етапів. На першому етапі розробляється модель загроз відповідно до специфіки функціонування глобально-розподілених реплікованих систем зберігання даних, включаючи принципи теорії гарантоздатності та кібербезпеки. Визначається взаємозв’язок властивостей ГРРС та моделі кібербезпеки, а також набір методів та моделей для реалізації цих принципів та встановлюються системні зв’язки між ними.

На другому етапі розробляється теоретико-множинна модель опису патернів розгортання ГРРС у хмарному провайдері та метод динамічного керування рівнем узгодженості операцій читання та запису, що дозволяє підвищити швидкодію системи, а через те – її готовність. Надаються оцінки підвищення готовності та стійкості до DDoS атак.

На третьому етапі розробляється метод надлишкових читань на основі ініціювання допоміжних запитів читання для зменшення екстремальних затримок та підвищенню готовності, або підвищення цілісності в умовах загроз порушення

даних. Також пропонується гібридна імітаційна модель для оцінки ефективності методу надлишкових читань.

Завершальним етапом дослідження є розробка комплексу інформаційних технологій для автоматизації і підтримки розроблених методів та моделей.

При вирішенні наукових використовуються методи математичного моделювання [61, 62], теорії ймовірності та математичної статистики [63], теорії систем масового обслуговування [64] при розробці та дослідженні методу динамічного керування рівнем узгодженості для ГРРС, методу надлишкових читань та гібридної імітаційної моделі для оцінки ефективності методу надлишкових читань.

1.8. Висновки до першого розділу

1. У цьому розділі проведено аналіз областей застосування глобально-розподілених систем зберігання великих даних. Проаналізовані теореми та варіанти архітектур ГРРС. Визначено основні властивості глобально-розподілених систем: готовність, узгодженість, стійкість до розділів та затримка. Проаналізовано та виділено недоліки та переваги архітектурних підходів для забезпечення надійності та швидкодії систем.

2. Шляхом аналізу визначено, що сильна узгодженість не може бути ефективно досягнута у реплікованих глобально-розподілених системах. Виходячи з цього запропоновано CAP/L модель компромісів між властивостями розподілених сховищ даних.

3. Проаналізовано модель гарантоздатності для глобально-розподілених систем у контексті інформаційної безпеки. Виявлено, що класичне розуміння властивостей ІБ для ГРРС розширяються за рахунок появи додаткових загроз порушення цілісності через неузгодженість даних або розділ, а також готовності через сильний рівень узгодженості або великі затримки.

4. Існує необхідність в розробці інформаційної технології реалізації методів та моделей забезпечення кібербезпеки реплікованих глобально-розподілених

систем зберігання даних, що дозволить автоматизувати роботу розробників, клієнтів ГРРС.

5. На основі проведеного аналізу сформульована загальна науково-технічна задача дисертаційного дослідження, яке розбивається на ряд часткових завдань: аналіз методів і засобів підвищення кібербезпеки ГРРС, дослідження та удосконалення моделі загроз для ГРРС, розробка теоретико-множинної моделі для описання патернів розгортання ГРРС, розробка та дослідження методу динамічного керування рівнем узгодженості для ГРРС, розробка та дослідження методу надлишкових читань для глобально-розподілених систем зберігання даних, розробка гібридної імітаційної моделі для оцінки ефективності методу надлишкових читань, розробка інформаційної технології для підвищення готовності та швидкодії ГРРС на основі методів та моделей.

РОЗДІЛ 2
МЕТОД ДИНАМІЧНОГО КЕРУВАННЯ РІВНЕМ УЗГОДЖЕНОСТІ ГЛОБАЛЬНО-РОЗПОДІЛЕНИХ СИСТЕМ ЗБЕРІГАННЯ ВЕЛИКИХ ОБСЯГІВ ІНФОРМАЦІЇ ДЛЯ ПІДВИЩЕННЯ ГОТОВНОСТІ

2.1. Модель кіберзагроз глобально-розподілених реплікованих систем зберігання даних

Початковими кроками при загальному аналізі механізмів, методів та процесів – є визначення моделі загроз. Модель загроз [32] – це формальний опис методів та засобів здійснення загроз для інформації в конкретних умовах функціонування автоматизованої інформаційної системи. На даний час найбільш відомий та поширений метод оцінки загроз є STRIDE [65, 66, 67], який є акронімом від **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, **E**levation of Privilege. Метод STRIDE є частиною Microsoft Software Development Lifecycle (SDL) який був сформульований у 1999 році Лореном Конфельдером і Прерітом Гаргом. У таблиці 2.1 показано загрози STRIDE.

Для побудови моделі загроз будемо використовувати загальну модель розподіленої системи, яка складається з двох рівнів. Опис зв’язків між компонентами системи, зображено на рис. 2.1 у вигляді Data Flow Diagram (DFD).

Таблиця 2.1 – Модель STRIDE

Загроза	Порушення	Опис	Ціль
Spoofing	Аутентифікація	Видавати себе за щось чи когось, крім себе	Процеси, зовнішні сутності, люди.
Tampering	Цілісність	Змінити щось на диску чи в мережі.	Сховища даних, потоки даних, процеси.
Repudiation	Невідмова	Стверджувати, що ви чогось не зробили або не несете відповідальності.	Процеси.

Продовження таблиці 2.1

Information Disclosure	Конфіденційність	Розголошення інформації неавторизованим сторонам.	Сховища даних, потоки даних, процеси.
Denial of Service	Готовність	Поглинання ресурсів, необхідних для надання послуг.	Сховища даних, потоки даних, процеси.
Elevation of Privilege	Авторизація	Надання доступу неавторизованим користувачам для виконання різноманітних дій.	Процеси.

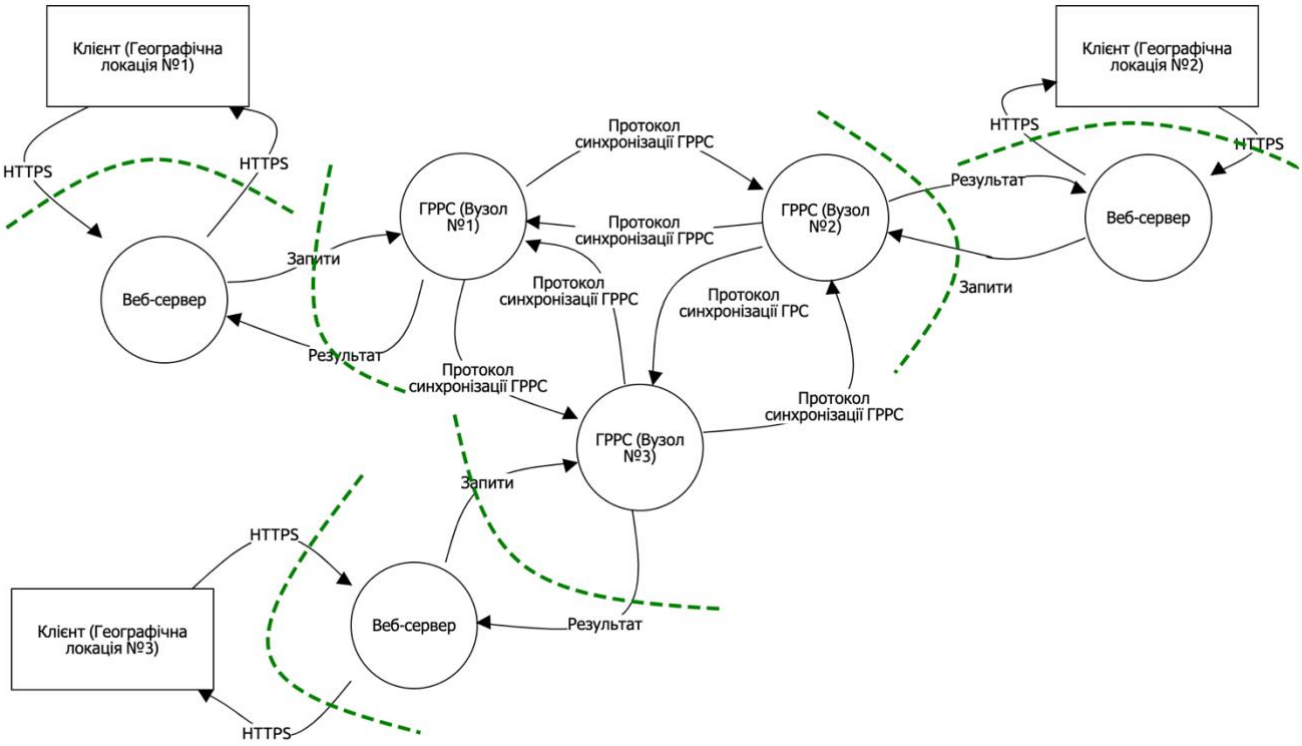


Рисунок 2.1 – DFD діаграма розподіленої системи

На рисунку 2.1 зображено географічно розподілених клієнтів, які знаходяться у різних географічних локаціях. У свою чергу, клієнти взаємодіють зі системою через протокол HTTPS з найближчим до них веб-сервером. Веб-сервер взаємодіє з другим рівнем системи для вибірки необхідної інформації з бази даних.

За результатами аналізу робіт [68, 69, 70, 71, 72, 73, 74] та експертних висновків запропоновано наступний перелік кіберзагроз ГРРС (див. табл. 2.2).

Також додатково виділені загрози, які спрямовані специфічно на вузли (репліки) системи або є загальними для всіх ГРРС.

Таблиця 2.2 – Зіставлення моделі STRIDE зі загрозами ГРРС

Загроза	S	T	R	I	D	E	Тип загрози
Слабкий механізм аутентифікації	Так	Так	Ні	Так	Ні	Так	Специфічна для репліки ГРРС
Слабкі методи авторизації	Так	Так	Ні	Так	Ні	Так	Специфічна для репліки ГРРС
Схильність до ін’єкційних атак	Так	Так	Ні	Так	Ні	Так	Загальна ГРРС
Інсайдерські атаки	Ні	Ні	Так	Ні	Так	Так	Загальна ГРРС
Атака типу «відмова в обслуговуванні» (DDoS)	Ні	Ні	Ні	Ні	Так	Ні	Специфічна для репліки ГРРС; Загальна для ГРРС в залежності від рівня узгодженості
Перевантаження мережі	Ні	Так	Ні	Ні	Так	Ні	Специфічна для репліки ГРРСС; Загальна для ГРРС в залежності від рівня узгодженості та встановленого тайм-ауту
Прослуховування мережі	Так	Ні	Ні	Так	Ні	Так	Специфічна для репліки ГРРС
Загрози природного походження	Ні	Ні	Ні	Так	Так	Ні	Загальна ГРРС
Помилки апаратних засобів	Ні	Ні	Ні	Так	Так	Ні	Специфічна для репліки ГРРС
Відмова підсистем	Ні	Ні	Ні	Так	Так	Ні	Загальна ГРРС
Помилки оператора системи	Ні	Так	Так	Так	Так	Так	Загальна ГРРС
Помилки налаштування системи	Так	Так	Ні	Ні	Так	Ні	Загальна ГРРС
Дії щодо дезорганізації системи	Так	Так	Так	Так	Так	Так	Загальна ГРРС
Використання апаратних/програмних вразливостей	Так	Так	Ні	Так	Ні	Так	Загальна ГРРС

Продовження таблиці 2.2

Переповнення буферу	Так	Так	Ні	Так	Так	Так	Специфічна для репліки ГРРС
Загроза спотворення, видалення або модифікування даних	Ні	Так	Ні	Ні	Так	Ні	Загальна ГРРС
Атака грубою силою	Так	Так	Ні	Так	Ні	Так	Специфічна для репліки ГРРС
Витік інформації	Так	Так	Ні	Ні	Ні	Так	Загальна ГРРС
Вразливості допоміжного ПЗ	Так	Так	Ні	Так	Ні	Так	Специфічна для репліки ГРРС
Несанкціонований фізичний доступ	Так	Так	Ні	Так	Ні	Так	Специфічна для репліки ГРРС
Підміна програмних/апаратних засобів	Так	Так	Ні	Так	Ні	Так	Специфічна для репліки ГРРС
Ураження шкідливим програмним забезпеченням	Так	Так	Ні	Так	Ні	Так	Загальна ГРРС
Віддалене виконання коду	Так	Так	Ні	Так	Ні	Так	Специфічна для репліки ГРРС
Атаки на включення файлів	Так	Так	Ні	Так	Ні	Так	Специфічна для репліки ГРРС
Незашифрована комунікація	Так	Так	Ні	Так	Ні	Так	Специфічна для репліки ГРРС
Відсутність узгодженості між компонентами (репліками)	*Ні*	*Так*	*Ні*	*Ні*	*Ні*	*Ні*	*Загальна ГРРС; унікальна для ГРРС*
Вимоги до сильної узгодженості	*Ні*	*Ні*	*Ні*	*Ні*	*Так*	*Ні*	*Загальна ГРРС; унікальна для ГРРС; залежить від встановленого тайм-ауту та к-сті реплік*

Більшість розглянутих загроз є типовими для інформаційних (клієнт-серверних) систем, але для глобально-розподілених систем з реплікацією, специфічними загрозами зокрема є відсутність узгодженості між компонентами, що є загрозою порушення цілісності. Для вирішення цієї проблеми може-бути встановлені вимоги до сильної узгодженості. Однак, це зменшує продуктивність

системи та збільшує час обробки запитів, що, у свою чергу, є загрозою порушення готовності.

Наприклад, якщо встановлено рівень узгодженості, при якому відповідь очікується від одного вузла зі всіх – це підвищує швидкодію, але може призвести до повернення клієнту застарілих даних, що є порушенням цілісності (див. рис. 2.3). З іншого боку, якщо у системі встановлений сильний рівень узгодженості, тобто необхідно очікувати відповідь від всіх вузлів (реплік) – це збільшує загальний час очікування, що підвищує ймовірність отримання відповіді після тайм-ауту і, тим самим, знижує рівень готовності системи (див. рис. 2.2).

Розглянуті загрози ГРРС є додатковими до традиційних загроз порушень цілісності та готовності, вирішення яких є актуальною.

У табл. 2.3 наведені оцінки ймовірності повернення застарілих даних (порушення цілісності) у загальному випадку та при використанні трьох реплік, що є типовою конфігурацією багатьох глобально-розподілених систем.

Таблиця 2.3 – Оцінка вплив рівня узгодженості ГРРС на стійкість до загроз порушення даних та DDoS атак для системи з трьома вузлами

Рівень узгодженості операції запису	Рівень узгодженості операції читання	Ймовірність неузгодженості системи (повернення клієнту застарілих даних)	
		n=3	n
ONE	ONE	0,66	$1-\frac{1}{n}$
ONE	QUORUM	0,33	$1-\frac{C_{n-1}^{q-1}}{C_n^q}$
ONE	ALL	0	0
QUORUM	ONE	0,33	1-q/n
QUORUM	QUORUM	0	0
QUORUM	ALL	0	0
ALL	ONE	0	0
ALL	QUORUM	0	0
ALL	ALL	0	0

Отже, для мінімізації загрози порушення цілісності через неузгодженість вузлів системи, необхідно підвищити рівень узгодженості. Сильна узгодженість (при якій ймовірність повернення клієнту застарілих даних дорівнює нулю) досягається, коли сума кількості вузлів, що беруть участь у виконанні операції запису та операції читання повинна бути більшою ніж фактор реплікації (див. формулу 1.1). Узгодженість досягається за рахунок того, що при читанні з декількох реплік порівнюється часова мітка запису та клієнту повертається той варіант даних, який має найновішу часову позначку. Однак, як було зазначено вище, підвищення рівня узгодженості негативно впливає на готовність системи, тому що необхідно очікувати відповідь від більшої кількості вузлів системи, що призводить до підвищення ймовірності отримання відповіді з більшою затримкою та запізненням.

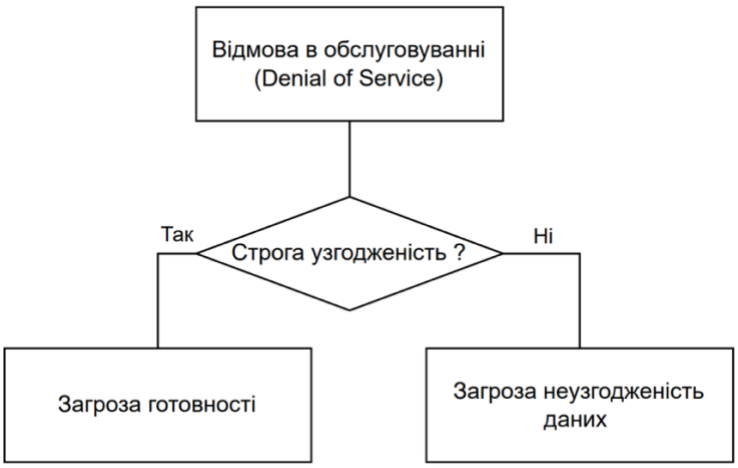


Рисунок 2.2 – Загроза відмови в обслуговуванні системи

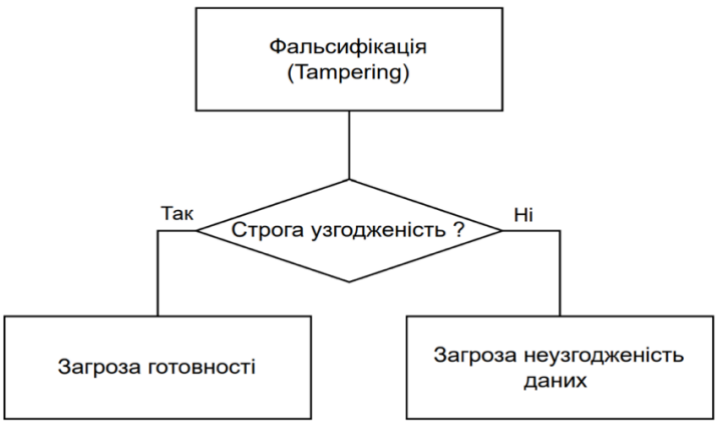


Рисунок 2.3– Загроза порушення цілісності системи

2.2. Теоретико-множинна модель патернів розгортання ГРРС у хмарному середовищі

2.2.1. Нотація розгортання розподіленої бази даних

Архітектура глобально-розподілених систем складається з наступних компонентів:

– вузол (node) – це найпростіший структурний компонент бази даних, який представляє собою обчислювач, який під'єднаний до мережі інших обчислювачів. Здебільшого, обчислювач може собою представляти фізичним сервером, віртуальною машиною, EC2, Azure VM тощо. Обмін інформацію між обчислювачами, відбувається у кільцевій мережі за допомогою протоколу gossip, де кожен обчислювач виконує одні й ті самі функції у порівнянні з іншими;

– віртуальний вузол (virtual node) – представляє собою рівень зберігання даних у вузлі. Кожен вузол за замовчуванням має 256 віртуальних вузлів, які мають визначений діапазон маркерів;

– серверна стійка (rack) – це логічне групування вузлів у кільці. Використовується для розміщення вузлів у різних серверних стійках, для підвищення надійності та готовності;

– датацентр (datacenter) – це логічне групування серверних стійок, які утворені для забезпечення реплікації, що дозволяє зменшити затримку у мережі;

– кластер (cluster) – це компонент, який має один або декілька датацентрів. Кластер глобально-розподіленої системи бази даних складаються з датацентрів, який має вузли, що можуть фізично знаходитись у одній або в декількох географічних локаціях. Якщо брати за приклад публічний хмарний провайдер Amazon web services (AWS), то можна виділити наступні терміни (див. рис. 2.4):

– регіон (region) – це географічне розташування датанцетрів Amazon. Наприклад, Канада (ca-central-1), Велика Британія (eu-west-2);

– зона доступності (availability zone) – це один або група датацентрів, які надають послуги та сервіси в рамках AWS регіону та мають незалежне

енергопостачання, швидкісну мережу передачі інформації. Наприклад, ca-center-1a, eu-west-2b;

– стратегія розміщення (placement groups) – це можливість розмістити обчислювальні ресурси на апаратних можливостях датацентру в залежності від стратегії розміщення: cluster, partition, spread.

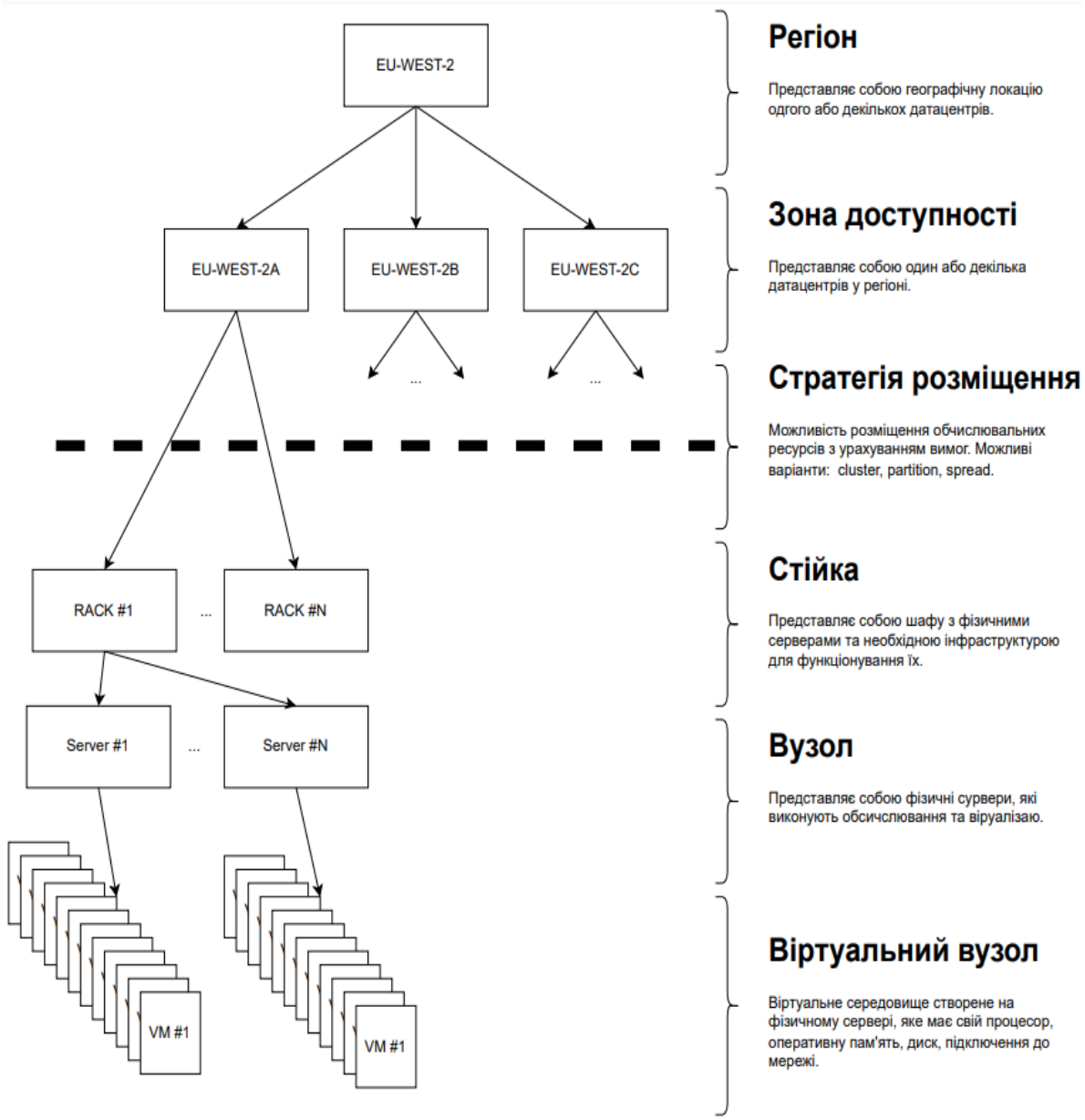


Рисунок 2.4 – Архітектура глобально-розподіленої системи зберігання даних у хмарному провайдері

Для описання різних варіацій розміщення вузлів ГРРС у хмарному провайдері пропонується ввести нотацію, яка покликана формалізувати відношення бази даних до його вузлів:

– круглі дужки () – для визначення кордонів ГРРС;

– кутові дужки <> – для визначення стратегії розміщення вузлів N_i ГРРС у одній зоні доступності хмарного провайдера;

– фігурні дужки {} – для групування клієнта C ГРРС та вузлів N_i у тому ж регіоні хмарного провайдера;

– квадратні дужки [] – для групування вузлів у тій самій зоні доступності. Наприклад, запис $\{[C, (N_1]\}, \{[N_2], \{[N_3]\})$ можна тлумачити, як ГРРС з трьома вузлами, які розміщенні наступним чином: N_1 та клієнт C в одному регіоні та зоні доступності; N_2 та N_3 розгорнуті в іншому регіоні та кожен в окремій зоні доступності.

2.2.2. Патерн розгортання розподіленого кластеру бази даних та клієнта у одній географічній локації та в різних датацентрах

Сценарій розгортання бази даних ГРРС за патерном $\{[C],([N_1],[N_2],[N_3])\}$, характеризується тим, що з точки зору хмарного провайдера кластерні вузли N_1, N_2, N_3 та клієнт C знаходяться в одному регіоні та відмінних один від одного зонах доступності (див. рис. 2.5).

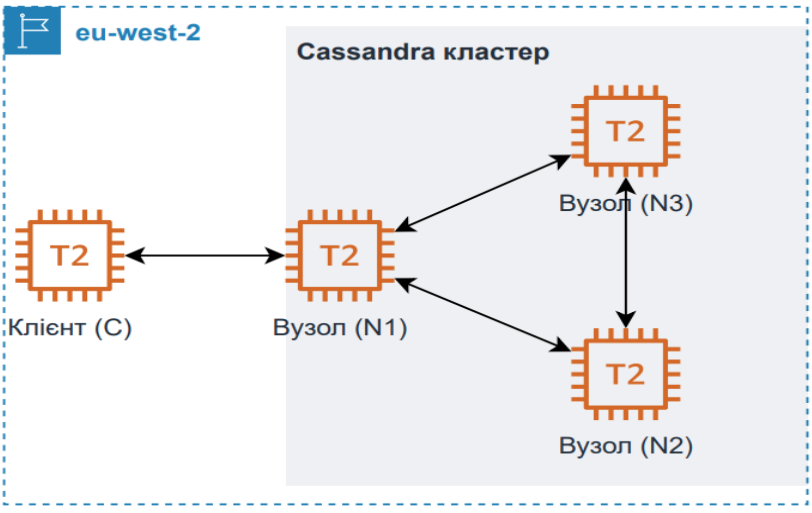


Рисунок 2.5 – Патерн $\{[C], ([N_1],[N_2],[N_3])\}$

2.2.3. Патерн розгортання розподіленого кластеру бази даних у різних датацентрах однієї географічної локації та клієнта у відмінній географічній локації

Сценарій розгортання бази даних ГРСС за патерном {[C]}, ({[N_1],[N_2],[N_3]}), характеризується тим, що з точки зору хмарного провайдера кластерні вузли N_1, N_3, N_3 та клієнт C знаходяться у відмінних регіонах та відмінних один від одного зонах доступності (див. рис. 2.6).

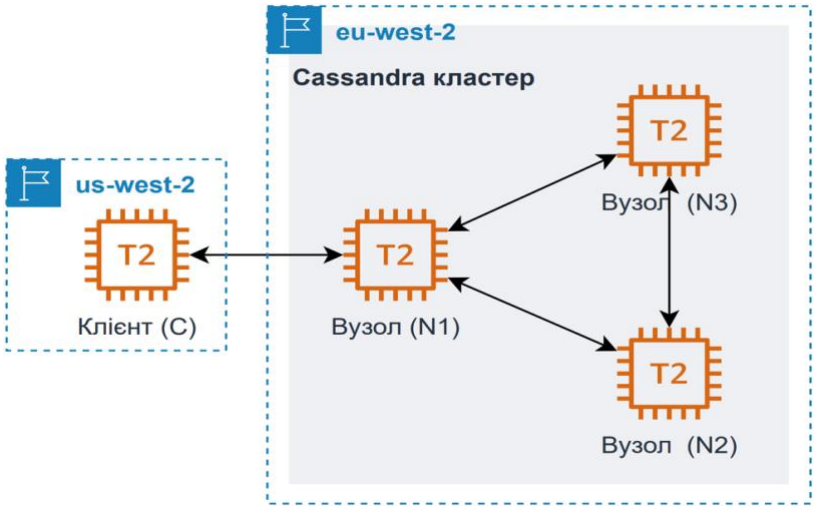


Рисунок 2.6 – Патерн {[C]}, ({[N_1],[N_2],[N_3]})

2.2.4. Патерн розгортання розподіленого кластеру бази даних у різних географічних локаціях та клієнта у одній з географічній локації розподіленого кластеру бази даних

Сценарій розгортання бази даних ГРРС за патерном {[C], ([N_1]}, {[N_2],[N_3]}), характеризується тим, що з точки зору хмарного провайдера кластерний вузол N_1 та клієнт C знаходяться у одному регіоні, а інші вузли N_3, N_3 знаходяться у відмінних регіонах (див. рис. 2.7).

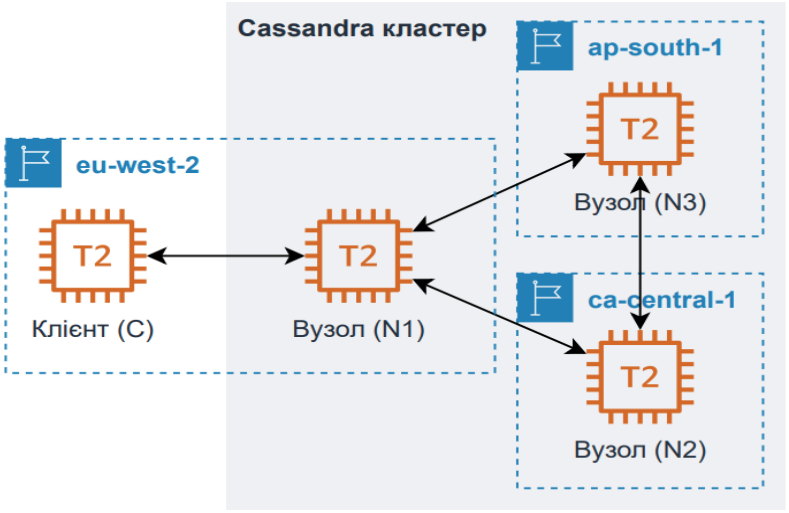


Рисунок 2.7 – Патерн {[C], ([N_1]}, {[N_2],[N_3]})

2.2.5. Патерн розгортання розподіленого кластеру бази даних та клієнта у різних географічних локаціях

Сценарій розгортання бази даних ГРРС за патерном {[C]}, ({[N_1]},{[N_2]},{[N_3]}), характеризується тим, що з точки зору хмарного провайдера кластерний вузли N_1, N_3, N_3 та клієнт C знаходяться у відмінних регіонах (див. рис. 2.8).

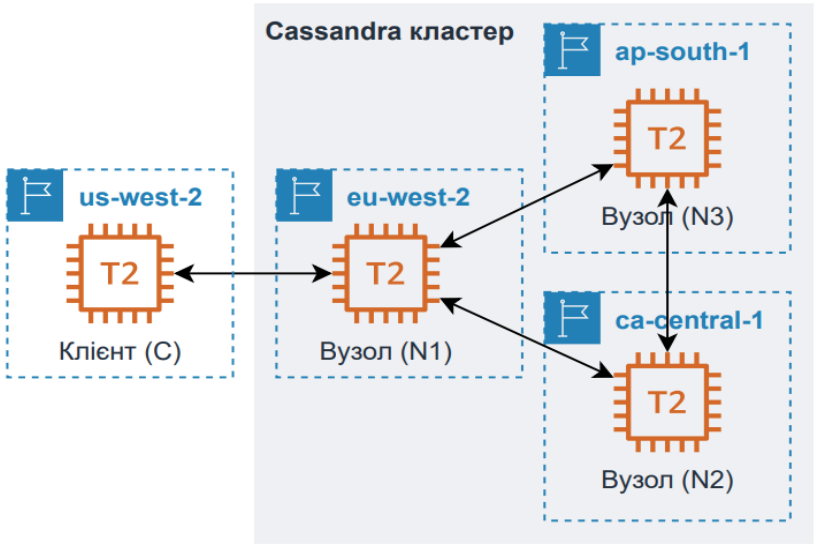


Рисунок 2.8 – Патерн {[C]}, ({[N_1]},{[N_2]},{[N_3]})

2.3. Методика експериментального дослідження характеристик ГРРС

2.3.1. Створення середовища розгортання

У якості середовища для тестування було розгорнуто ГРРС – Cassandra кластер у публічному хмарному середовищі AWS. Цей провайдер хмарних послуг характеризується великою кількістю зон доступності у різних географічних локаціях, що дозволяє розгорнути розподілений Cassandra кластер за патернами, які були описані у попередньому розділі. На сьогоднішній день, типова конфігурація складається з трьох вузлів. Коефіцієнт реплікації рівний трьом є найбільш вживаним для багатьох Інтернет застосунків, наприклад Amazon S3, Amazon EMR, Amazon DunamoDB та інших.

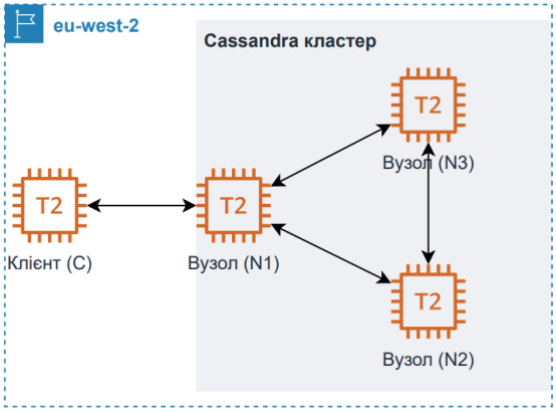
Архітектура кластеру базується на використанні віртуальних машин розмірністю *t2.xlarge*, що характеризується ємністю оперативної пам’яті 16 гігабайт, 4-ма віртуальними процесорами та SSD диском з ємністю у 2x50 гігабайт. Кластерні та клієнтські вузли використовують операційну систему *Ubuntu 16.04 LTS*. Кластерні SSD диски мають конфігурацію *io1*, що дозволяє отримати високу пропускну здатність до 64000 IOPS. Комунікація між клієнтом і вузлами кластера між собою ізольована та сегментована за допомогою Amazon Virtual Private Cloud (VPC), що дозволяє обмінюватися інформацією не виходячи до глобальної мережі Інтернет. VPC є регіональним сервісом тому для обміну інформацією між різними географічними локаціями використовується сервіс VPC Peering Connection. Інфраструктура тестового середовища була описана за допомогою мови HashiCorp Configuration Languages (HCL) [75] для автоматизованого розгортання у публічному хмарному середовищі AWS. Налаштування операційної системи було автоматизовано за допомогою Ansible [76], що дозволяє проводити інсталяцію та налаштування віртуальних машин [12].

Взагалі, було розгорнуто чотири тестових середовища у відповідності до рис. 2.9, що можна формально описати наступним чином [10]:

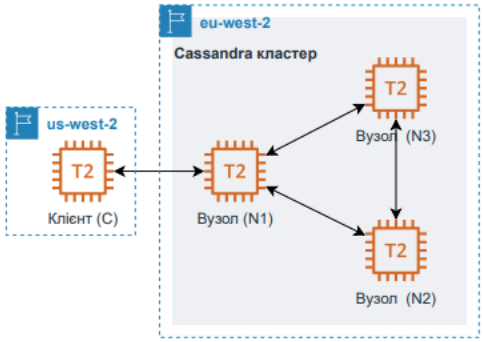
- Централізований (див. рис. 2.9а): $\{[C], ([N_1],[N_2],[N_3])\}$;
- Розподілений (див. рис. 2.9б): $\{[C]\}, (\{[N_1],[N_2],[N_3]\})$;

– Розподілений (див. рис. 2.9в): {[C], ([N_1]}, {[N_2],[N_3]});

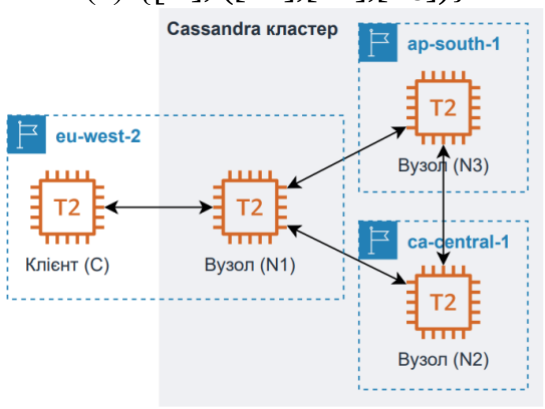
– Розподілений (див. рис. 2.9г): {[C]}, ({[N_1]},{[N_2]},{[N_3]}).



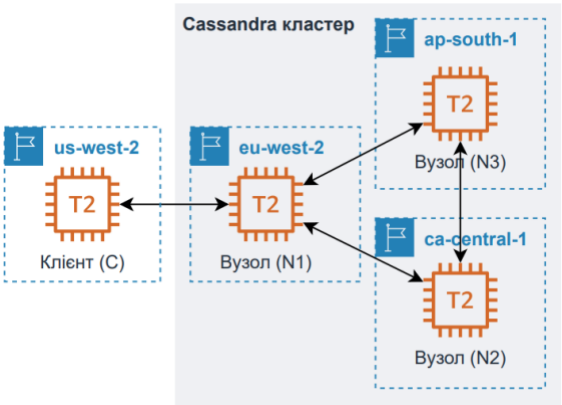
(а) {[C], ([N_1],[N_2],[N_3])}



(б) {[C]}, ({[N_1],[N_2],[N_3]})



(в) {[C], ([N_1]}, {[N_2],[N_3]})



(г) {[C]}, ({[N_1]},{[N_2]},{[N_3]})

Рисунок 2.9 – Сценарії розгортання Cassandra кластеру зі трьома вузлами

2.3.2. Вибір та налаштування інструментарію дослідження продуктивності

На сьогоднішній день стандартизованого інструменту тестування продуктивності NoSQL баз даних не існує, але є деякі ініціативи з боку розробників програмного забезпечення. Існують декілька тестових фреймворків, які підтримують тестування продуктивності таких баз даних, наприклад, Yahoo! Cloud Serving Benchmark (YCSB) [77], KVZone [78], YCSB++ [79] та YCSB+T [80].

Найбільш відомим інструментом тестування є YCSB, який має тестові набори для більшості NoSQL баз даних, що дозволяє виконати порівняння їхньої продуктивності. YCSB може виконувати робочі навантаження щодо операцій читання, запису та оновлення для різних баз даних NoSQL, таких як Cassandra, MongoDB та HBase. Для кожної бази даних реалізовано рівень інтерфейсу бази

даних, який виконує задумане (читання, запис) на рідній мові запитів бази даних. Робоче навантаження виконує операцію над первинним ключем з пов'язаними полями. Однак, робоче навантаження і генерація даних YCSB є дуже простими і не мають можливостей для детальної оцінки всіх можливостей бази даних. Подібні спостереження дали підставу для розширення можливостей YCSB фреймворка у вигляді YCSB+T та YCSB++. Наприклад, YCSB+T фреймворк має на меті оцінити транзакційні накладні витрати операцій з базою даних. YCSB++ фреймворк зорієнтований на підтримку паралельного запуску тестових наборів для різних клієнтів.

Інструмент тестування KVZone [78] є корисним для проведення широкого аналізу сховищ «ключ-значення», маючи кілька параметрів конфігурації для визначення властивостей ключів, значень та операцій з ними, але на даний час підтримує тестування виключно BerkeleyDB, Tokyo Cabinet, SQLite та Alphard баз даних.

Для дослідження швидкодії ГРРС використовується фреймворк YCSB [77, 81]. YCSB фреймворк написаний на мові програмування Java та має відкритий код. YCSB має набір тестів для баз даних, що дозволяє виміряти продуктивність сучасних систем керування базами даних SQL та NoSQL за допомогою згенерованих даних. Також цей фреймворк можна застосовувати для аналізу багатьох архітектурно різних баз даних і вимірювання продуктивності різних конфігурацій баз даних під різними робочими навантаженнями.

Структура YCSB включає шість готових робочих навантажень, кожен з яких тестує різні загально прийняті сценарії використання бази даних з певною суміщу операцій читання та запису, наприклад 50/50, 55/45 тощо. Для експериментального дослідження продуктивності ГРРС було підготовлено окремі тестові набори для операцій запису та читання з урахуванням кількості потоків. Вхідні дані тестових наборів мають наступні параметри [77]:

- розмір ключа, загальна кількість потоків пари «ключ-значення»;
- загальна кількість пар «ключ-значення»;
- загальна кількість операцій;

- кількість потоків;
- рівень узгодженості;
- тип розподілу запитів у ключовому просторі (Zipfian, uniform, latest);
- співвідношення операцій з даними (читання, запис та інші);
- порядок запису (послідовний, хешований).

2.3.3. Алгоритм проведення експериментального дослідження. та налаштування інструментарію дослідження продуктивності

Для проведення експериментального дослідження продуктивності ГРРС було налаштовано тестовий фреймворк YCSB наступним чином [12]:

- кількість потоків від 100 до досягнення межі продуктивності (1000 у нашому випадку) з кроком, достатнім до виявлення закономірності (100 потоків у нашому випадку);
- рівень узгодженості (ONE, QUORUM, ALL);
- кількість запитів в одному потоці;
- співвідношення операцій з даними: тільки читання або запис;
- розмір запису: 1000 байт;
- кількість повторень не менше 10;
- вихідні дані у форматі csv.

Загальний алгоритм проведення експериментального дослідження зображений на рисунку 2.10.

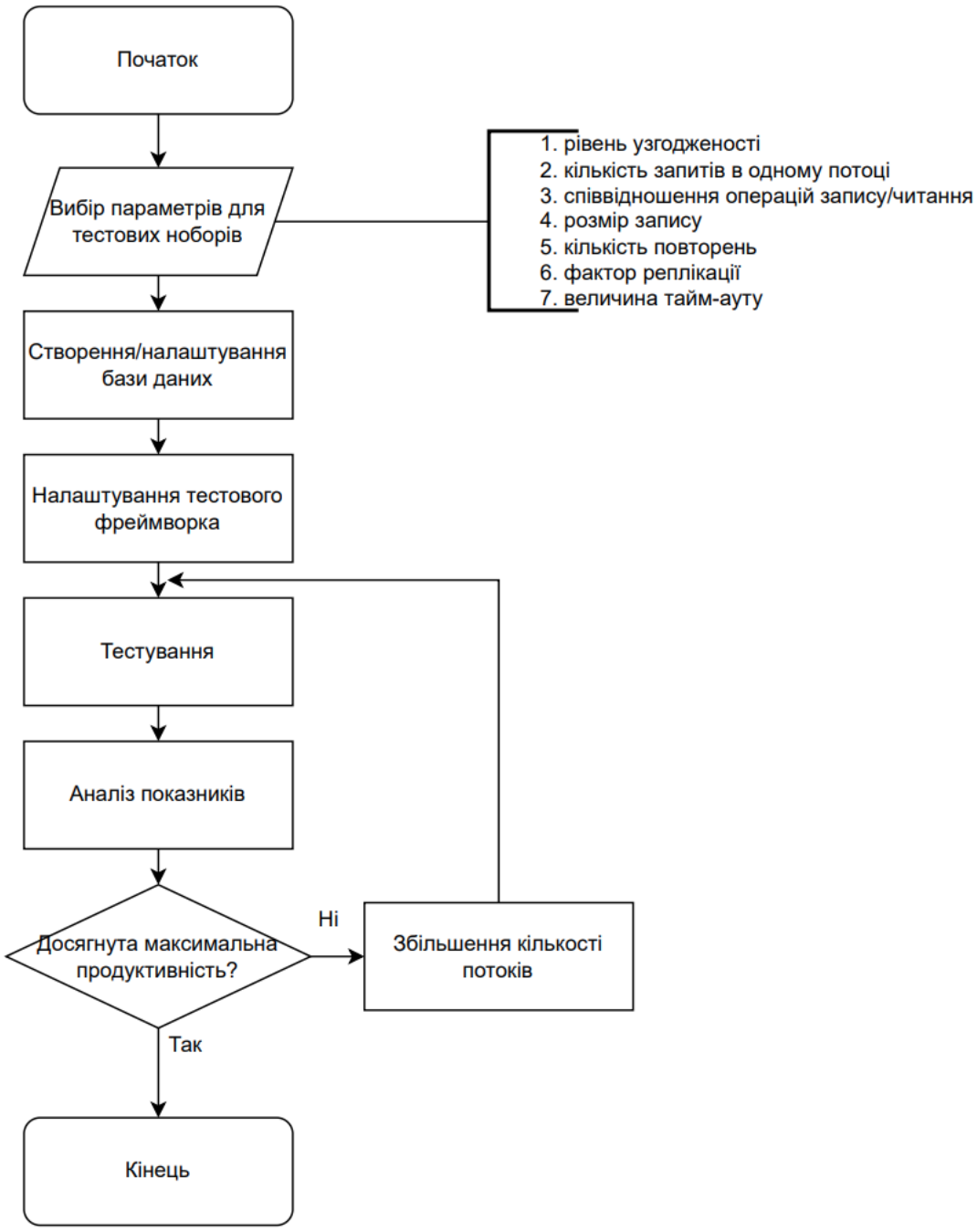
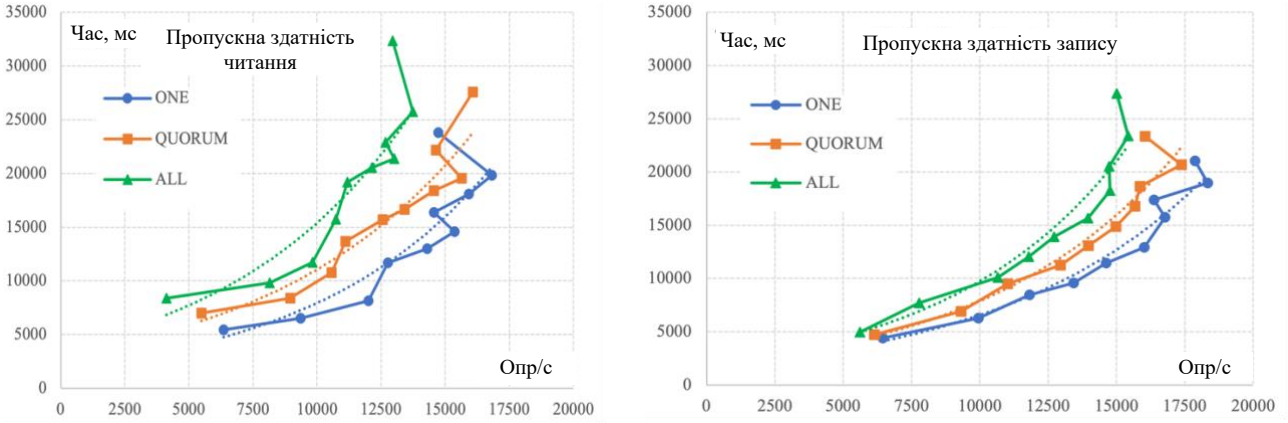


Рисунок 2.10 – Алгоритм проведення експериментального дослідження

2.4. Аналіз експериментальних даних та дослідження взаємозв'язку між узгодженістю та показниками продуктивності

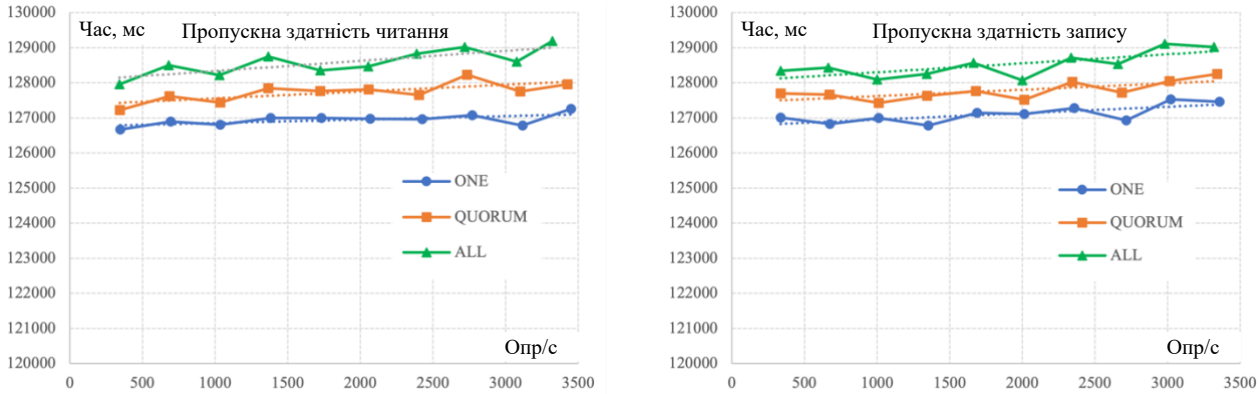
2.4.1. Аналіз продуктивності при операціях читання та запису

Для аналізу продуктивності операцій читання та запису було розгорнуто ГРРС з огляду на патерни, які були розглянуті у розділах: 2.2.2, 2.2.3, 2.2.4 та 2.2.5. Також було виконане навантажувальне тестування у відповідності до алгоритму, який наведений у розділі 2.3.2. Повні результати тестування у табличному вигляді наведені у додатку А. На рисунках 2.11-2.14 наведено графіки часової затримки та пропускної здатності відповідно до патернів розгортання ГРРС.



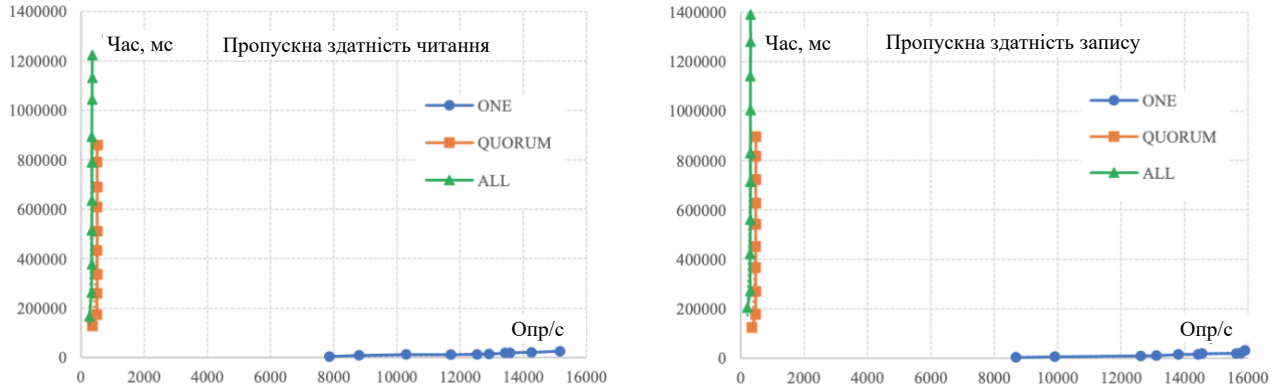
(а) (б)

Рисунок 2.11 – Продуктивність ГРРС за патерном {[C], ([N_1],[N_2],[N_3])}: (а) читання; (б) запис



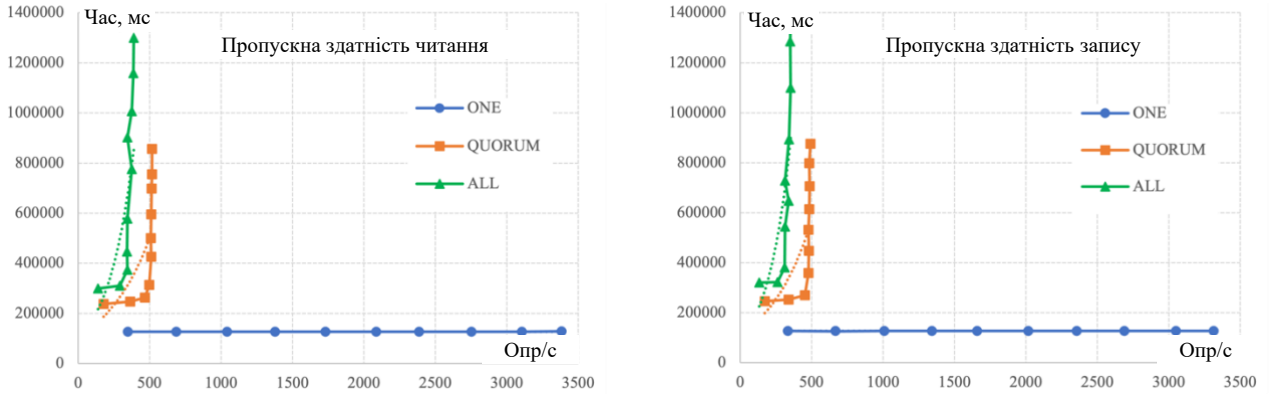
(а) (б)

Рисунок 2.12 – Продуктивність ГРРС за патерном {{[C]}, ({[N_1],[N_2],[N_3]}): (а) читання; (б) запис



(а) (б)

Рисунок 2.13 – Продуктивність ГРРС за патерном {[C], ([N_1]}, {[N_2],[N_3]}): (а) читання; (б) запис



(а) (б)

Рисунок 2.14 – Продуктивність ГРРС за патерном {[C]}, ({[N_1]},{[N_2]},{[N_3]}): (а) читання; (б) запис

Патерн {[C], ([N_1],[N_2],[N_3])}, який має клієнта та репліки в одному регіоні характеризується тим, що в середньому досягає насиченості при 800 потоках. При досягнені максимальної пропускної здатності спостерігається збільшення затримок у експоненціальній прогресії. Також представлені графіки показують, що чим сильніше налаштування узгодженості тим нижча пропускна здатність. Цей факт особливо спостерігається при рівнях узгодженості QUORUM та ALL при патернах {[C], ([N_1]}, {[N_2],[N_3]}) та {[C]}, ({[N_1]},{[N_2]},{[N_3]}), коли застосовується сценарій розгортання у декількох регіонах.

Важливо зауважити, що при дослідженні продуктивності не було досягнуто максимально значення насиченості при 1000 потоках для всіх рівнів узгодженості патерна {[C]}, ({[N_1],[N_2],[N_3]}) та для рівня узгодженості ONE патерна

{[C], ([N_1]}, {[N_2],[N_3]}). З огляду на цей факт Cassandra кластер не досяг максимальної продуктивності через величезний внесок мережевих затримок у загальному часу відгуку (див. табл. 2.4). В інших сценаріях найвища пропускна здатність ГРРС була досягнута під час пікового навантаження (див. табл. 2.5) і була близькою до максимальної пропускної здатності, але у деяких випадках (підкреслені значення) насиченості не було досягнуто.

Таблиця 2.4 – Мережеві затримки між регіонами

Регіони AWS	us-west-2	ca-central-1	eu-west-2	ap-south-1
us-west-2	2.42 мс	66.71 мс	135.22 мс	221.57 мс
ca-central-1	66.93 мс	3.5 мс	79.87 мс	189.02 мс
eu-west-2	135.56 мс	80.14 мс	3.95 мс	111.69 мс
ap-south-1	221.82 мс	188.88 мс	111.89 мс	3.15 мс

Таблиця 2.5 – Найвища пропускна здатність глобально-розподіленого кластеру

Патерн розгортання	ONE		QUORUM		ALL	
	Читання	Запис	Читання	Запис	Читання	Запис
{[C], ([N_1],[N_2],[N_3])}	16830 опр/с	18338 опр/с	16074 опр/с	17384 опр/с	13735 опр/с	15434 опр/с
{[C]}, ({[N_1],[N_2],[N_3]})	*3450* опр/с	*3357* опр/с	*3424* опр/с	*3340* опр/с	*3320* опр/с	*3319* опр/с
{[C], ([N_1]}, {[N_2],[N_3]})	15160 опр/с	15906 опр/с	517 опр/с	478 опр/с	355 опр/с	312 опр/с
{[C]},({[N_1]},{[N_2]},{[N_3]})	*3381* опр/с	*3313* опр/с	518 опр/с	491 опр/с	390 опр/с	357 опр/с

У глобально-розподіленому кластері, який розгорнутий у відповідності до патернів: {[C], ([N_1],[N_2],[N_3])} з усіма рівнями узгодженості та {[C]}, ({[N_1],[N_2],[N_3]}) з рівнем узгодженості ONE, спостерігається факт, що продуктивність операції запису перевищує продуктивність читання в середньому на 9%. Цей факт дає змогу підтвердити твердження про те, що досліджувана глобально-розподілена система була спеціально розроблена, як розподілена система зберігання даних, яка здатна забезпечувати дуже високу пропускну здатність операції запису.

Іншим цікавим спостереженням є той факт, що пропускна здатність операцій читання або запису при патерні {[*C*], ([N_1],[N_2],[N_3])} (при рівні узгодженості ONE) перевищує продуктивність патерну {[*C*], ([N_1]}, {[N_2],[N_3]}) (при рівні узгодженості ONE) на 13%, незважаючи на очевидну схожість. Цей факт можна пояснити тим, що у випадку патерна {[*C*], ([N_1]}, {[N_2],[N_3]}) всі клієнтські запити завжди надсилаються до того самого координаційного вузла відповідно до правила балансування навантаження глобально-розподіленого кластуру, яка враховує мережеву відстань. У патерні {[*C*], ([N_1],[N_2],[N_3])} робоче навантаження клієнта розподіляється рівномірно між усіма репліками глобально-розподіленого кластеру, що збільшує загальну пропускну здатність [10].

2.4.2. Вибір функції регресії для моделювання часових затримок

Функція регресії дає змогу обрати функцію, яка мінімізує функцію втрат. У свою чергу, функція втрат характеризує наскільки сильно обрана функція відхиляється від значень в заданих точках. Точки, які отримані в ході експериментального дослідження мають деякі відхилення, шум і тому функція регресії повинна показувати загальну тенденцію. Для вибору функції регресії було обрано типові сценарії розгортання глобально-розподіленого кластеру, а саме централізований (патерн {[C],([N_1],[N_2],[N_3])}) та розподілений (патерн {[C]},({[N_1]},{[N_2]},{[N_3]})) кластери. Базуючись на отриманих показниках (див. розд. 2.4.1), була оцінена функція регресії за експериментальними даними з використанням методу найменших квадратів, яка дозволяє ефективно прийняти рішення щодо точної оцінки затримок при різних робочих навантаженнях. У таблицях 2.6-2.7 наведені значення показника R-квадрат [82], який оцінює точність апроксимації різних функцій регресії [10].

Таблиця 2.6 – Точність функції регресії експериментальних даних для централізованого Cassandra кластеру

Операція	Рівень узгодженості	Поліноміальна регресія			Лінійна регресія	Експ. регресія
		order=2	order=3	order=4		
READ	ONE	0.9879	0.9898	0.9971	0.9866	0.9597
	QUORUM	0.9732	0.9868	0.9992	0.9697	0.9586
	ALL	0.9642	0.9736	0.9981	0.9626	0.9512
WRITE	ONE	0.9971	0.9971	0.9977	0.9968	0.9569
	QUORUM	0.9976	0.9996	0.9997	0.9976	0.9407
	ALL	0.9943	0.9996	0.9997	0.9897	0.9542

Таблиця 2.7 – Точність функції регресії експериментальних даних для розподіленого Cassandra кластеру

Операція	Рівень узгодженості	Поліноміальна регресія			Лінійна регресія	Експ. регресія
		order=2	order=3	order=4		
READ	ONE	0.9879	0.9898	0.9971	0.9866	0.9597
	QUORUM	0.9732	0.9868	0.9992	0.9697	0.9586
	ALL	0.9642	0.9736	0.9981	0.9626	0.9512
WRITE	ONE	0.9971	0.9971	0.9977	0.9968	0.9569
	QUORUM	0.9976	0.9996	0.9997	0.9976	0.9407
	ALL	0.9943	0.9996	0.9997	0.9897	0.9542

Аналізуючи таблиці 2.6-2.7, можна дійти висновку, що найбільш точно описують експериментальні дані, поліноміальна функція регресії четвертого порядку:

$$yD_{All}^{Read}(x) = 200.82x^4 - 6079.9x^3 + 66136x^3 - 165848x + 411562; \quad (2.1)$$

$$yD_{Quorum}^{Read}(x) = 106.71x^4 - 3507x^3 + 40777x^2 - 110191x + 316066; \quad (2.2)$$

$$yD_{One}^{Read}(x) = 1.6149x^4 - 29.622x^3 + 174.2x^2 - 245.82x + 126515; \quad (2.3)$$

$$yD_{All}^{Write}(x) = 172.28x^4 - 4022.4x^3 + 42148x^2 - 76501x + 350165; \quad (2.4)$$

$$yD_{Quorum}^{Write} = 148.92x^4 - 4316.7x^3 + 44987x^2 - 111081x + 317546; \quad (2.5)$$

$$yD_{One}^{Write}(x) = -1.4094x^4 + 31.096x^3 - 219.89x^2 + 657.15x + 126118; \quad (2.6)$$

$$yC_{All}^{Read}(x) = 27.425x^4 - 564.14x^3 + 3807.9x^2 - 7084.5x + 7922.5; \quad (2.7)$$

$$yC_{Qourum}^{Read}(x) = 16.528x^4 - 323.57x^3 + 2048.9x^2 - 2692.5x + 7922.5; \quad (2.8)$$

$$yC_{One}^{Read}(x) = 11.94x^4 - 248.65x^3 + 1719.6x^2 - 2541.7x + 6483.5; \quad (2.9)$$

$$yC_{All}^{Write}(x) = 0.1047x^4 + 25.676x^3 - 381.66x^2 + 3714.8x + 1623.3; \quad (2.10)$$

$$yC_{Quorum}^{Write}(x) = 1.1473x^4 - 10.468x^3 - 60.659x^2 + 2627.1x + 2132.3; \quad (2.11)$$

$$yC_{One}^{Write}(x) = -3.0313x^4 + 66.617x^3 - 473.1x^2 + 4016.3x + 1779.1, \quad (2.12)$$

де $yD_{All}^{Read}, yD_{Quorum}^{Read}, yD_{One}^{Read}, yD_{All}^{Write}, yD_{Quorum}^{Write}, yD_{One}^{Write}$– час відгуку глобально-розподіленого кластеру на операції читання або запису для різних налаштувань узгодженості розподіленого кластеру; $yC_{All}^{Read}, yC_{Quorum}^{Read}, yC_{One}^{Read}, yC_{All}^{Write}, yC_{Quorum}^{Write}, yC_{One}^{Write}$ – час відгуку Cassandra кластеру на операції читання або запису для різних налаштувань узгодженості централізованого кластеру; x – кількість потоків (паралельних запитів).

2.5. Модель оцінки продуктивності та метод динамічного керування рівнем узгодженості при змішаному навантажені

2.5.1. Моделі оцінки продуктивності ГРРС при змішаному навантажені та гарантованої узгодженості

Глобально-розподілена база даних може гарантувати строгу узгодженість даних, якщо сума реплік, які використовуються для виконання операцій читання та запису перевищує коефіцієнт реплікації в іншому випадку можлива тільки ймовірнісна узгодженість. Тому для глобально-розподіленого кластеру з трьома репліками, що є широко використовується для більшості інтернет застосунків

(Instagram, Facebook, Twitter тощо), можна виділити наступні налаштування узгодженості операцій читання або запису, які завжди гарантують строгу узгодженість даних:

- читання ONE – запис ALL (1R-3W);
- читання QUORUM – запис QUORUM (2R-2W);
- читання ALL – запис ONE (3R-1W);
- читання QUORUM – запис ALL (2R-3W);
- читання ALL – запис QUORUM (3R-2W);
- читання ALL – запис ALL (3R-3W).

Всі вище перелічені конфігурації забезпечують строгу узгодженість, тому розробники розподілених застосунків можуть обирати конфігурацію, яка могла би забезпечити мінімальний, в середньому, час обслуговування. Базуючись на експериментальних дослідженнях виходить, що чим менше реплік викликається, тим вище швидкодія усього кластеру. З огляду на цей факт можна відкинути надлишкові конфігурації, а саме останні три (2R-3W, 3R-2W, 3R-3W). Також важливо зауважити, що затримка відповіді та пропускна здатність глобально-розподіленого кластеру залежить від навантаження та відсоткового співвідношення між кількістю операцій запису та читання. Таким чином, для оцінки часових затримок глобально-розподіленого кластеру при змішаному навантажені пропонується використовувати наступні моделі:

$$y_{1R-3W}(x) = P_{Read} * \ y_{One}^{Read}(x) + P_{Write} * y_{All}^{Write}(x); \qquad (2.13)$$

$$y_{2R-2W}(x) = P_{Read} * \ y_{Quorum}^{Read}(x) + P_{Write} * y_{Quorum}^{Write}(x); \qquad (2.14)$$

$$y_{3R-1W}(x) = P_{Read} * \ y_{All}^{Read}(x) + P_{Write} * y_{One}^{Write}(x), \qquad (2.15)$$

де P_{Read}, P_{Write}– ймовірність надходження запиту на операцію читання або запису даних, які залежать від поточної суміші цих операцій $P_{Read} + \ P_{Write} = 1$.

У таблицях 2.8-2.9 наведено вибіркові показники затримки обслуговування централізованого та розподіленого глобально-розподіленого кластеру для різних конфігурацій, що гарантують строгу узгодженість при змішаному навантаженні. Ці оцінки отримані за допомогою 2.1-2.15 формул.

У таблицях 2.8-2.9 виділено мінімальні значення показників затримок обслуговування серед трьох можливих варіантів: 1R-3W, 2R-2W та 3R-1W.

Таблиця 2.8 – Оцінка часу обслуговування централізованого глобально-розподіленого кластеру для різних рівнів узгодженості в залежності від навантаження та пропорцій операції читання (R) та запису (W)

К-сть потоків	R/W=10/90%			R/W=30/70%			R/W=50/50%			R/W=90/10%		
	1R-3W	2R-2W	3R-1W	1R-3W	2R-2W	3R-1W	1R-3W	2R-2W	3R-1W	1R-3W	2R-2W	3R-1W
100	**5026**	**4917**	**4797**	**5114**	**5374**	**5620**	**5203**	**5830**	**6443**	**5380**	**6743**	**8089**
200	**7608**	**7211**	**6693**	**7357**	**7477**	**7273**	**7106**	**7743**	**7853**	**6605**	**8275**	**9012**
300	**9889**	**9438**	**8543**	**9600**	**9760**	**9382**	**9311**	**10082**	**10222**	**8733**	**10726**	**11901**
400	**11939**	**11510**	**10367**	**11724**	**11943**	**11578**	**11509**	**12375**	**12790**	**11080**	**13241**	**15212**
500	**13852**	**13400**	**12190**	**13695**	**13881**	**13639**	**13538**	**14363**	**15088**	**13224**	**15325**	**17987**
600	**15759**	**15148**	**14033**	**15569**	**15573**	**15488**	**15379**	**15997**	**16942**	**14999**	**16845**	**19852**
700	**17818**	**16859**	**15920**	**17488**	**17150**	**17195**	**17158**	**17442**	**18469**	**16498**	**18024**	**21019**
800	**20221**	**18700**	**17874**	**19683**	**18887**	**18976**	**19146**	**19074**	**20079**	**18070**	**19449**	**22285**
900	**23188**	**20904**	**19919**	**22473**	**21193**	**21197**	**21757**	**21483**	**22475**	**20326**	**22063**	**25031**
1000	**26973**	**23767**	**22079**	**26262**	**24618**	**24365**	**25552**	**25469**	**26652**	**24131**	**27171**	**31225**
1100	**31857**	**27653**	**24379**	**31546**	**29849**	**29139**	**31234**	**32046**	**33899**	**30612**	**36438**	**43419**

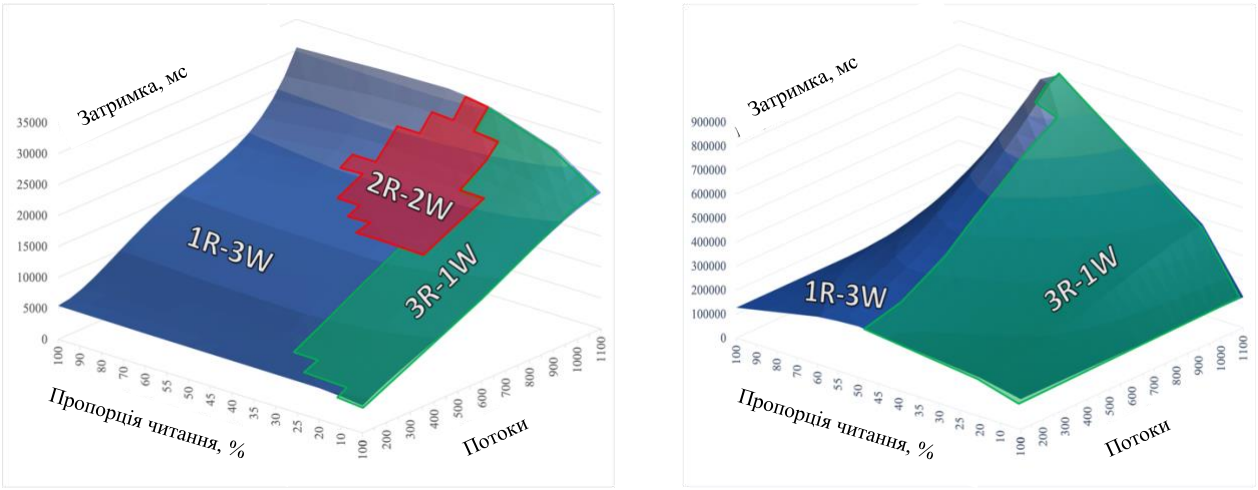
Отримані результати аналітичного моделювання були перевірені за допомогою експериментального вимірювання часу обслуговування глобально-розподіленого кластеру для тих же самих варіантів змішаного навантаження, що наведені на рис. 2.15.

Розглядаючи можливі конфігурації, що гарантують строгу узгодженість, можна дійти до висновку, що компромісним при змішаному навантажені є варіація 2R-2W (тобто при кворумі реплік для операцій читання та запису).

Таблиця 2.9 – Оцінка часу обслуговування розподіленого глобально-розподіленого кластеру для різних рівнів узгодженості в залежності від навантаження та пропорцій операції читання (R) та запису (W)

К-сть потоків	R/W=10/90%			R/W=30/70%			R/W=50/50%			R/W=90/10%		
	1R-3W	2R-2W	3R-1W	1R-3W	2R-2W	3R-1W	1R-3W	2R-2W	3R-1W	1R-3W	2R-2W	3R-1W
100	**286701**	**240629**	**143711**	**251066**	**240187**	**177840**	**215432**	**239746**	**211970**	**144163**	**238862**	**280230**
200	**323543**	**249747**	**144993**	**279776**	**247153**	**181569**	**236010**	**244560**	**218145**	**148477**	**239374**	**291298**
300	**383805**	**288774**	**151055**	**326680**	**284778**	**199608**	**269555**	**280783**	**248161**	**155305**	**272791**	**345267**
400	**466237**	**351397**	**160900**	**390812**	**346605**	**228966**	**315387**	**341814**	**297031**	**164537**	**332230**	**433161**
500	**569589**	**431302**	**173533**	**471209**	**426178**	**266651**	**372828**	**421054**	**359770**	**176068**	**410806**	**546008**
600	**692610**	**522176**	**187955**	**566904**	**517041**	**309674**	**441199**	**511905**	**431393**	**189789**	**501634**	**674832**
700	**834049**	**617705**	**203171**	**676935**	**612736**	**355043**	**519821**	**607768**	**506915**	**205592**	**597831**	**810659**
800	**992657**	**711575**	**218184**	**800336**	**706809**	**399767**	**608015**	**702043**	**581350**	**223372**	**692511**	**944515**
900	**1167184**	**797472**	**231997**	**936143**	**792801**	**440855**	**705102**	**788131**	**649712**	**243020**	**778790**	**1067427**
1000	**1356378**	**869082**	**243614**	**1083390**	**864258**	**475315**	**810403**	**859433**	**707016**	**264428**	**849784**	**1170419**
1100	**1558989**	**920093**	**252037**	**1241115**	**914721**	**500157**	**923240**	**909350**	**748278**	**287490**	**898608**	**1244518**

Але, аналізуючи таблиці 2.7-2.8, конфігурація 2R-2W є краще у досить обмеженій області робочого навантаження та співвідношення операцій запису та читання. Тому, якщо у співвідношенні операцій запису та читання перевищує операція запису (налаштування 3R-1W), забезпечує найменшу затримку для більшості з можливих навантажень. У свою чергу, домінування операції читання (налаштування 1R-3W), стає краще (див. рис. 2.15а). Для розподіленого кластеру цей кордон проходить майже по середині (див. рис. 2.15б). Межа між 1R-3W та 3R-1W знаходиться на 30/70 (30% операцій читання та 70% операцій запису) для централізованого кластеру. Якщо кількість активних потоків більше ніж 700 в діапазоні співвідношення операцій читання/запису від 30/70 до 55/45, то кращим налаштуванням також є і 2R-2W.



(а) (б)

Рисунок 2.15 – Області навантаження з налаштуваннями узгодженості глобально-розподіленого кластеру: (а) централізований кластер; (б) розподілений кластер

Графіки тривимірної поверхні, представлені на рис. 2.15, визначають домени у змішаному робочому навантажені з налаштуваннями узгодженості даних, що забезпечують найбільшу швидкодію глобально-розподіленого кластеру. Вочевидь, вибір налаштувань узгодженості для кожного окремого запиту до бази даних дозволить зменшити час виконання операцій запису/читання. Таким чином актуальним є розробка методу динамічного керування рівнем в залежності від робочого навантаження та пропорції між операціями читання та запису.

2.5.2. Метод динамічного курування рівнем узгодженості операцій читання та запису

Треба зауважити, що отримані у попередніх розділах кількісні результати та обрані функції регресії є унікальними для нашого експериментального дослідження та, вочевидь, зазначені на рис. 2.15 домени налаштувань узгодженості не можуть бути краще для всіх можливих варіантів побудови глобально-розподіленого кластеру.

Дійсно, продуктивність розподіленої інформаційної системи залежить від багатьох факторів, включаючи розмір і структуру бази даних, використовуване обладнання, кількість реплік та їх географічне розташування тощо.

Цей розділ покликаний узагальнити експериментальні дані та формалізує метод динамічного керування рівнем узгодженості за рахунок використання результатів навантажувального тестування, проведення якого є обов'язковим етапом розробки розподілених інформаційних систем.

Запропонований метод (див. рис. 2.16) включає наступні етапи (кроки 1-5 можуть бути виконані одноразово під час навантажувального тестування системи; крок 6 повинен виконуватися безпосередньо під час роботи системи):

1. Розгортання глобально-розподіленого кластеру в реальному виробничому середовищі.

2. Модифікація тестових навантажень YCSB для виконання операцій читання та запису, що є специфічними для розробленої інформаційної системи, що є необхідним для оцінки продуктивності системи в реалістичних сценаріях застосування.

3. Виконання навантажувального тестування глобально-розподіленого кластеру при різних робочих навантаженнях (потоків в секунду) з різними налаштуваннями узгодженості згідно сценарію тестування, описаного в розділі 2.2 та передбачуваним умовам експлуатації.

4. Пошук функцій регресії, які найбільш точно будуть описувати середню затримку виконання операцій читання та запису в залежності від робочого навантаження для різних параметрів узгодженості за результатами практичного вимірювання.

5. Визначення налаштувань узгодженості, використовуючи формули 2.13 - 2.15 для забезпечення мінімальної затримки глобально-розподіленого системи в залежності від робочого навантаження та співвідношення запитів читання/запису. В якості результату, розробники системи зможуть визначити домени змішаного робочого навантаження з кращими налаштуваннями узгодженості (наприклад, див. рис. 2.14).

6. Постійний моніторинг поточного навантаження та співвідношення операцій читання/запис під час роботи системи та динамічне керування рівнем

узгодженості операціями читання та запису, використовуючи домени робочого навантаження, визначені на попередньому кроці.

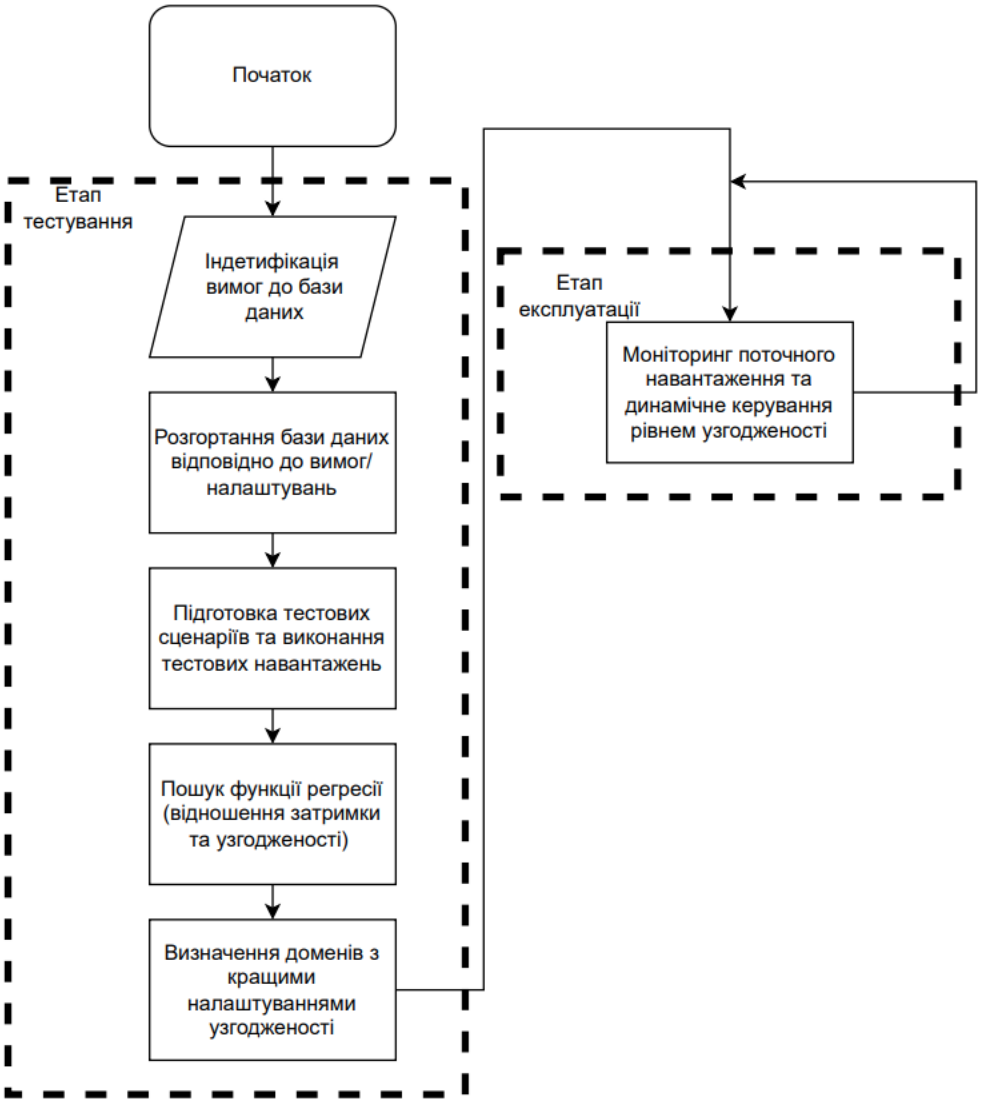


Рисунок 2.16 – Метод динамічного керування рівнем узгодженості операцій читання та запису

2.5.3. Оцінка ефективності методу динамічного курування рівнем узгодженості

Метод запропонований у попередньому розділі був практично реалізований для нереляційної бази даних Cassandra. Після цього його ефективність була перевірена шляхом робочого навантаження для варіантів кластерів розгорнутого централізовано та розподіленого, при наступних змішаних робочих навантаженнях операцій читання/запису: 10/90%, 30/70%, 50/50%, 90/10%. Отримані

експериментальні дані були порівняні з розрахунковими, які наведені у таблицях 2.8-2.9, наприкладі Cassandra кластеру. Таблиці 2.10-2.11 показують відхилення між експериментально виміряних та отриманих аналітичним шляхом у відношенні до кращого налаштування 2R-2W за допомогою формул 2.13-2.15.

Таблиця 2.10 – Виграш за швидкодією (процент зменшення часової затримки) централізованого кластеру за різних робочих навантажень в залежності від співвідношення читання/запису у порівнянні до конфігурації 2R/2W

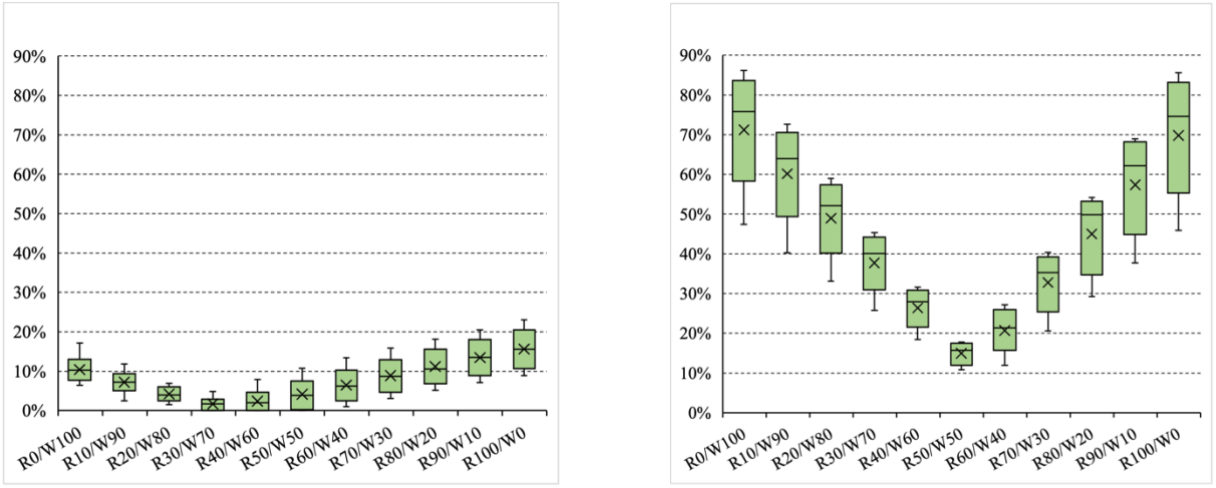
К-сть потоків	R/W=10/90%	R/W=30/70%	R/W=50/50%	R/W=90/10%
100	2%	5%	11%	20%
200	7%	3%	8%	20%
300	9%	4%	8%	19%
400	10%	3%	7%	16%
500	9%	2%	6%	14%
600	7%	1%	4%	11%
700	6%	0%	2%	8%
800	4%	0%	0%	7%
900	5%	0%	0%	8%
1000	7%	1%	0%	11%
1100	12%	2%	3%	16%

Динамічне керування рівнем узгодженості для кожного окремого запиту дозволяє зменшити час виконання операцій читання та запису в середньому на 8% для централізованого кластеру та на 44% для розподіленого кластера при умові забезпечення сильної узгодженості даних (див. рис. 2.17).

Результати такої перевірки свідчать про те, що максимальна різниця між теоретичними та експериментальними значеннями часових затримок не перевищують 17% (ця різниця суттєво зменшується при зростанні робочого навантаження), а запропонований метод дозволяє обрати налаштування узгодженості, що є краще у 92% випадків.

Таблиця 2.11 – Виграш за швидкодією (процент зменшення часової затримки) розподіленого кластеру за різних робочих навантажень в залежності від співвідношення читання/запису у порівнянні до конфігурації 2R/2W

К-сть потоків	R/W=10/90%	R/W=30/70%	R/W=50/50%	R/W=90/10%
100	40%	26%	12%	40%
200	42%	27%	11%	38%
300	48%	30%	12%	43%
400	54%	34%	13%	50%
500	60%	37%	15%	57%
600	64%	40%	16%	62%
700	67%	42%	17%	66%
800	69%	43%	17%	68%
900	71%	44%	18%	69%
1000	72%	45%	18%	69%
1100	73%	45%	18%	68%



(а) (б)

Рисунок 2.17 – Зниження часу обслуговування запитів за рахунок динамічного керування рівнем узгодженості в залежності від пропорції операцій читання/запису у порівнянні з конфігурацією 2R-2W: (а) централізований кластер; (б) розподілений кластер

2.6. Оцінка підвищення готовності та стійкості до DDoS атак при використанні методу динамічного керування рівнем узгодженості ГРРС

Важливим механізмом відмовостійкості для глобально-розподілених систем зберігання даних є тайм-аут, який дозволяє виявити стан розподілу системи внаслідок мережевих затримок, повільної відповіді від вузла, неправильного налаштування системи тощо. Час за який система виявить стан розділу при умові розосередження вузлів у мережі Інтернет, буде неприйнятно великим.

Зазвичай для глобально-розподілених систем зберігання даних значення тайм-ауту жорстко задане, але для ГРРС це може призвести до ситуації, коли відповідь перевершила у часі значення тайм-ауту і в результаті система поверне повідомлення про виняткову ситуацію (помилку). Таким чином перспективнішим є варіант задання значення тайм-ауту динамічно, враховуючи поточне навантаження та пропорцію операцій читання/запису. У якості тайм-аута можна використовувати процентилі в залежності до вимог готовності, наприклад 99.9%, 99% та 95%. Тобто, якщо система має значення тайм-ауту у 99%, то готовність системи перебуває на рівні 99% за умови не невикористання допоміжних засобів обробки виняткових ситуацій. Метод динамічного керування рівнем узгодженості (див. розд. 2.4.2) дає змогу підібрати значення тайм-ауту і отримати виграш по готовності розподіленого та централізованого кластерам (див. табл. 2.11 - 2.16).

Налаштування 2R-2W є компромісним варіантом конфігурації узгодженості для операцій читання та запису і тому обраний рівень тайм-ауту на рівні 99.9%, 99% та 95% від цієї конфігурації. Якщо використовувати цей компромісний варіант налаштування узгодженості то максимальне значення затримки для розподіленого кластеру 99,9% – 575887 мс, 99% – 538825 мс, 95% – 514473 мс, для централізованого кластеру 99,9% – 38335 мс, 99% – 49951 мс, 95% – 61366 мс відповідно. При динамічному налаштуванні узгодженості, пропорції операцій читання/запису, кількості потоків (одночасних запитів) і значенням тайм-ауту на рівні 99.9%, 99%, 95% від конфігурації 2R-2W, готовність системи збільшується,

наприклад для розподіленого кластеру з готовністю 95%, готовність системи з динамічними налаштуваннями збільшується до 98,6%.

Таблиця 2.12 – Підвищення готовності та швидкодії розподіленого кластеру у порівнянні з конфігурацією сильної узгодженості 2R-2W при значенні тайм-ауту 95% процентиль

Конфігурація кластеру	Тайм-аут = 95% процентиль		
	готовність, %	тайм-аут, мс	середня затримка, мс
2R-2W	95	514473	326175
Динамічне налаштування	98,6		201113

Таблиця 2.13 – Підвищення готовності та швидкодії розподіленого кластеру у порівнянні з конфігурацією сильної узгодженості 2R-2W при значенні тайм-ауту 99% процентиль

Конфігурація кластеру	Тайм-аут = 99% процентиль		
	готовність, %	тайм-аут, мс	середня затримка, мс
2R-2W	99	538825	341615
Динамічне налаштування	99,71		211885

Таблиця 2.14 – Підвищення готовності та швидкодії розподіленого кластеру у порівнянні з конфігурацією сильної узгодженості 2R-2W при значенні тайм-ауту 99.9% процентиль

Конфігурація кластеру	Тайм-аут = 99.9% процентиль		
	готовність, %	тайм-аут, мс	середня затримка, мс
2R-2W	99.9	575887	365112
Динамічне налаштування	99,97		226130

Таблиця 2.15 – Підвищення готовності та швидкодії централізованого кластеру у порівнянні з конфігурацією сильної узгодженості 2R-2W при значенні тайм-ауту 95% процентиль

Конфігурація кластеру	Тайм-аут = 95% процентиль		
	готовність, %	тайм-аут, мс	середня затримка, мс
2R-2W	95	38335	24304
Динамічне налаштування	95,69		23560

Таблиця 2.16 – Підвищення готовності та швидкодії централізованого кластеру у порівнянні з конфігурацією сильної узгодженості 2R-2W при значенні тайм-ауту 99% процентиль

Конфігурація кластеру	Тайм-аут = 99% процентиль		
	готовність, %	тайм-аут, мс	середня затримка, мс
2R-2W	99	50951	32302
Динамічне налаштування	99,19		31551

Таблиця 2.17 – Підвищення готовності та швидкодії централізованого кластеру у порівнянні з конфігурацією сильної узгодженості 2R-2W при значенні тайм-ауту 99.9% процентиль

Конфігурація кластеру	Тайм-аут = 99.9% процентиль		
	готовність, %	тайм-аут, мс	середня затримка, мс
2R-2W	99.9	62366	39540
Динамічне налаштування	99,9048		38720

Якщо мінімальною цілю атакуючого, який виконує атаку DDoS атаку на систему є підвищення середнього часу обслуговування до встановленого тайм-ауту [83], у такому випадку йому необхідно, наприклад, при середньому часі обслуговування 201113 мс і 514473 мс там-ауті, що дорівнює 95% процентилю (див. табл. 2.12), перевантажити систему у 2.55 рази. Реалізація запропонованого

методу дозволяє зменшити середній час обслуговування з 326175 мс до 201113 мс та забезпечити стійкість до DDoS атак на 61% (див. табл. 2.12).

2.8. Висновки до другого розділу

1. У цьому розділі побудована модель загроз глобально-розподілених систем за допомогою методу STRIDE. Виділені специфічні загрози порушення цілісності та готовності глобально-розподілених систем з реплікацією.

2. Вперше запропонована теоретико-множинна модель опису патернів розгортання глобально-розподілених реплікованих систем зберігання даних спираючись на можливості її та провайдера хмарних послуг.

2. Вперше запропоновано метод динамічного керування рівнем узгодженості ГРРС для кожної операції читання та запису динамічно під час роботи системи шляхом моніторингу поточного навантаження та використання результатів бенчмаркінгу, зібраних на етапі навантажувального тестування.

3. Аналіз результатів використання методу динамічного керування рівнем узгодженості глобально-розподілених реплікованих систем зберігання великих даних показав, що:

– підвищено швидкодію ГРРС для централізованого кластеру у середньому на 8% та розподіленого кластеру на 44%;

– підвищено готовність ГРРС для централізованого кластеру при початковій готовності з 99% до 99.19% та розподіленого кластеру при початковій готовності з 99% до 99.71%;

– підвищено стійкість до DDoS атак ГРРС на 61%, при умові зменшення середнього часу обслуговування запитів системи.

РОЗДІЛ 3
МЕТОД НАДЛИШКОВИХ ЧИТАНЬ ГЛОБАЛЬНО-РОЗПОДІЛЕНИХ РЕПЛІКОВАНИХ ІНФОРМАЦІЙНИХ СИСТЕМ ДЛЯ ПІДВИЩЕННЯ ГОТОВНОСТІ ТА ЦІЛІСНОСТІ

3.1. Механізм надлишкових читань

3.1.1. Підхід до підвищення готовності та зменшення екстремальних часових затримок на основі використання надлишкових читань

Для підвищення швидкодії операцій читання у глобально-розподілених реплікованих системах, які підтримують послаблений рівень узгодженості, пропонується механізм надлишкових читань. Цей механізм полягає в ініціюванні додаткових (n - k)-запитів (n – загальна кількість вузлів; k – рівень узгодженості) читання до інших вузлів. Після отримання відповіді від k-перших реплік, що задовольняє встановлений рівень узгодженості, результат повертається клієнту. Відповіді, що надійшли пізніше – скасовуються. З одного боку, це дозволяє мінімізувати ймовірність екстремальних затримок, а з іншого – це може підвищити середній час очікування, оскільки, підвищується навантаження на кластер бази даних.

3.1.2. Теоретико-множина нотація для опису надлишкових читань

Модель узгодженості при виконанні операції читань доповнюється параметром s, який буде відображати кількість вузлів до яких потрібно відправити надлишковий запит на читання. Операція читання буде вважатися успішно виконаною, а її результат буде відправлений користувачеві після отримання перших k із s відповідей, у свою чергу, інші відправлені дочірні запити будуть ігноруватися. Таким чином кожна операція читання замість двійки (k, n) буде визначатися трійкою (k, s, n), де k – рівень узгодженості, тобто кількість реплік від який очікується відповідь для успішного завершення операції читання; s – кількість

реплік, яким фактично надсилається запит на читання; n – загальна кількість вузлів. Параметр s може визначатися як статично, так і динамічно в процесі роботи системи з урахуванням поточного навантаження та вимог щодо затримок обслуговування при виконанні читань.

3.1.3. Приклад реалізації надлишкових читань

Для ілюстрації роботи механізму надлишкових читань для підвищення готовності або цілісності, запропоновано розглянути наступні конфігурації:

– кількість реплік – 3, рівень узгодженості – ONE, надлишкове читання відсутнє (нотація 1-1-3) (див. рис. 3.1);

– кількість реплік – 3, рівень узгодженості – ONE та з надлишковим читанням (нотація 1-2-3). Механізм роботи надлишкового читання для підвищення готовності зображено на рис. 3.2, а для підвищення цілісності на рис. 3.4;

– кількість реплік – 3, рівень узгодженості – ONE та з надлишковим читанням (нотація 1-3-3). Механізм роботи надлишкового читання для підвищення готовності зображено на рис. 3.3, а для підвищення цілісності на рис. 3.5.

Наприклад, якщо надлишкові читання з нотацією (1-3-3) застосовуються для підвищення готовності системи виконуються наступні дії (див. рис. 3.5). По-перше, відправляються запити до усіх наявних реплік – трьох вузлів. По-друге, отримавши найскорішу відповідь від одного вузла, ця відповідь негайно відправляється клієнту, не чекаючи відповідей від інших вузлів. У той же час після отримання найшвидшої відповіді, відправляється запит на скасування у тому випадку, якщо вона ще не надійшла.

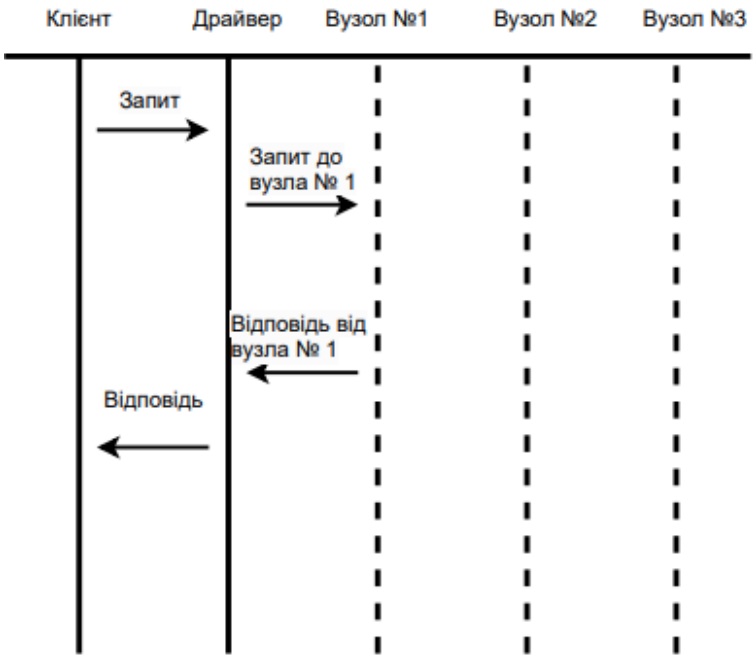


Рисунок 3.1 – Механізм роботи надлишкових читань (нотація 1-1-3) для підвищення готовності

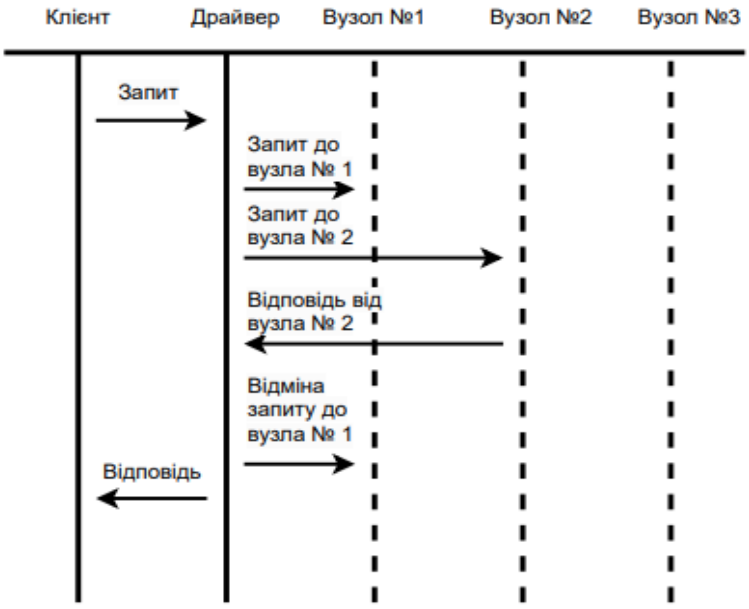


Рисунок 3.2 – Механізм роботи надлишкових читань (нотація 1-2-3) для підвищення готовності

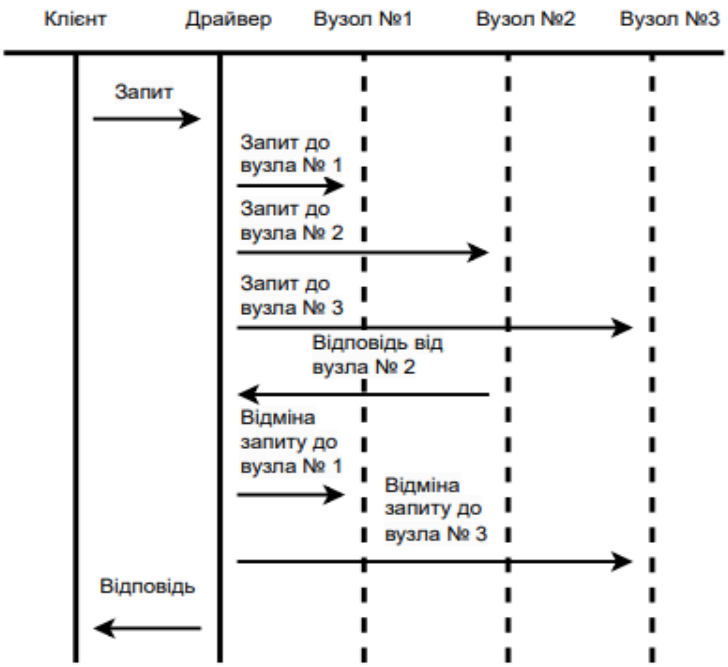


Рисунок 3.3 – Механізм роботи надлишкових читань (нотація 1-3-3) для підвищення готовності

Якщо використовуються надлишкові читання для підвищення цілісності даних, виконуються наступні дії. По-перше, відправляється запит до всіх наявних вузлів в системі. По-друге, очікується відповіді від усіх наявних реплік. По-третє, отримавши відповіді виконується часових міток і контрольних сум. За результатами мажоритарного голосування клієнту відправляється відповідь. Якщо допустити, що був факт порушення цілісність даних однієї з реплік то це дозволяє виправити порушення даних. Для нотації 1-2-3 (див. рис. 3.4), коли кількість реплік недостатньо для мажоритарного голосування, тим не менш можливо визначити факт порушення даних. У такому випадку можливі наступні варіанти: відправка повідомлення про порушення даних замість відповіді або випадково обрана відповідь зі супроводжуючим повідомленням про можливе порушення даних.

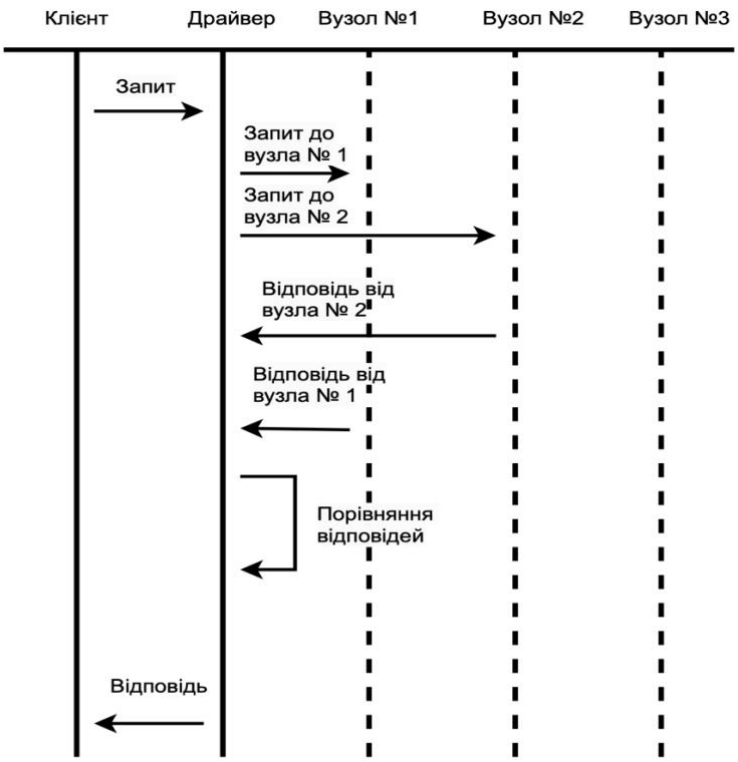


Рисунок 3.4 – Механізм роботи надлишкових читань (нотація 1-2-3) для підвищення цілісності

У випадку застосування нотації 1-3-3 (див. рис. 3.5) запит на вибірку даних відправляється до всіх вузлів у системі. Після отримання відповідей від усіх вузлів-реплік на першому етапі проводиться порівняння часових міток та обираються найбільш новіші дані. На другому етапі виконується порівняння цих даних (наприклад, за рахунок порівняння контрольних сум). Якщо відповіді різняться то виконується мажоритарне голосування (при наявності достатньої кількості відповідей), результат якого повертається клієнту.

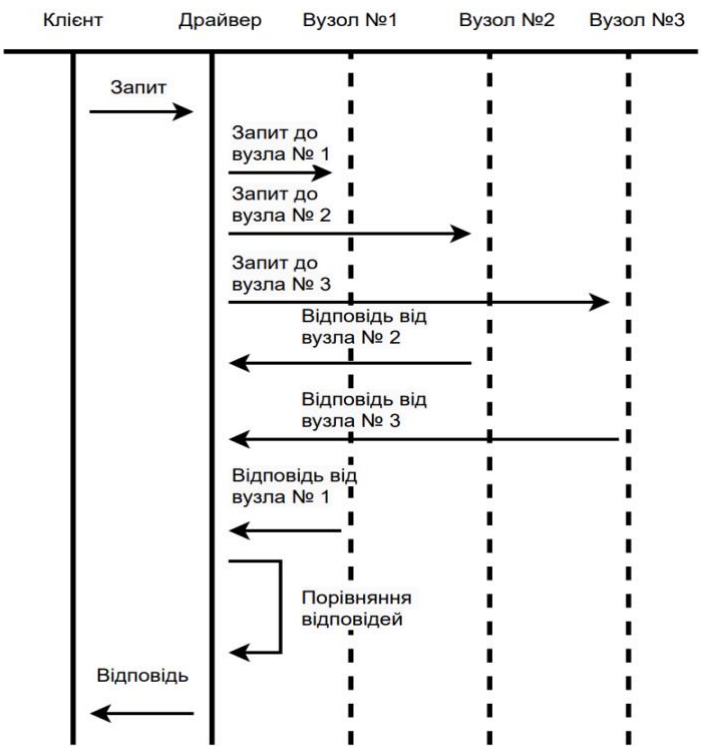


Рисунок 3.5 – Механізм роботи надлишкових читань (нотація 1-3-3) для підвищення цілісності

3.2. Імітаційна модель оцінки продуктивності надлишкових читань при їхньому використанні для підвищення готовності

3.2.1. Розробка імітаційної моделі

Для використання запропонованого механізму надлишкових читань необхідно мати можливість оцінки впливу параметру *s* на зменшення екстремальних затримок (наприклад, 95% процентиль) з одного боку, та можливого збільшення середнього часу обслуговування. У багатьох випадках функціонування клієнт-серверних систем може бути описана за допомогою теорії масового обслуговування (ТМО) [84]. Теорія масового обслуговування використовує поняття, як заявка (вимога) на обслуговування або обробка вимоги, черга та вихідний потік вимог. Предметом ТМО є встановлення залежності між характеристиками потоку заявок та ефективність процесу їх обробки при різних способах організації цього процесу. У ТМО вхідні та вихідні (оброблені) потоки характеризуються випадковим законом

розподілу. Здебільшого використовується пуассонівський закон розподілу, який дає змогу легко описати і побудувати математичну модель.

Однак, у глобально-розподілених системах зберігання даних вхідний та вихідний потік заявок складно представити за допомогою пуассонівського закону розподілу або інших тому, що існує велика невизначеність часових затримок у глобальній мережі Інтернет [85], а також наявність багатьох факторів, які впливають на час обслуговування. Більш того, фактично неможливо відокремити час пересування інформації мережою Інтернет від саме часу обробки запитів. Подальші експерименти показали, що загальний час обслуговування (без можливості розділу на час обслуговування та мережеву затримку) можна представити за допомогою розподілів зі повільно спадним хвостом, наприклад, розподілами Гамма або Вейбулла, але тільки впродовж дуже незначного проміжку часу, яке охоплює, як правило не більш ніж 20-30 останніх послідовних запитів. Наприклад, результати перевірки гіпотез розподілу загального часу обслуговування для системи зберігання даних, яка була розгорнута у одній зоні доступності (availability zone) хмарного провайдера AWS представлені у табл. 3.1.

З урахуванням вище зазначених факторів, доцільним є розробка гібридної імітаційної моделі обслуговування запитів у глобально-розподілених інформаційних системах, яка б використовувала результати експериментально дослідження продуктивності для визначення законів розподілу часу обслуговування у залежності від поточного навантаження.

Для перевірки гіпотези про закон розподілу часу обслуговування запиту до глобально-розподіленого кластеру була використана методика, запропонована у статі [86]. Методика складається з наступних етапів: на першому етапі виконується налаштування параметрів відомих законів розподілу за отриманими статистичними даними у ході експериментальних досліджень; на другому етапі перевіряються гіпотези про несуперечність статистичних даних законам розподілу зі знайденими на першому етапі параметрами.

Для перевірки статистичних гіпотез і підбір параметрів законів розподілу випадкової величини використовувались функції програмного забезпечення

MATLAB [87], що має у своєму складі функцію *kstest([h,p]=kstest(x,cdf))*, яка виконує тест Колмогорова-Смірнова. Перевіряємою гіпотезою є підпорядкування експериментальних даних *x* певному закону розподілення зі заданою інтегральною функцією *cdf* (cumulative distribution function).

У результаті перевірки гіпотези формується результат *h*, який приймає 1 у разі відхилення перевіряємої гіпотези та 0 у іншому випадку. Також додатково повертається ймовірність *p* [88, 89] того, що експериментальні дані не суперечать гіпотезі, що перевіряється. Якщо ця ймовірність більша або дорівнює 0.05, альтернативна гіпотеза має бути відхилена, а прийнята нульова гіпотеза про підпорядкування експериментальних даних заданому закону розподілу.

Як свідчать результати, що представлені у табл. 3.1, для імітаційного моделювання часу обробки запитів доцільного використовувати Гамма розподіл з параметрами, які можуть бути обчисленні методом найменших квадратів з використанням експериментальних даних, що були отримані у розділі 2.4.

Таблиця 3.1 – Узгодженість експериментальних даних та теоретичних законів розподілу

Кількість екземплярів	Імовірна значущість (p-значення)					
	експ.	гамма	норм.	Релея	Вейбулла	Пуассона
300	1.45E-08	2.05E-05	6.34E-80	1.89E-07	1.67E-03	4.58E-65
Перші 150	1.24E-03	7.92E-04	1.75E-34	1.38E-07	2.86E-03	2.18E-32
Другі 150	1.37E-06	2.53E-02	3.77E-35	0.1557	**0.1659**	7.58E-28
Перші 50	0.0879	**0.2748**	4.71E-20	2.23E-05	0.0763	1.22E-18
Перші 25	0.0847	**0.7717**	8.17E-07	4.68E-03	0.0813	8.17E-07
Другі 25	1.59E-03	**0.6791**	1.73E-09	2.08E-03	0.0715	1.73E-09

3.2.2. Структура гібридної імітаційної моделі

Гібридна імітаційна модель включає до себе наступні процеси (див. рис. 3.6).

1. *Моніторинг часу обслуговування запитів та здвиг ковзаючого вікна*. Процес моніторинг навантаження розгорнутої системи у виробничому середовищі та рух ковзаючого вікна для відслідковування послідовності запитів. Розмір ковзаючого

вікна залежить від гіпотези про закон розподілу ймовірності, здебільшого це 20-30 послідовних останніх запитів.

2. *Побудова функцій щільності часу обслуговування.* За результатами отриманих статистичних даних виконується побудова гістограм щільності часу обслуговування.

3. *Визначення функцій щільності розподілу часу обслуговування та їх параметрів.* На основі передне отриманих статистичних даних визначаються функції щільності розподілу часу обслуговування для кожного вузла системи, перевірка статистичних гіпотез та параметрів законів розподілу випадкової величини. У результаті для кожної репліки буде отримана функція щільності розподілу часу обслуговування, яка є входом до імітаційної моделі.

4. Гібридна імітаційна модель за методом Монте-Карло. Вхідні параметри для гібридної імітаційної моделі слугують функції щільності часу обслуговування та величина там-ауту системи. Вихідними параметрами моделі є рівень готовності системи, параметри швидкодії (мінімальна, максимальна та середня затримки). Детальний аналіз та алгоритм гібридної імітаційної моделі описано у розділі 3.2.3.

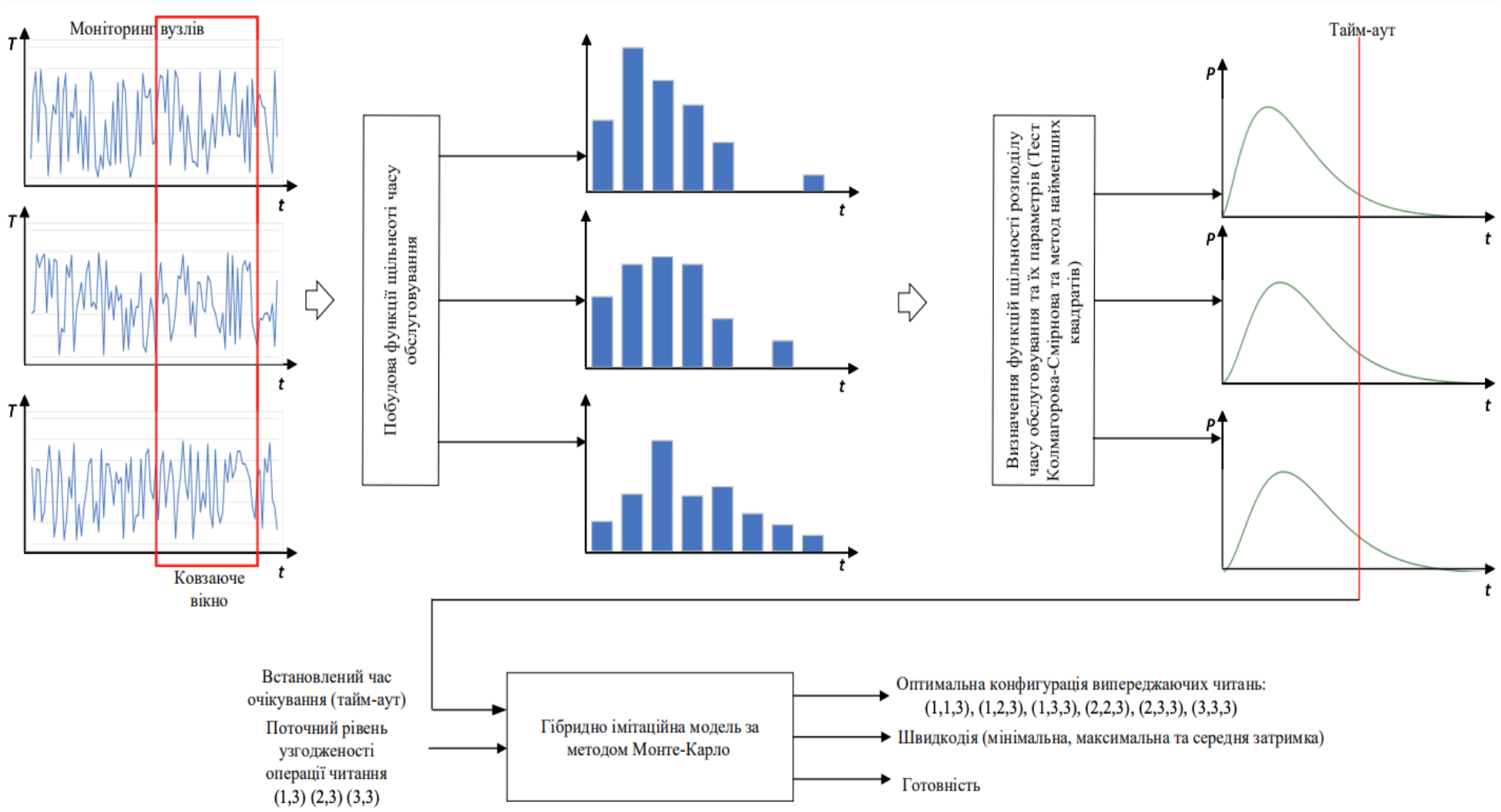


Рисунок 3.6 – Процес гібридно імітаційного моделювання

91

3.2.3. Алгоритм імітаційного моделювання продуктивності надлишкових читань за методом Монте-Карло

Для моделювання поведінки роботи глобально-розподіленої системи зберігання даних будемо застосовувати метод Монте-Карло з параметрами законів розподілу часу обслуговування при різних навантаженнях, що були отримані при аналізі експериментальних даних (див. розділ 2.5).

Вхідними даними імітаційної моделі є: трійка (k, s, n); параметри законів розподілу часу обслуговування при встановленому навантаженні та навантаженнях кратних s; кількість запитів до інформаційної системи. В якості вихідних даних модель повертає часові затримки обслуговування запитів, що відповідають параметрам (k, s, n). Ці дані будуть використовуватися для побудови рядів щільності розподілу часу обслуговування та розрахунку процентилей для оцінки ефективності надлишкових читань.

Розглянемо роботу алгоритму (див. рис. 3.7) для системи зберігання даних з трьома репліками ($n = 3$). Для даної конфігурації глобально-розподіленого кластеру, рівень узгодженості k може приймати наступні значення: *ONE*, *QUORUM* та *ALL*. При рівні узгодженості *ONE* можливі наступні конфігурації: $s = 1$ – надлишкове читання відсутнє і значення затримки визначається часом відповіді від репліки вибір, якої залежить від алгоритму роботи глобально-розподіленого кластеру; $s = 2$ – конфігурація з надлишковим читанням, значення затримки визначається найменшим часом відповіді з двох опитаних реплік; $s = 3$ – конфігурація з надлишковим читанням, визначення затримки є ідентичним до попередньої конфігурації, але в опитування приймає участь три репліки.

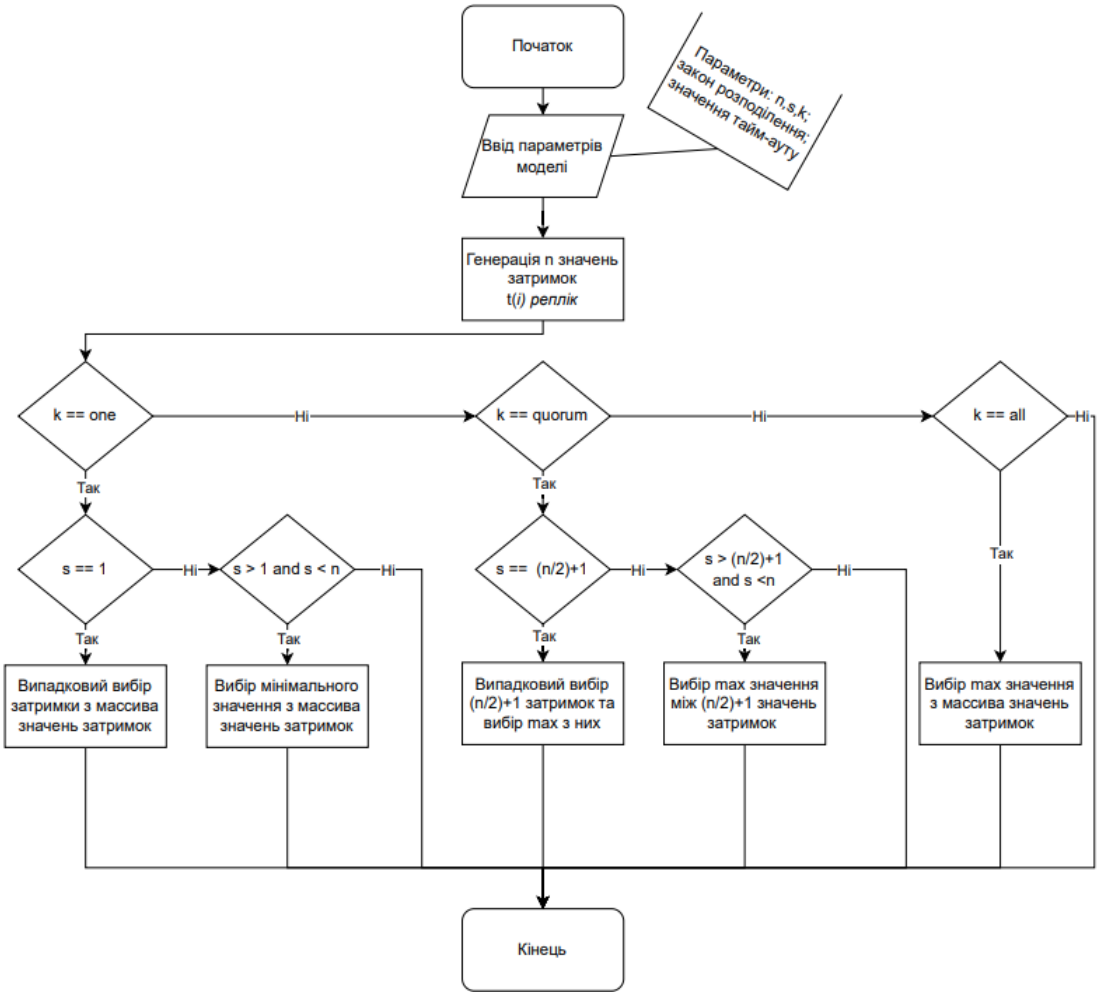


Рисунок 3.7 – Алгоритм гібридного імітаційного моделювання часових затримок при використанні методу надлишкових читань

При рівні узгодженості *QUORUM* можливі наступні конфігурації: $s = 2$ – надлишкове читання відсутнє, значення затримки визначається максимальним часом відповіді з поміж двох опитаних реплік; $s = 3$ – конфігурація з надлишковим читанням, значення затримки визначається вибором максимального часу відповіді з двох мінімальних затримок при опитуванні трьох реплік. При рівні узгодженості *ALL* можлива лише один варіант конфігурації при $s = 3$ – надлишкове читання відсутнє і затримка визначається максимальним часом відповіді для трьох опитаних реплік.

3.3. Метод надлишкових читань для підвищення готовності ГРРС

3.3.1. Структура методу надлишкових читань з використанням гібридної імітаційної моделі

Метод надлишкових читань для глобально-розподілених реплікованих систем складається з наступних кроків.

1. *Отримання поточних налаштувань узгодженості (k) та кількісті вузлів у системі (n*). В залежності від кількості вузлів у кластері зберігання даних і налаштувань узгодженості, обираються можливі варіації налаштувань надлишкових читань. З огляду на це, формується список можливих налаштувань відповідно до вимог кластеру.

2. *Розгортання глобально-розподіленого кластеру у реальному виробничому середовищі*. Розгортання хмарної інфраструктури за допомогою інструментів автоматизації, відповідно до патерну розгортання кластеру у хмарному середовищі, налаштувань узгодженості та кількості вузлів.

3. *Отримання поточної інформації щодо продуктивності системи*. Процес отримання часових затримок роботи глобально-розподіленого кластеру, розгорнутого у хмарному середовищі з оглядом на рівень узгодженості даних та кількості вузлів кластеру.

4. *Застосування гібридної імітаційної моделі для отримання ряду щільності розподілу, мінімальний, максимальний та середній час обробки запиту*. Процес застосування гібридної імітацийної моделі, описаної у розділі 3.3.2, для отримання часових затримок, а саме мінімального, максимального та середнього. Результатом моделювання є вихідні параметри часових затримок та процентилей, які використовуються для оцінки ефективності налаштувань надлишкових читань.

5. *Вибір конфігурації надлишкового читання*. Процес вибору конфігурації надлишкових читань, яке максимально знижує найгірший час обробки запиту до системи (worst execution time) та не підвищує середній час обслуговування вище, ніж встановлене у вимогах до системи.

6. *Підрахунок залежності від середнього часу обробки запиту до максимального*. Процес співставлення максимального та середнього значень та оцінки ефективності надлишкових читань.

Особливістю запропонованого методу є те що він може вирішувати задачу зменшення екстремальних затримок при деякому збільшені середнього часу обробки запитів. З огляду на це, можна здійснити вибір параметру *s,* вирішуючи задачу у двох постановках:

– зменшення екстремальних часових затримок (максимального часу обслуговування або певного процентилю, наприклад, 90%, 95% або 99%) при обмеженні на зростання середнього часу обслуговування запитів;

– мінімізація зростання середнього часу обслуговування при обмеженні на максимальний час обслуговування або величини певного процентилю 90%, 95% або 99%).

Об’єктом застосування, описаного вище методу, є глобально-розподілені репліковані системи зберігання даних з багатьма вузлами. Якщо вузли системи зберігання даних знаходяться в одному датацентрі то ефект від методу може бути незначним. У ситуації, коли вузли розподілені глобально через мережу Інтернет, де час обслуговування запитів є великим і варіація затримок також є значною, то ефект від методу буде прослідковуватися.

3.3.2. Результати використання методу надлишкових читань для рівня узгодженості ONE з використанням гібридної імітаційної моделі

Розглянемо конфігурацію глобально-розподіленого кластеру з трьома репліками, що використовує рівень узгодженості ONE. У відповідності до запропонованої теоретико-множинної моделі (див. розд. 3.2.1), така конфігурація описується трійкою (*k, s, n*), як (*1,1,3*): рівень узгодженості ONE, кількість опитуваних вузлів 1 від загальної кількості вузлів, загальна кількість вузлів 3. У такій системі у відповідності до результатів експериментального дослідження (див. розд. 2.5.1) середній час обслуговування запитів дорівнює 4092 мс, а максимальний

час затримки дорівнює 38968 мс (див. табл. 3.2). Нехай для такої системи там-аут встановлений на рівні 90% процентиль (див. рис. 3.9а), тобто 8635 мс.

Тепер розглянемо систему, яка описується патерном (*k, s, n*), як (*1,2,3*): рівень узгодженості ONE, кількість опитуваних вузлів 2 від загальної кількості вузлів, загальна кількість вузлів 3. За результатами моделювання гібридної імітаційної моделі (запропонованої у розділі 3.2.3) середній час дорівнює 3429 мс, а максимальний час затримки дорівнює 20126 мс (див. табл. 3.2). При використанні (*1,2,3*) патерну спостерігається підвищення готовності системи (див. табл. 3.3) з 90% до 96.5% (див. рис. 3.8б), а також зменшення середньої затримки з 4092 мс до 3429 мс.

Таблиця 3.2 – Значення затримок при використанні методу надлишкових читань

Конфігурація	1-1-3	1-2-3	1-3-3
Середнє значення, мс	4092	3429	4302
Максимальне значення, мс	38968	20126	12736

Розглядаючи систему за патерном (*k, s, n*), як (*1,3,3*): рівень узгодженості ONE, кількість опитуваних вузлів 3 від загальної кількості вузлів, загальна кількість вузлів 3. У відповідності до гібридної імітаційної моделі, при використанні цього патерну, середній час дорівнює 4302 мс, а максимальний час затримки дорівнює 12736 мс (див. табл. 3.2). При цьому спостерігається явище зростання готовності системи до 97.8% (див. табл. 3.3), але спостерігається деяке збільшення середньої затримки на 6% у порівнянні з конфігурацією (*1,1,3*) (див. табл. 3.2).

На рисунках 3.9-3.11 показана функція густини розподілу імовірності для конфігурацій системи (*1,1,3*), (*1,2,3*) та (*1,3,3*) відповідно. Якщо провести лінію тайм-ауту на рівні 90% процентилю для (*1,1,3*), (*1,2,3*) та (*1,3,3*) відповідно (див.

рис. 3.9-3.11), спостерігається зміщення середнього часу обслуговування запитів вправо та рух екстремальних затримок вліво.

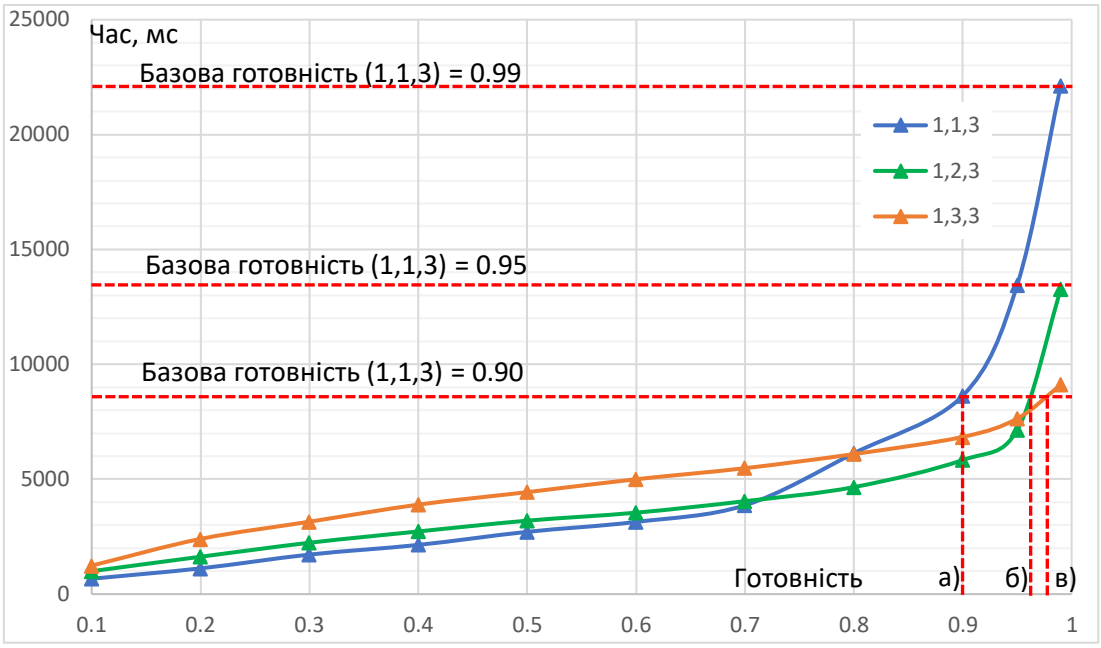


Рисунок 3.8 – Підвищення рівня готовності системи за рахунок використання надлишкових читань: а) рівень готовності системи для конфігурації (*1,1,3*); б) рівень готовності системи для конфігурації (*1,2,3*); в) рівень готовності системи для конфігурації (*1,3,3*)

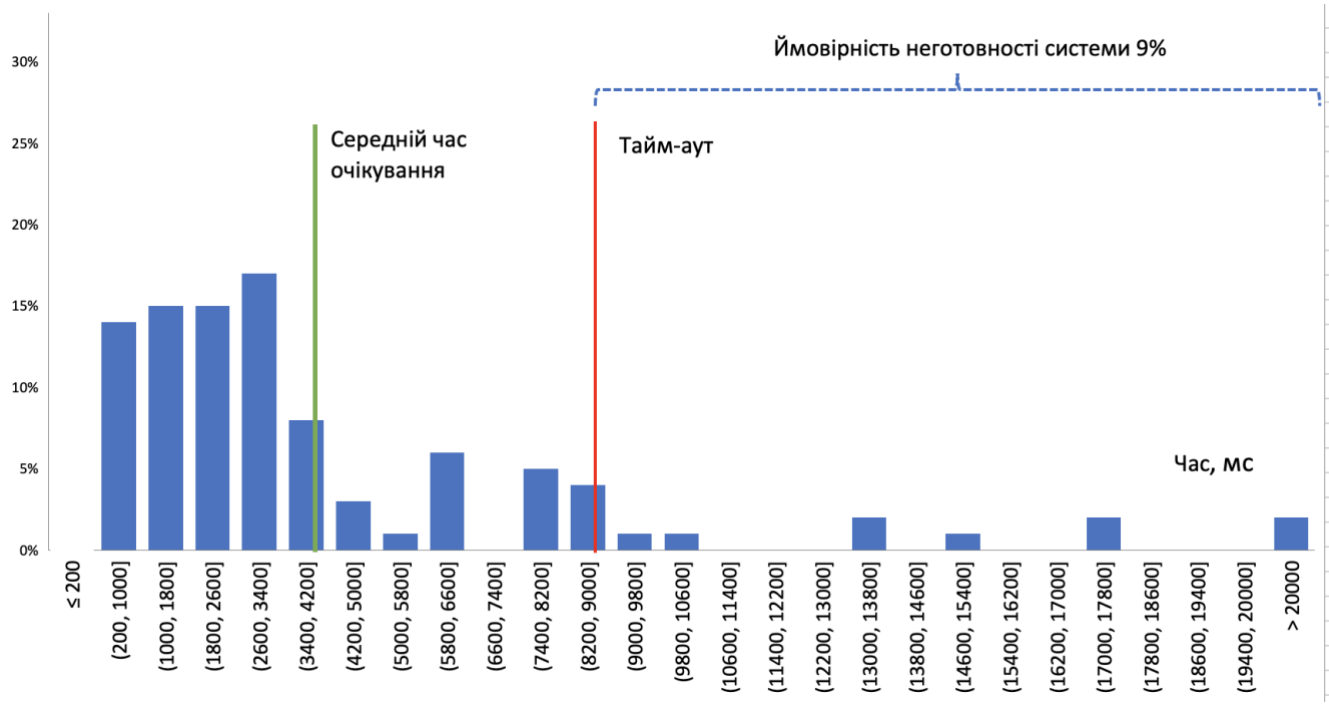


Рисунок 3.9 – Функція густини розподілу імовірності для конфігурації (*1,1,3*)

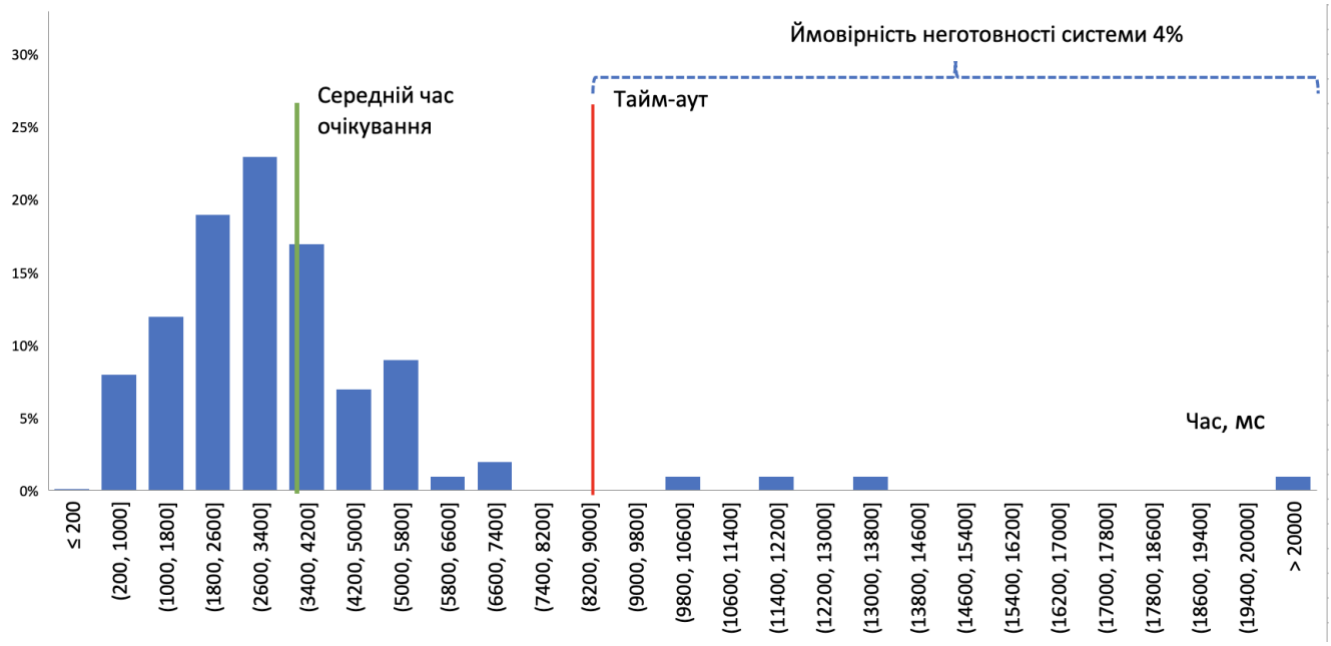


Рисунок 3.10 – Функція густини розподілу імовірності для конфігурації (*1,2,3*)

Рисунок 3.11 – Функція густини розподілу імовірності для конфігурації (*1,3,3*)

Таблиця 3.3 – Підвищення готовності системи у порівнянні з базовою конфігурацією (1,1,3)

Конфігурація	Готовність (в залежності від процентиля, що використовується в якості тайм-ауту)		
Базова (1,1,3)	90 %	95 %	99%
(1,2,3)	96.5% (+7%)	99,3 % (+5%)	99,96 % (+1%)
(1,3,3)	97.8%, (+9%)	99,92 % (+5%)	99,995 % (+1%)

3.3.3. Результати використання методу надлишкових читань для рівня узгодженості QUORUM з використанням гібридної моделі

Розглянемо конфігурацію глобально-розподіленого кластеру з трьома репліками, що використовує рівень узгодженості QUORUM. У відповідності до запропонованої теоретико-множинної моделі (див. розд. 3.2.1), така конфігурація описується трійкою (*k, s, n*), як (*2,2,3*): рівень узгодженості QUORUM, кількість опитуваних вузлів 2 від загальної кількості вузлів, загальна кількість вузлів 3. У такій системі у відповідності до результатів експериментального дослідження (див. розд. 2.4.1) середній час обслуговування запитів дорівнює 8328 мс, а максимальний час затримки дорівнює 42411 мс (див. табл. 3.4). Нехай, для такої системи там-аут встановлений на рівні 90% процентиль (див. рис. 3.12а), тобто 16780 мс.

Розглянемо система, яка описується патерном (*k, s, n*), як (*2,3,3*): рівень узгодженості QUORUM, кількість опитуваних вузлів 3 від загальної кількості вузлів, загальна кількість вузлів 3. У відповідності до гібридної імітаційної моделі, при використанні цього патерну, середній час дорівнює 7155 мс, а максимальний час затримки дорівнює 28812 мс (див. табл. 3.4). При цьому спостерігається явище зростання готовності системи з 90% до 97.5% (див. табл. 3.5), а також зменшення середньої затримки з 8328 мс до 7155 мс.

Таблиця 3.4 – Значення затримок при використанні методу надлишкових читань

Конфігурація	2-2-3	2-3-3
Середне значення, мс	8328	7155
Максимальне значення, мс	42411	28812

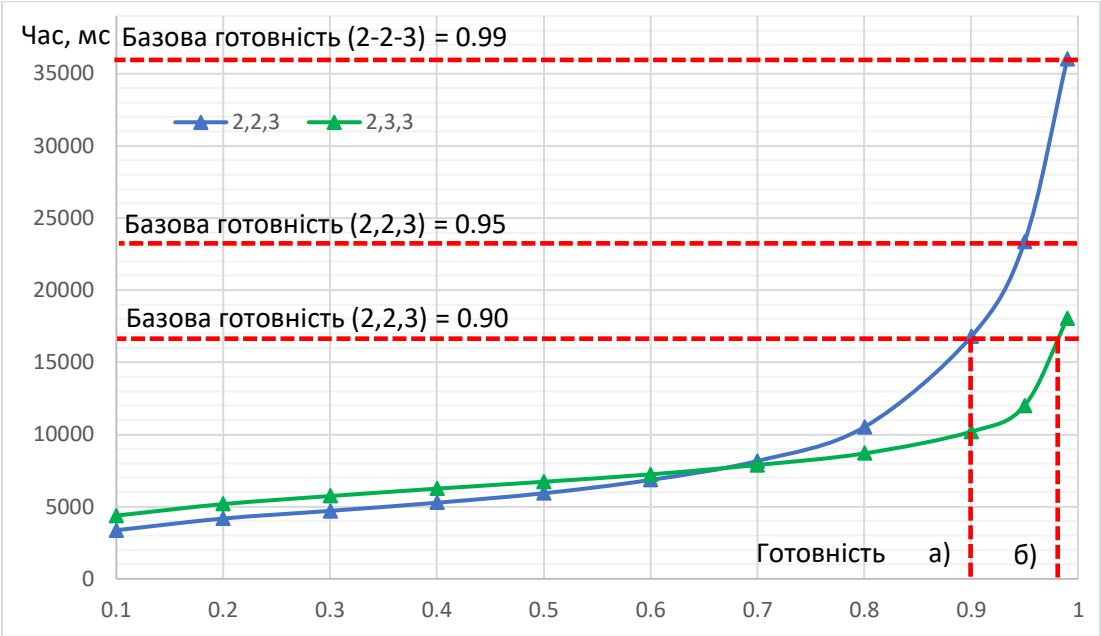


Рисунок 3.12 – Підвищення рівня готовності системи за рахунок використання надлишкових читань: а) рівень готовності системи для конфігурації (*2,2,3*); б) рівень готовності системи для конфігурації (*2,3,3*)

На рисунках 3.13-3.14 показана функція густини розподілу імовірності для конфігурацій системи (*2,2,3*) та (*2,3,3*) відповідно. Якщо провести лінію тайм-ауту на рівні 90% процентилю для (*2,2,3*) та (*2,3,3*) відповідно (див. рис. 3.13-3.14), спостерігається зміщення середнього часу обслуговування запитів вправо та рух екстремальних затримок вліво.

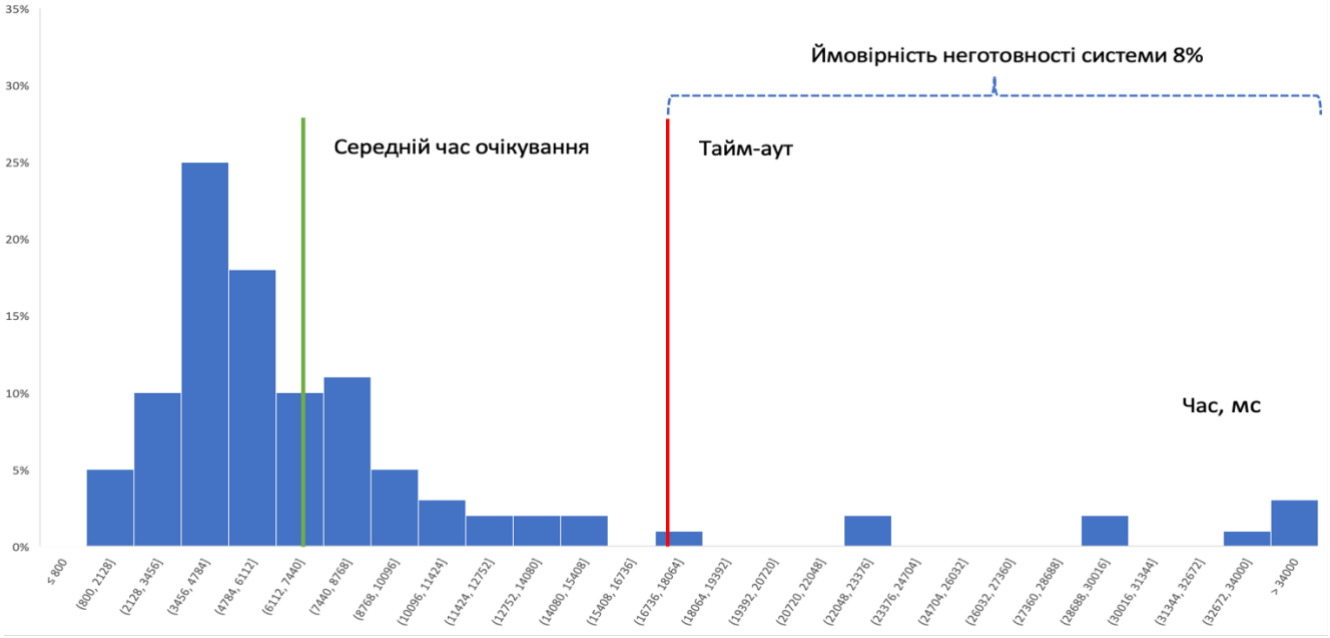


Рисунок 3.13 – Функція густини розподілу імовірності для конфігурації (*2,2,3*)

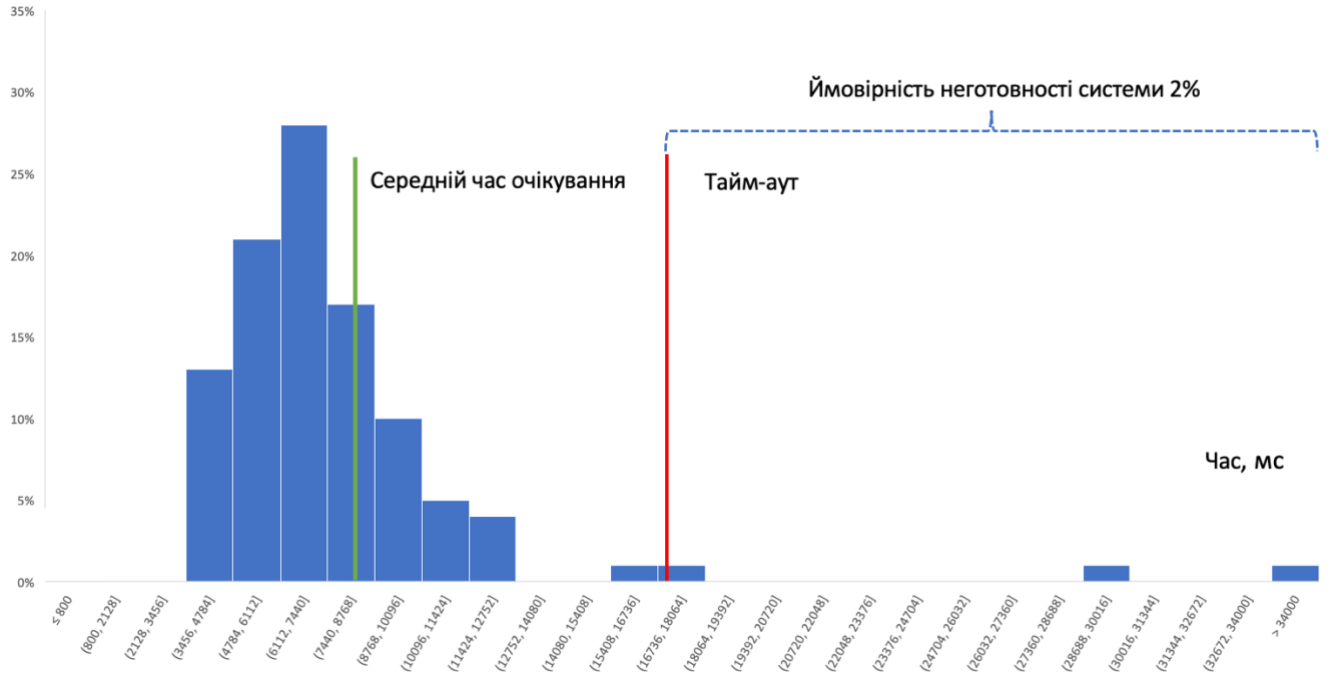


Рисунок 3.14 – Функція густини розподілу імовірності для конфігурації (*2,3,3*)

Таблиця 3.5 – Підвищення готовності системи у порівнянні з базовою конфігурацією (2,2,3)

Базова готовність конфігурація (2,2,3)	90 %	95 %	99%
Конфігурація (2,3,3)	97.5% (+8%)	99,84 % (+5%)	99,98 % (+1%)

3.4. Метод надлишкових читань для підвищення цілісності ГРРС

3.4.1. Структура методу

Метод надлишкових читань для підвищення цілісності глобально-розподілених реплікованих систем складається з наступних кроків представлених нижче.

1. *Отримання поточних налаштувань узгодженості (k) та кількісті вузлів у системі (n)*. В залежності від кількості вузлів у кластері зберігання даних і налаштувань сильної узгодженості, обираються можливі варіації налаштувань

надлишкових читань. З огляду на це, формується список можливих налаштувань відповідно до вимог кластеру.

2. *Розгортання глобально-розподіленого кластеру у реальному виробничому середовищі*. Розгортання хмарної інфраструктури за допомогою інструментів автоматизації, відповідно до патерну розгортання кластеру у хмарному середовищі, налаштувань узгодженості та кількості вузлів.

3. *Вибір конфігурації надлишкового читання*. Процес вибору конфігурації надлишкових читань, яка знижує ймовірність порушення цілісності відповідно до встановлених вимог. Механізм надлишкових читань може бути використаний для підвищення цілісності, як це було запропоновано у розд. 3.1.3. У такому випадку, система ініціює (*n-k)*-запитів (*n* – загальна кількість вузлів; *k* – рівень узгодженості) для всіх вузлів в системі. Після чого отримані відповіді від всіх вузлів, порівнюються між собою. На першому етапі проводиться порівняння часових міток, а на другому етапі виконується порівняння контрольних сум. У випадку, достатньої кількості вузлів-реплік проводиться мажоритарне голосування. Це порівняння дає змогу визначити, а у деяких випадках виправити можливі порушення даних у системі (див. розд. 3.1.3). Також може додатково ураховуватися збільшення часу обслуговування.

3.4.3. Оцінка забезпечення стійкості до порушень узгодженості

Розглянемо модель загроз при якій одна з реплік в процесі запису може бути скомпрометована, тобто порушена цілісність даних, при їх додаванні або оновленні. Тоді для системи з трьома вузлами-репліками використання надлишкових читань дозволяє зменшити ймовірність порушення цілісності наступним чином (див. табл. 3.6).

Якщо система використовує наступні параметри: рівень узгодженості операції запису – QUORUM, рівень узгодженості операції читання – QUORUM та кількість вузлів *n=3*. Операція запису при рівні узгодженості QUORUM виконується у два вузли системи. Допустимо, що перший вузол у системі має

спотворені/модифіковані дані. Якщо клієнт відправляє запит на читання, використовуючи рівень узгодженості QUORUM, то можливі наступні комбінації вузлів, які візьмуть участь у формуванні відповіді клієнту: перший та другий, другий та третій, перший та третій. У випадку залучення першого та другого вузлів, виконується порівняння відповідей за часовою міткою та контрольною сумою, яке дає змогу виявити факт порушення даних (відповіді не співпадають). За умови залучення першого та третього вузлів, відповідь першого вузла повернеться клієнту, тому що часова мітка даних першого вузла буде новішої ніж часова мітка третього вузла. У свою чергу, залучення другого та третього вузлів, відповідь другого вузла повернеться клієнту, тому що часова мітка буде новішої ніж часова мітка третього вузла.

Таблиця 3.6 – Оцінка забезпечення стійкості до порушень узгодженості

Конфігурації забезпечення сильної узгодженості		Конфігурація надлишкових читань	Ймовірність невизначеного порушення цілісності	Ймовірність виявлення (без виправлення) порушення цілісності	Ймовірність виправлення порушення цілісності
рівень узгоджен. операції запису	рівень узгоджен. операції читання				
ONE	ALL	3-3-3 (надлишкові читання відсутні)	1	метод надлишкових читань не працює	
QUORUM	QUORUM	2-2-3 (надлишкові читання відсутні)	0,33	0,33	0,33
QUORUM	ALL	2-3-3	0	1	0
ALL	ONE	1-1-3 (надлишкові читання відсутні)	0,33	0	0,66
ALL	QUORUM	1-2-3	0	0,66	0,33
ALL	ALL	1-3-3	0	0	1

У випадку використання методу надлишкових читань для забезпечення цілісності даних, спостерігається явище збільшення затримок, а отже зниження готовності. При використанні конфігурації (1,2,3) для розподіленого кластеру

затримки збільшуються у середньому на 11% (див. рис. 3.15), а для централізованого (див. рис. 3.17) на 8%. У випадку використання конфігурації (1,3,3) для розподіленого кластеру затримки збільшуються у середньому на 37% (див. рис. 3.16), а для централізованого (див. рис. 3.18) на 14%.

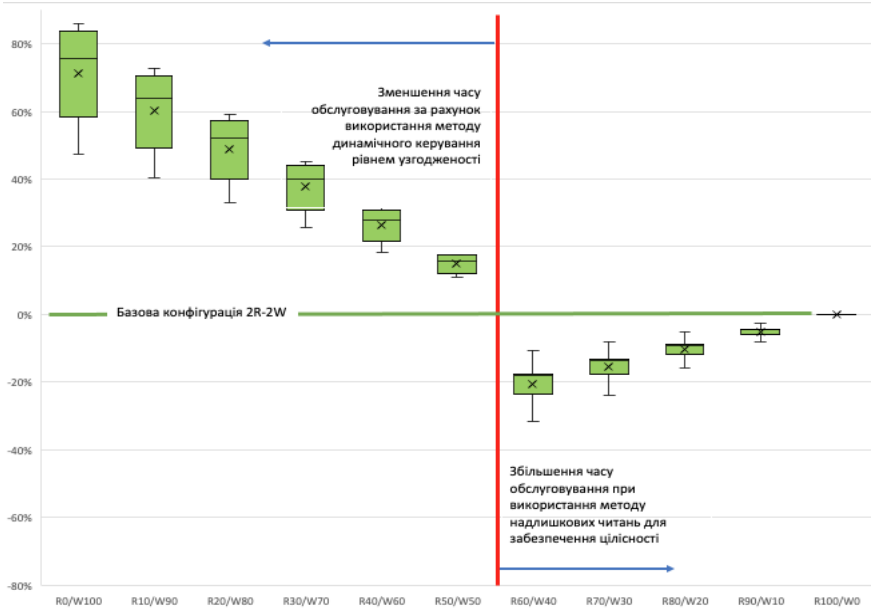


Рисунок 3.15 – Збільшення часу обслуговування запитів за рахунок використання методу надлишкових читань в залежності від пропорції операцій читання/запису у порівнянні з конфігурацією 2R-2W у розподіленому кластері за конфігурацією (1,2,3)

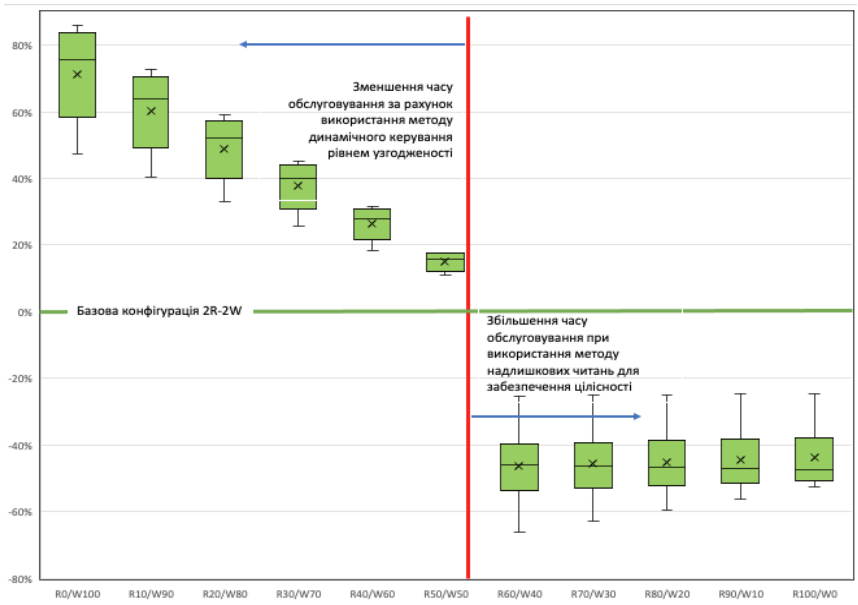


Рисунок 3.16 – Збільшення часу обслуговування запитів за рахунок використання методу надлишкових читань в залежності від пропорції операцій читання/запису у

порівнянні з конфігурацією 2R-2W у розподіленому кластері за конфігурацією (1,3,3)

Рисунок 3.17 – Збільшення часу обслуговування запитів за рахунок використання методу надлишкових читань в залежності від пропорції операцій читання/запису у порівнянні з конфігурацією 2R-2W у централізованому кластері за конфігурацією (1,2,3)

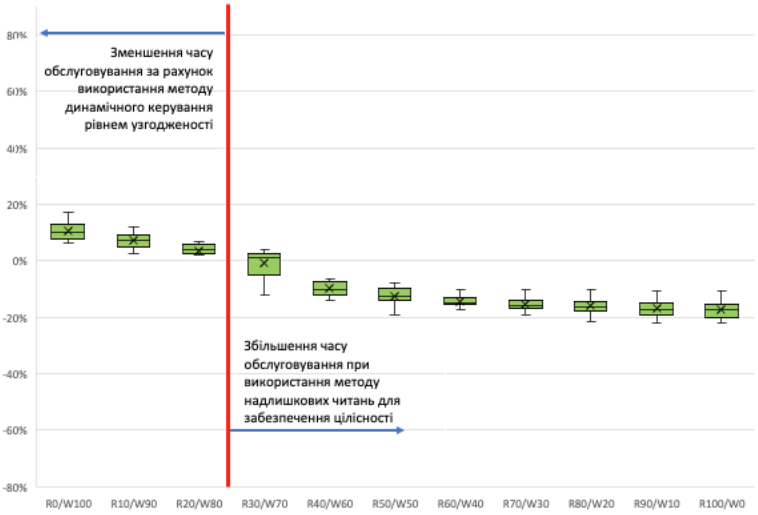


Рисунок 3.18 – Збільшення часу обслуговування запитів за рахунок використання методу надлишкових читань в залежності від пропорції операцій читання/запису у порівнянні з конфігурацією 2R-2W у централізованому кластері за конфігурацією (1,3,3)

3.5. Висновки до третього розділу

1. У цьому розділі вперше представлена модель конфігурації надлишкових читань, що дає змогу описати взаємозв’язок між рівнем узгодженості даних, кількістю опитаних реплік та загальною кількістю реплік ГРРС.

2. Вперше запропоновано гібридну імітаційну модель, яка дозволяє отримати щільності розподілу, мінімальний, максимальний та середній час обробки запиту, використовуючи експериментальні дані.

3. Удосконалено метод надлишкових читань глобально-розподілених реплікованих систем зберігання великих даних для підвищення готовності або цілісності.

4. Аналіз результатів використання методу надлишкових читань глобально-розподілених реплікованих систем зберігання великих даних показав, що підвищено готовність ГРРС при конфігурації (*1,1,3*) з 90% до 96.5 % (конфігурація (*1,1,3*)) та зменшено екстремальні часові затримки (див. табл. 3.2 та 3.4). У випадку використання методу для підвищення цілісності були отримані ймовірності виправлення та виявлення модифікування або спотворення даних (див. табл. 3.6). У той же час, використання метода надлишкових читань для підвищення цілісності, підвищує час обслуговування запитів, що знижує швидкодію, а отже і готовність системи (див. рис. 3.14-3.18).

РОЗДІЛ 4
ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ЗАБЕЗПЕЧЕННЯ КІБЕРБЕЗПЕКИ ГЛОБАЛЬНО-РОЗПОДІЛЕНИХ РЕПЛІЦІЙОВАНИХ СИСТЕМ ЗБЕРІГАННЯ ВЕЛИКИХ ДАНИХ

4.1. Інформаційна технологія забезпечення кібербезпеки глобально-розподіленої реплікованої системи зберігання даних

4.1.1. Контексти застосування наукових результатів

Інформаційна технологія (ІТ) визначає методи, прийоми засоби обчислювальної техніки для підтримки розроблених моделей та методів. Інформаційна технологія відображає практичне впровадження у визначеному контексті їх використання, який визначає суб’єкт та об’єкт використання тощо. На рисунку 4.1 представлена систематизація наукових результатів з урахуванням можливих контекстів їх використання.

За суб’єктом використання розроблених та удосконалених методів та моделей можна виокремити провайдера хмарних послуг (AWS, MS Azure тощо), розробник ГРРС та клієнт ГРРС.

За об’єктом використання можна виділити глобально-розподілені системи зберігання даних.

За часом використання можна виділити контекст розробки(створення) та експлуатації.

За типом віршувальних задач можна виділити контекст аналізу та синтезу.

Таким чином, було виділено вісім контекстів практичного впровадження наукових результатів, які визначають умови, послідовність та інші особливості їхнього спільного використання.

Для інформаційної підтримки запропонованих моделей та методів були виділені контексти. Виходячи з цього розроблено інструментальні засоби, які підтримують базовий функціонал.

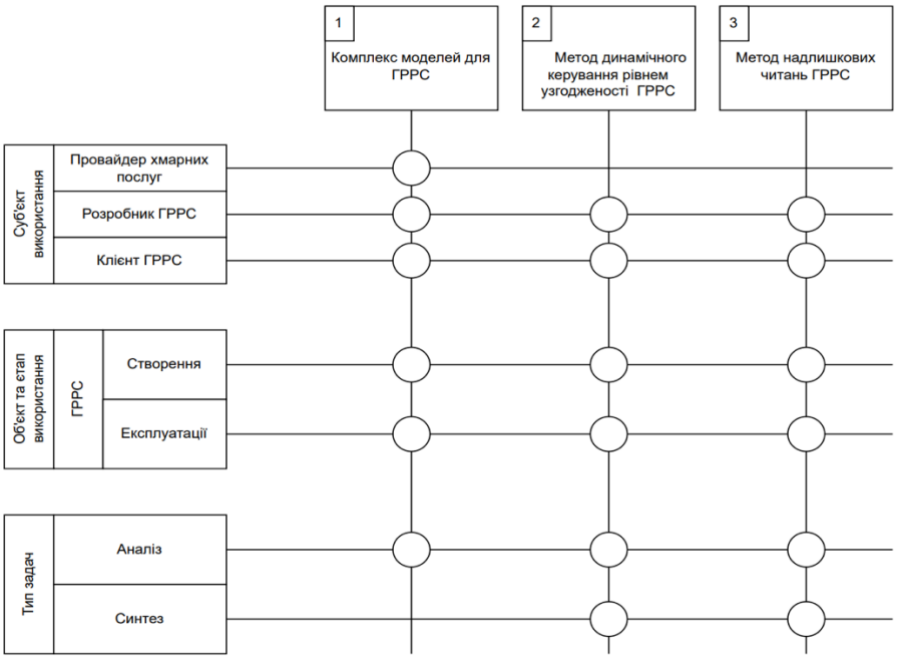


Рисунок 4.1 – Контексти використання наукових результатів

4.1.2. Структура інструментальних засобів

Інструментальні засоби є ключовою складовою ІТ і направлені на підтримку, автоматичне та автоматизоване виконання функцій ІТ. Для практичного впровадження результатів дисертаційного дослідження була ініційована робота над створенням інструментарію забезпечення кібербезпеки глобально-розподілених реплікованих систем зберігання даних. Інструментарій включає в себе чотири програмних компонентів (frameworks, FW):

– FW1: інформаційний застосунок розгортання глобально-розподіленої реплікованої системи зберігання даних у хмарному середовищі;

– FW2: інформаційний застосунок навантажувального тестування глобально-розподіленої реплікованої системи зберігання даних;

– FW3: інформаційний застосунок підтримки аналізу налаштувань глобально-розподіленої реплікованої системи зберігання;

– FW4: інформаційний застосунок безперервного моніторингу та корекції налаштувань глобально-розподіленої реплікованої системи зберігання.

Вихідний код інструментарію представлений у додатку Б. Програмні компоненти інструментарію FW1 – FW2 представляють собою інформаційні

засоби, які є надбудовою для існуючих додатків, у свою чергу, FW3 – FW4 представляють собою консольні додатки і не використовують сторонні компоненти.

Взаємозв’язок між інформаційними засобами та результатами дисертаційного дослідження представлено на рис. 4.2.

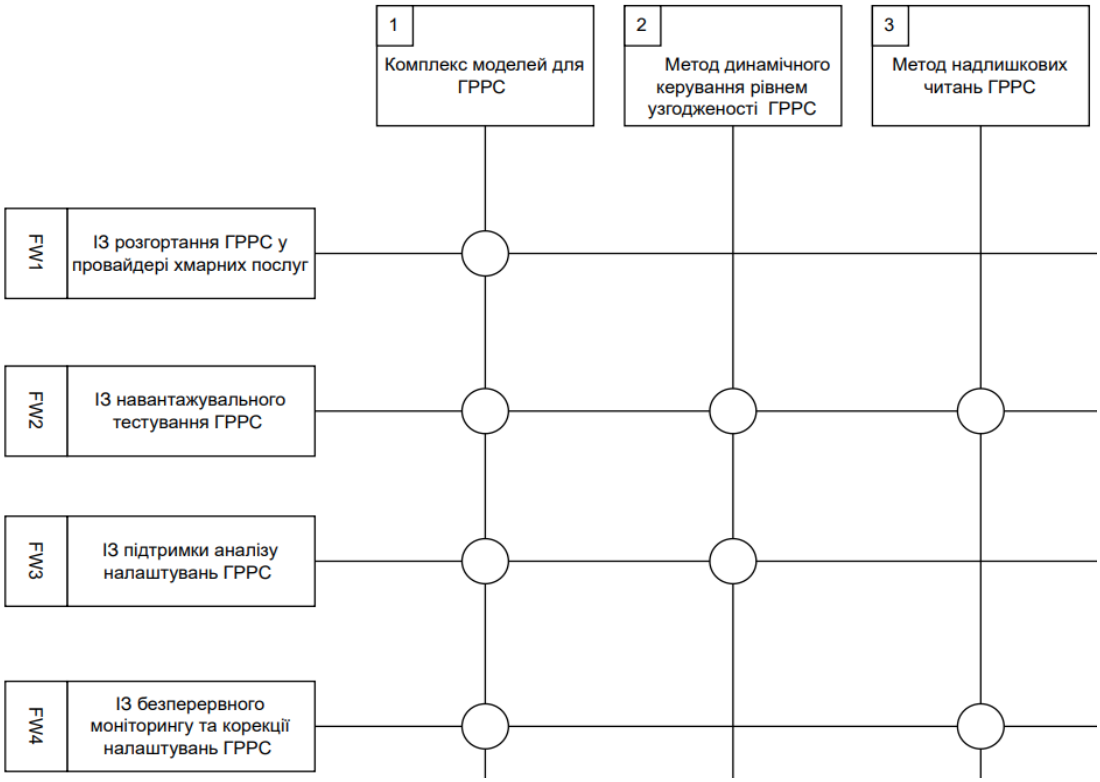


Рисунок 4.2 – Взаємозв’язок між науковими результатами та інструментальними засобами

4.1.3. Платформа розгортання та інтеграції інструментального засобу

При розробці інструментального засобу розгортання, тестування, моніторингу та аналізу глобально-розподіленої реплікованої системи зберігання даних були використані наступні принципи та концепції:

– кросплатформність, на основі використання ansible, terraform, python;

– відкритості, за основу були взяті програмні компоненти з відкритим вихідним кодом;

– розширюваності, інструментальний засіб можна легко розширити додавши нові вимоги або модифікувати існуючи.

Для реалізації програмних компонентів FW1 – FW4 була використане середовище розробки Visual Studio Code з необхідними плагінами для консольних програм, як ansible, terraform та python. Розгортання хмарної інфраструктури проводиться за допомогою генерації файлів з описом інфраструктури за допомогою мови HCL на основі патерна розгортання (див. розділ 2.2). Налаштування глобально-розподіленої системи зберігання даних виконується за допомогою yaml інструкцій для програми управління конфігураціями ansible. У навантажувальному тестуванні беруть участь ansible та YCSB бенчмарк. Всі вище перелічені програмні засоби є програмами з відкритим кодом. Програмні компоненти аналізу та моніторингу налаштувань написані за допомогою мови програмування python та представлені у додатку Б.

UML діаграма прецедентів інструментального засобу представлена на рис. 4.3.

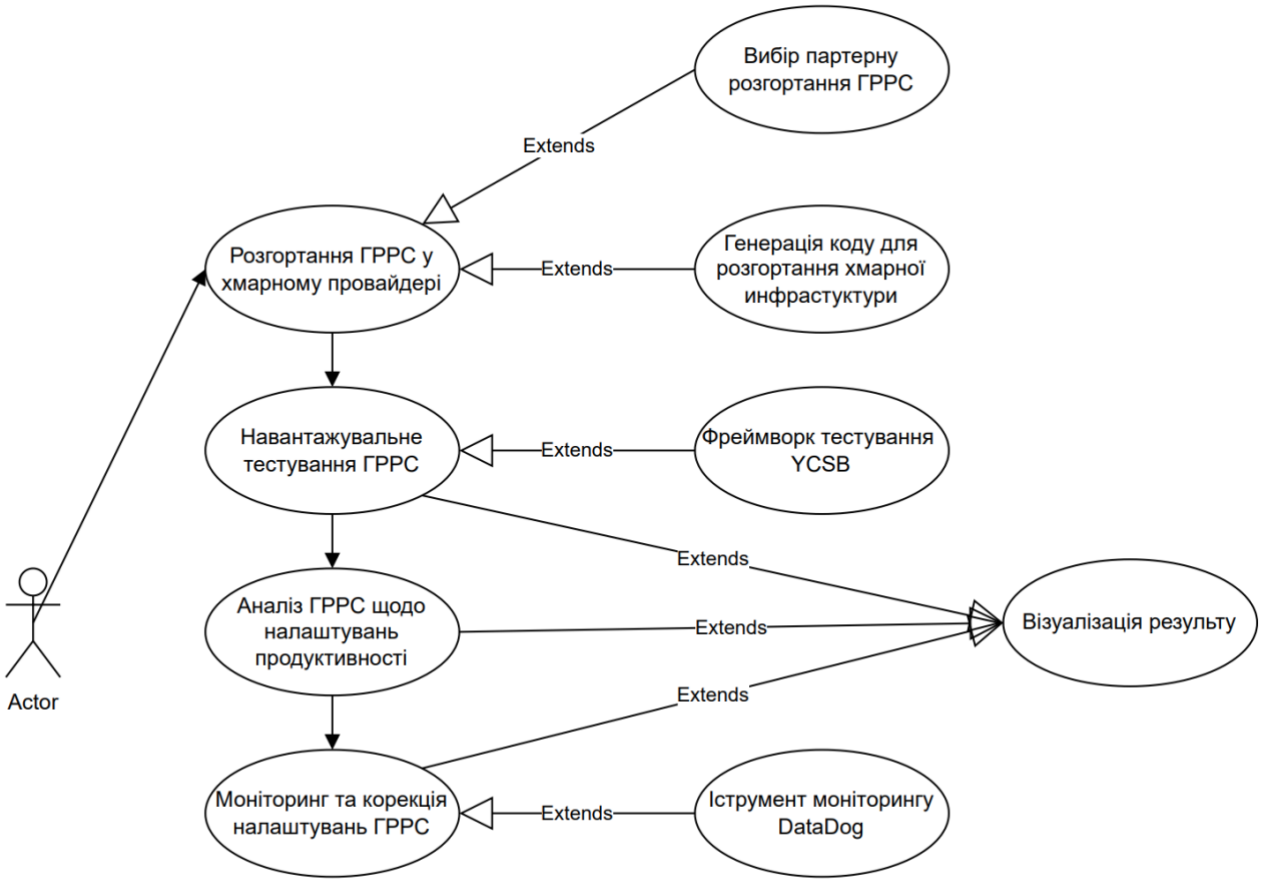


Рисунок 4.3 – Діаграма прецедентів

Таблиця 4.1 – Роз’яснення UML діаграми прецедентів інформаційного засобу розгортання, тестування, моніторингу та аналізу глобально-розподілених реплікованих систем зберігання даних

№	Прецедент	Роз’яснення
1.	Розгортання ГРРС у хмарному провайдері	Процес розгортання хмарної інфраструктури за допомогою інструменту terraform.
2.	Навантажувальне тестування ГРРС	Процес навантажувального тестування ГРРС для отримання часових затримок за допомогою інструменту ansible.
3.	Аналіз ГРРС щодо налаштувань продуктивності	Процес аналізу часових показників ГРРС з побудовою графіків залежності від рівня узгодженості.
4.	Моніторинг та корекція налаштувань ГРРС	Процес аналізу та корекції налаштувань ГРРС у процесі експлуатації системи.
5.	Вибір патерну розгортання ГРРС	Процес вибору патерну розгортання ГРРС у хмарному середовищі в залежності від вимог та охоплення користувачів.
6.	Генерація коду для розгортання хмарної інфраструктури	Генерація опису хмарної інфраструктури написаного на мові HCL для інструменту terraform.
7.	Фреймворк тестування YCSB	Інформаційний застосунок тестування ГРРС, представлений у вигляді фреймворку.
8.	Інструмент моніторингу DataDog	Інструмент моніторингу хмарної інфраструктури, застосунків.
9.	Візуалізація результату	Відображення результатів роботи.
Примітка № – порядковий номер у таблиці.		

4.1.4. Елементи інформаційних засобів

Інформаційні засоби забезпечення кібербезпеки глобально-розподілених систем зберігання великих даних на основі контролюючої узгодженості розроблена на основі IDF0 діаграми.

На рисунку 4.4 зображено контекстну IDF0 діаграму, що ілюструє загальний вигляд розробленої інформаційної технології.

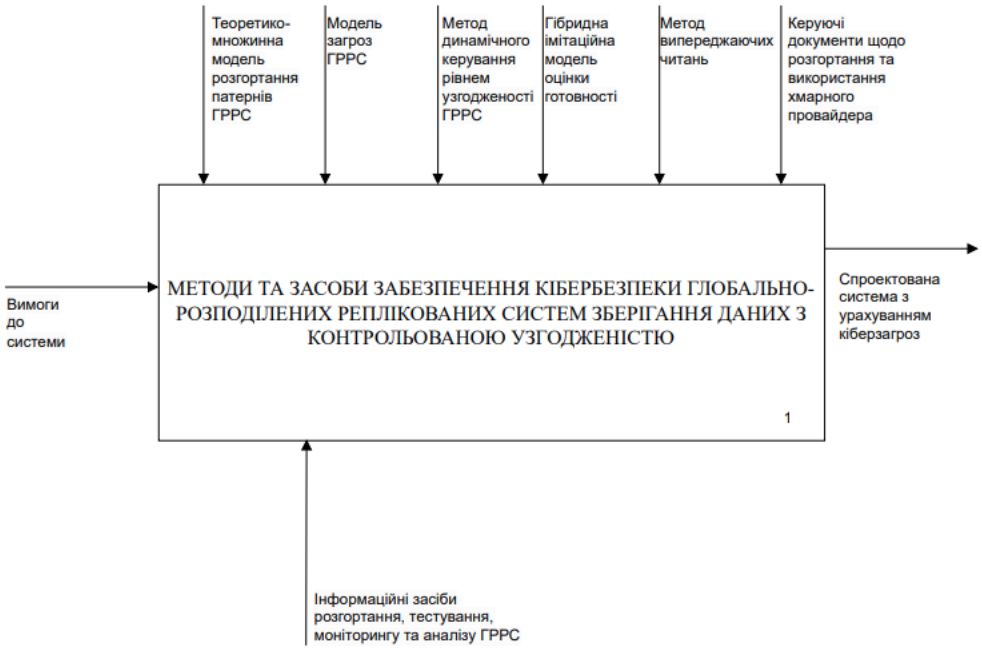


Рисунок 4.4 – Контекстна IDF0 діаграми інформаційної технології

Вхідні інформаційні потоки (горизонтальні стрілки):

– вимоги до системи – це технічне завдання, неформальні вимоги до ГРРС.

Вихідні інформаційні потоки:

– спроектована система з урахуванням кіберзагроз – це налаштована глобально-розподілена система зберігання даних з урахуванням рівня необхідного кібербезпеки.

На рисунку 4.5 зображена IDF0 діаграма першого рівня.

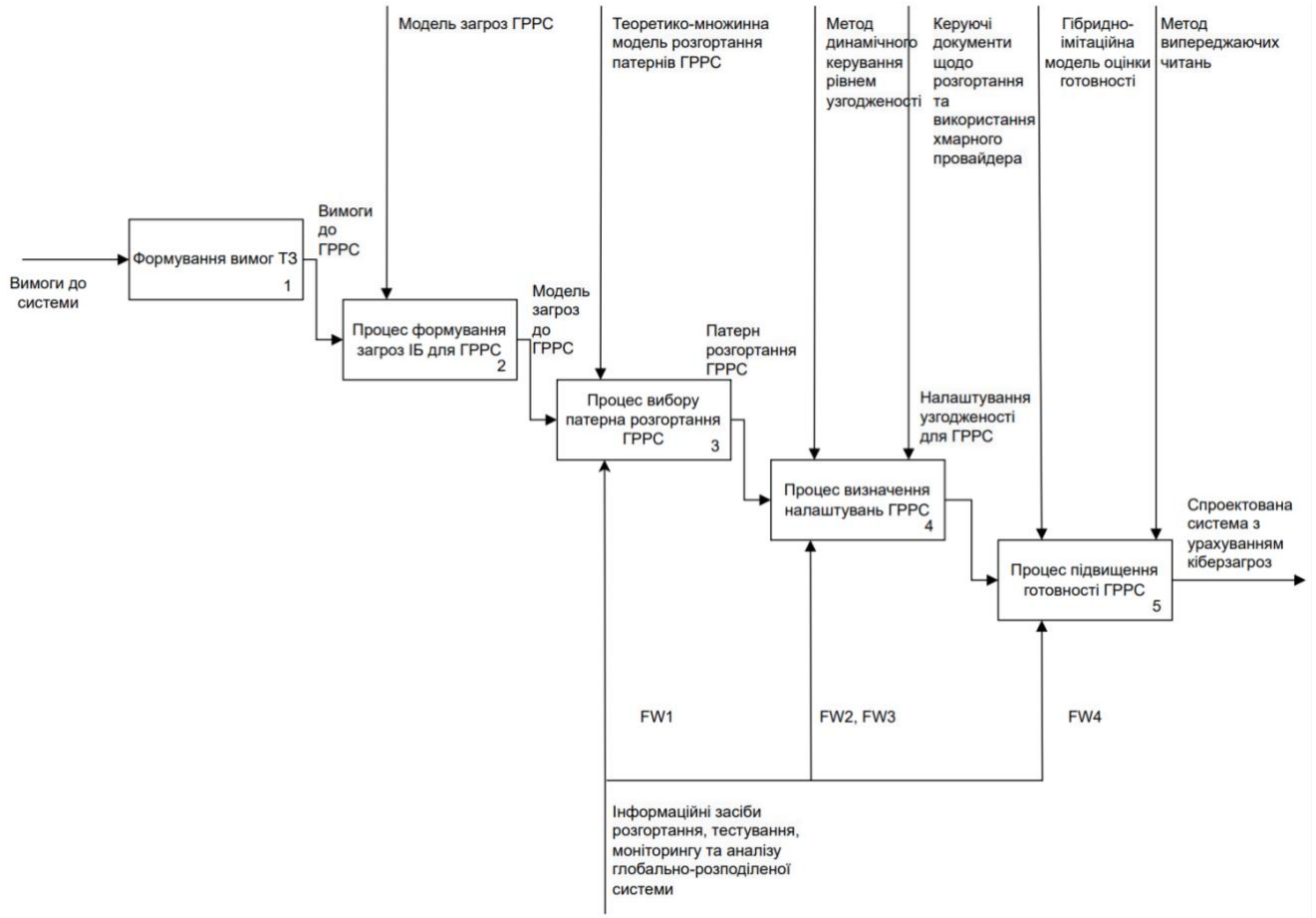


Рисунок 4.5 – IDF0 діаграма першого рівня деталізації

Проміжні інформаційні потоки (горизонтальні стрілки):

– вимоги до ГРРС – формальні та неформальні вимоги, технічне завдання до глобально-розподіленої системи зберігання даних з огляду на вимоги системи;

– модель загроз до ГРРС – опис методів і засобів здійснення шкідливих дій (загроз) порушення кібербезпеки ГРРС;

– патерн розгортання ГРРС – архітектура розподіленої системи зберігання даних на основі вимог до ГРРС;

– налаштування узгодженості для ГРРС – отримані налаштування глобально-розподіленої системи зберігання даних, які дозволяють забезпечити необхідний рівень кібербезпеки.

Функції (процеси) обробки інформаційних потоків (вертикальні стрілки направлені зверху-вниз):

– формування вимог – процес формування вимог до архітектури, швидкодії, готовності, цілісності та інформаційної безпеки (ІБ) на основі керуючих документів і пропозицій замовника;

– процес формування загроз інформаційної безпеки – процес формування моделі загроз на основі метода STRIDE та загальнодоступних реєстрів загроз;

– процес вибору патерну розгортання ГРРС – процес вибору архітектури ГРРС на основі клієнтських вимог, технічного завдання та кібербезпеки.

– процес визначення налаштувань ГРРС – процес розрахунку та аналізу параметрів ГРРС у відповідності до вимог;

– процес підвищення готовності ГРРС – неперервний процес аналізу налаштувань та готовності ГРРС та корекція налаштувань системи.

Інформаційний засіб розгортання, тестування, моніторингу та аналізу ГРРС (вертикальні стрілки направлені знизу-вгору):

– FW1: інформаційний застосунок розгортання глобально-розподіленої реплікованої системи зберігання даних у хмарному середовищі;

– FW2: інформаційний застосунок навантажувального тестування глобально-розподіленої реплікованої системи зберігання даних;

– FW3: інформаційний застосунок підтримки аналізу налаштувань глобально-розподіленої реплікованої системи зберігання;

– FW4: інформаційний застосунок безперервного моніторингу та корекції налаштувань глобально-розподіленої реплікованої системи зберігання.

4.2. Практичне впровадження результатів дисертаційного дослідження

4.2.1. Аналіз результатів впровадження

Результати дисертаційної роботи було впроваджено у (див. додаток В):

– у ТОВ «Софтсерв Інновації» (акт впровадження від 28 серпня 2023) при розробці підсистеми виконання асинхронних завдань в рамках e-commerce системи;

– у Leeds Beckett University (акт впровадження від 20 липня 2022) при формуванні курсу «Хмарних обчислень» та у формуванні магістерських тем в рамках GrEen Networking;

– у НДР ХАІ (акт впровадження від 16 березня 2023) при науково-дослідній роботі за темами «Методологічні засади та технології оцінювання та забезпечення безпеки (захисту) критичних інформаційних інфраструктур» (№ ДР 0119U100979, 2019-2021) та «Методи, моделі та інформаційні технології підвищення надійності та безпечності складних ІТ-систем на етапах розроблення та впровадження» (№ Д/Р 0121U113842, 2021-2023).

У табл. 4.2 наведені організації від яких були отримані наукові результати. Стовбці таблиці містять характеристики отриманих результатів згідно з тим, як це відображено в актах впровадження:

– галузь – характеризує область діяльності організації в який були впроваджені результати;

– системи, процеси та продукти – вказує вид систем, процесів та продуктів при реалізації яких були впроваджені результати;

– використані наукові результати – характеризує набір результатів дисертаційного дослідження впроваджених у організаціях; результати пронумеровані у тому порядку, як це зроблено у розділі «Вступ» цієї роботи;

– ефект – якісні та кількісні характеристики поліпшення у продуктах та процесах організацій внаслідок впровадження зазначених результатів дисертаційного дослідження.

Таблиця 4.2 – Відомості про впровадження результатів дисертаційного дослідження

Організація	Газуль	Системи, процеси, продукти	Використані наукові досягнення	Ефект
ТОВ «Софтсерв Технології»	ІТ	інформаційна система підтримки асинхронних завдань	1,2,3	при використані послідовно методів 2 та 3 підвищено готовність (з 99,2 % до 99,63 %) та оперативність (на 56-62 %) системи
Національний аерокосмічний університет «ХАІ»	Освіта	НДР, навчальні курси	1,2,3	Підвищення фундаментальності та практичної спрямованості навчального процесу, підвищення якості наукових та академічних проектів
Leeds Beckett University	Освіта	навчальні курси	2,3	Підвищення фундаментальності та практичної спрямованості навчального процесу, підвищення якості наукових та академічних проектів

4.2.2. Приклад впровадження результатів дисертаційного дослідження для системи підтримки виконання асинхронних завдань

У ТОВ «СофтСерв Інновації» було впроваджено результати дисертаційного роботи при проектуванні підсистеми підтримки асинхронних завдань у рамках e-commerce системи (див. рис. 4.6). E-commerce система обробки замовлень розгорнута у системі оркестрації контейнерів Kubernetes [90] у вигляді наступних компонентів:

– застосунок web – виконує функцію інтерфейсу між системою та користувачем;

– застосунок order – виконує функцію замовлення товарів;

– застосунок delivery – виконує функцію формування відправлення;

– застосунок web-temporal – сторонній компонент Temporal [91], виконує функцію візуалізації асинхронних завдань;

– застосунок worker-temporal – сторонній компонент Temporal, виконує функцію опрацювання асинхронних завдань;

– застосунок service-temporal – сторонній компонент Temporal, виконує функцію оповіщення застосунків про завершення асинхронного завдання.

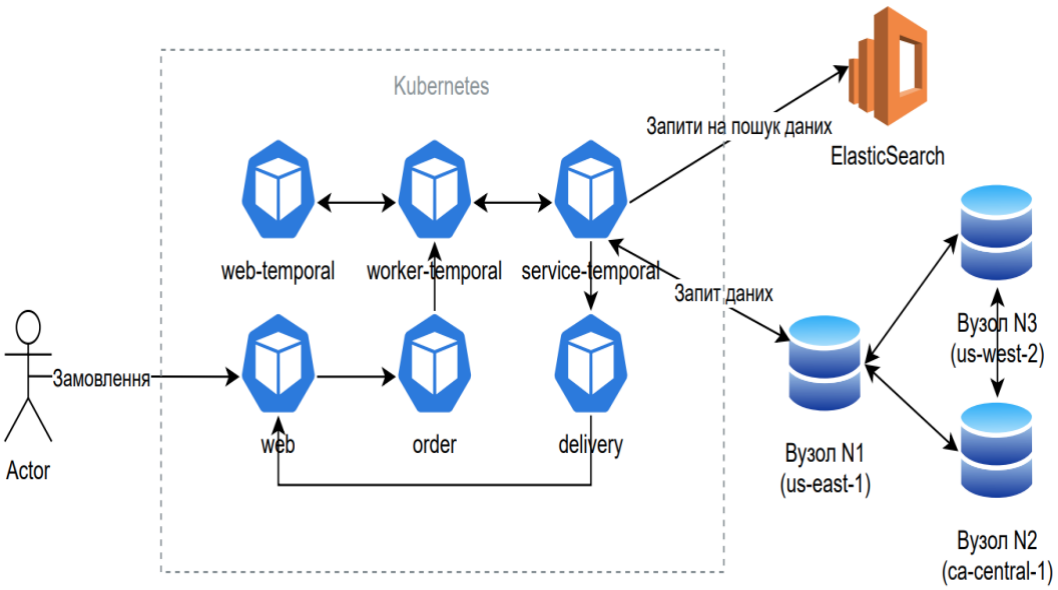


Рисунок 4.6 – Узагальнена архітектура e-commerce системи

Також Temporal у вигляді розгортання представленого на рисунку 4.6, використовує наступні компоненти поза межами Kubernetes [92]:

– ElasticSearch [93] – сервіс AWS з відкритим вихідним кодом, який виконує функції пошуку, вибірки та фільтрації текстових даних.

– Cassandra – виконує функцію глобально-розподіленої системи зберігання асинхронних завдань, розгорнуте у вигляді віртуальних машин AWS EC2 [94].

Особливу увагу при проектуванні e-commerce системи приділяється надійності та розподіленості для охоплення ринку товарів та послуг у США та Канаді. Важливим фактом є те що система зорієнтована на переважну більшість запитів на читання ніж на виконання операцій запис.

Застосування метода динамічного керування рівнем узгодженості (див. розділ 2) дало змогу зменшити максимальну затримку на 25,3% та підвищити готовність системи до 99,37%. У свою чергу, застосування методу надлишкових читань (див. розділ 3) зменшив максимальну затримку до 15,8 % та підвищити готовність до 99,63%. У таблиці 4.3 представлені початкові та вихідні показники (після застосування методів).

Таблиця 4.3 – Підвищення готовності та швидкодії розгорнутої глобально-розподіленої e-commerce системи

Етап	Максимальна затримка, мс	Готовність, %
Характеристики системи після її початкового розгортання	85689.2	99,2
Характеристики системи після застосування Методу динамічного керування рівнем узгодженості	64009.83	99,37
Характеристики системи після послідовного застосування методу динамічного керування рівнем узгодженості та методу надлишкових читань	53896.27	99,63

4.3. Висновки до четвертого розділу

1. У цьому розділі розроблена структура інформаційної технології підвищення кібербезпеки глобально-розподілених реплікованих систем зберігання даних, яка ґрунтується на розроблених та запропонованих у попередніх розділах моделях та методів.

2. У рамках інформаційної технології було розроблено:

– інформаційний застосунок розгортання глобально-розподіленої реплікованої системи зберігання даних у провайдері хмарних послуг (FW1), що використовує terraform та ansible для автоматизації процесів налаштування;

– інформаційний застосунок навантажувального тестування глобально-розподіленої реплікованої системи зберігання даних (FW2), що використовує YCSB фреймворк для автоматизації процесів навантажувального тестування;

– інформаційний застосунок підтримки аналізу налаштувань глобально-розподіленої реплікованої системи зберігання (FW3), що дозоляє отримати дані залежності кількості активних підключень, затримок та рівнем узгодженості;

– інформаційний застосунок безперервного моніторингу та корекції налаштувань глобально-розподіленої реплікованої системи зберігання (FW4), що дозволяє отримати безперервно дані для подальшої корекції налаштувань ГРРС.

3. Результати дисертаційного дослідження були впроваджені у освітніх закладах та ІТ компанії. Приклад реалізації результатів дисертаційного дослідження у ТОВ «Софтсерв Іннофвації» предстане у розділі 4.2.2 .

ВИСНОВОК

1. У дисертаційній роботі вирішена актуальна наукова-прикладна задача підвищення кібербезпеки глобально-розподілених реплікованих систем зберігання даних.

2. Розроблено комплекс нових та удосконалених моделей для глобально-розподілених систем зберігання даних, які забезпечують: 1) деталізацію загроз кібербезпеки ГРРС; 2) формалізацію опису патернів розгортання ГРРС у хмарному середовищі з урахуванням доменів готовності за допомогою апарату теоретико-множинного представлення; 3) оцінювання готовності та зменшення часу обслуговування в умовах кібератак завдяки використанню механізму надлишкових читань за допомогою гібридного імітаційного підходу.

3. Вперше запропоновано метод динамічного керування рівнем узгодженості ГРРС, який, на відміну від відомих, базується на побудові доменів змішаного робочого навантаження та дозволяє підвищити готовність системи, гарантуючи при цьому строгу узгодженість даних для підвищення стійкості до DDoS атак.

4. Удосконалено метод надлишкових читань ГРРС, який ґрунтується на використанні надлишковості щодо встановленого рівня узгодженості операцій читання та дозволяє зменшити екстремальні часові затримки та підвищити готовність при встановленому обмеженні на час обслуговування або цілісність в умовах кіберзагроз порушення даних та відмов в обслуговуванні.

5. Достовірність отриманих наукових і практичних результатів підтверджуються обґрунтованістю допущень, прийнятих при розробці моделей та методів; співпадінням з допустимою точністю результатів аналітичного та імітаційного моделювання при ідентичних структурах та параметрах глобально-розподілених реплікованих систем зберігання даних з результатами практичних випробувань та навантажувального тестування.

6. Практичне значення отриманих результатів в рамках дисертаційного дослідження полягають в тому, що основні положення даної роботи реалізовані у вигляді методів, моделей та відповідних програмних засобів, які лежать в основи

інформаційної технології підвищення кібербезпеки глобально-розподілених реплікованих систем зберігання даних. Розроблено програмну реалізацію гібридної імітаційної моделі для оцінки методу надлишкових читань.

7. Виконана у дисертаційному дослідженні кількісна оцінка ефективності запропонованих методів, яка підтверджена результатами практичного впровадження на прикладі системи e-commerce, свідчить про їхню перевагу у порівнянні зі статичними налаштуваннями ГРРС. Підвищено готовність системи з 99,2% до 99,63% при послідовному застосувані запропонованих методів; підвищено швидкодію на 63%, що дозволяє кратно збільшити стійкість до DDoS атак; також забезпечено стійкість до порушень цілісності однієї з трьох реплік системи.

8. Результати дисертаційного дослідження впроваджено у ТОВ «Софтсерв Інновації» (акт впровадження від 28 серпня 2023) при розробці підсистеми виконання асинхронних завдань в рамках e-commerce системи, у Leeds Beckett University (акт впровадження від 20 липня 2022), а також при виконанні науково-дослідних робіт за держбюджетними темами «Методологічні засади та технології оцінювання та забезпечення безпеки (захисту) критичних інформаційних інфраструктур» і «Методи, моделі та інформаційні технології підвищення надійності та безпечності складних ІТ-систем на етапах розроблення та впровадження» (акт впровадження від 16 березня 2023).

9. Подальші дослідження слід проводити у напрямку вдосконалення методів, моделей та інформаційних засобів підвищення кібербезпеки глобально-розподілених реплікованих систем зберігання даних на основі контролюючої узгодженості, зокрема, вирішення задач оптимального вибору та розміщення вузлів-реплік у хмарному середовищі для забезпечення більшої стійкості до можливих кіберзагроз.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. M. van Steen та A. Tanenbaum, Distributed Systems, 3rd ed., 2017.
2. S. Gilbert та N. Lynch, «Brewer’s Conjecture and the Feasi- bility of Consistent, Available, Partition-Tolerant Web Services,» т. 33, № 2, 2002.
3. D. Abadi, «Consistency tradeoff in modern distributed database system design,» т. 45, № 2, 2002.
4. A. Avizienis, J. Laprie, B. Randell та C. Landwehr, «Basic concepts and taxonomy of dependable and secure computing,» *IEEE Transactions on Dependable and Secure Computing*, т. 1, № 1, pp. 11-33, 2004.
5. A. Gorbenko, A. Romanovsky, O. Tarasyuk, V. Kharchenko та S. Mamutov, «Exploring Uncertainty of Delays as a Factor in End-to- End Cloud Response Time,» в *9th European Dependable Computing Conference (EDCC’2012)*, Sibiu, Romania, 2012.
6. R. Potharaju та N. Jain, «When the Network Crumbles: An Empirical Study of Cloud Network Failures and their Impact on Services,» в *4th ACM Symposium on Cloud Computing (SOCC'2013)*, Santa Clara, CA, 2013.
7. C. Scott, D. Choffnes та I. Cunha, «LIFEGUARD: practical repair of persistent route failures,» в *ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, New York, USA, 2012.
8. A. Gorbenko, V. Kharchenko, O. Tarasyuk, Y. Chen та A. Romanovsky, «The threat of uncertainty in Service-Oriented Architecture,» в *RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems*, Newcastle, UK, 2008.
9. A. Gorbenko, V. Kharchenko, O. Tarasyuk та A. Furmanov, «F(I)MEA-Technique of Web Services Analysis and Dependability Ensuring,» *Rigorous Development of Complex Fault-Tolerant Systems*, pp. 153-167, 2006.
10. A. Gorbenko, A. Karpenko та O. Tarasyuk, «Analysis of trade-offs in fault-tolerant distributed computing and replicated databases,» в *2020 IEEE 11th*

International Conference on Dependable Systems, Services and Technologies, Kyiv, 2020.

11. A. Gorbenko, A. Karpenko та O. Tarasyuk, «Performance evaluation of various deployment scenarios of the 3-replicated Cassandra NoSQL cluster on AWS,» *Radioelectronic and computer systems,* т. 4, № 100, pp. 157-165, 2021.
12. А. Карпенко, О. Тарасюк та А. Горбенко, «Дослідження узгодженості та продуктивності у нереляційних реплікованих базах даних,» *Сучасні інформаційні системи,* т. 5, № 3, pp. 66-75, 2021.
13. О. Тарасюк, А. Горбенко та А. Карпенко, «Розвиток архітектури, теорія та моделі властивостей розподілених систем зберігання даних,» *Вимірювальна та обчислювальна техніка в технологічних процесах,* № 2, pp. 5-13, 2022.
14. J. Ahmed, A. Karpenko, O. Tarasyuk, A. Gorbenko та A. Sheikh-Akbari, «Consistency issue and related trade-offs in distributed replicated systems and databases: a review,» *Radioelectronic and Computer Systems,* т. 2, № 106, pp. 171-179, 2022.
15. Oracle Cloud Infrastructure, «The Evolution of Big Data and tge Future of the Data Lakehouse,» [Онлайновий]. Available: https://www.oracle.com/big-data/what-is-big-data/. [Дата звернення: 29 вересня 2019].
16. D. Laney, «3D Data managment: Controlling Data Volume, Velocity, and Variety,» 2001.
17. AnalytixLabs, «What Are the Key Characteristics of Big Data?,» [Онлайновий]. Available: https://www.analytixlabs.co.in/blog/characteristics-of-big-data/. [Дата звернення: 14 листопада 2019].
18. B. Medjahed та M. Ouzzani, «Generalization of ACID properties,» 2009.
19. D. Pritchett, «BASE: An Acid Alternative,» т. 6, № 3, 2008.

20. University of Cape Town, «Distributed Database Systems,» [Онлайновий]. Available: https://www.cs.uct.ac.za/mit_notes/database/pdfs/chp15.pdf. [Дата звернення: 16 October 2019].

21. Oracle Server Distributed Systems Manual, «Understanding Distributed Systems,» [Онлайновий]. Available: https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SD173/ch1.htm. [Дата звернення: 5 жовтня 2019].

22. GeeksForGeeks, «Client-Server Model,» [Онлайновий]. Available: https://www.geeksforgeeks.org/client-server-model/. [Дата звернення: 27 листопада 2019].

23. R. Strickland, Cassandra High Availability, Packt, 2014.

24. A. Ibrar, «Multi-Master Replication Solutions for PostgreSQL,» 9 June 2020. [Онлайновий]. Available: https://www.percona.com/blog/2020/06/09/multi-master-replication-solutions-for-postgresql/. [Дата звернення: 17 січня 2021].

25. D. Tylor, «NoSQL Tutorial: What is, Types of NoSQL Databases & Example,» [Онлайновий]. Available: https://www.guru99.com/nosql-tutorial.html. [Дата звернення: 22 грудня 2019].

26. S. Samual, «Types of databases,» Tutorialspoint, 18 June 2020. [Онлайновий]. Available: https://www.tutorialspoint.com/Types-of-databases. [Дата звернення: 16 вересня 2020].

27. Fixstars, «NoSQL Database Architectural Comparison,» 29 June 2017. [Онлайновий]. Available: https://griddb.net/en/docs/NoSQL_Database_Architectural_Comparison.pdf. [Дата звернення: 14 грудня 2019].

28. Microsoft, «Azure Cosmos DB documentation,» [Онлайновий]. Available: https://learn.microsoft.com/en-us/azure/cosmos-db/. [Дата звернення: 29 грудня 2019].

29. A. Tanenbaum та M. Van Steen, «Distributed systems: Principles and Paradigms,» 2006.

30. J. Brutlag, «Speed Matters for Google Web Search,» Google, Inc., [Онлайновий]. Available: http://services.google.com/fh/files/blogs/google_delayexp.pdf. [Дата звернення: 16 березня 2022].

31. A. Gorbenko та R. A., «Time-outing Internet Services,» *IEEE Security & Privacy,* т. 11, № 2, pp. 68-71, 2013.

32. Ю. Горбенко, Побудування та аналіз систем, протоколів і засобів криптографічного захисту інформації, Харків: Видавництво "Форт", 2015.

33. R. Kalakuntla, A. B. Vanamala та R. R. Kolipyaka, «Cyber Security,» *HOLISTICA – Journal of Business and Public Administration,* т. 10, № 2, pp. 115-128, 2019.

34. Ç. Bakır та M. Güçlü, «Multi-level security model in distributed database systems,» *Pamukkale University Journal of Engineering Sciences,* т. 29, № 6, pp. 369-378, 2023.

35. S. Kumar, J. Seetha та S. Vinotha, «Security Implications of Distributed Database Management System Models,» *International Journal of Soft Computing And Software Engineering* , т. 2, № 11, pp. 20-28, 2012.

36. D. Arokia Sathis Kumar та N. M., «An Overview of Security in Distributed Database Management System,» *International Journal for Scientific Research & Development,* т. 3, № 10, pp. 232-236, 2015.

37. X. Wang, C. Xu, Z. Zhou, S. Yang та L. Sun, «A Survey of Blockchain-based Cybersecurity for Vehicular Networks,» в *International Wireless Communications and Mobile Computing (IWCMC)*, Limassol, 2020.

38. W. Naeem, M. A. Shah та A. K. Malik, «Privacy-Preserving in Collaborative Working Environments,» в *Proceedings of the IOARP International Conference on Communication and Networks,* London, 2015.

39. L. Bouganim та Y. Guo, Database Encryption, Boston: Springer, 2011.

40. A. Rafique, D. Van Landuyt, E. Heydari Beni, B. Lagaisse та W. Joosen, «CryptDICE: Distributed data protection system for secure cloud data storage and computation,» *Information Systems,* т. 96, 2021.

41. P. Singh та K. Kaur, «Database security using encryption,» в *015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, Greater Noida, 2015.

42. R. Byali, «Using MongoDB to Understand the Underlying Methods Techniques Encryption in NoSQL database,» *International jornal of research publication and reviews,* т. 3, № 9, pp. 862-866,, 2022.

43. L. Dandurand та O. S. Serrano, «Towards improved cyber security information sharing,» в *5th International Conference on Cyber Conflict (CYCON 2013)*, Tallinn, 2013.

44. M. Chapple, J. Stewart та D. Gibson, CISSP Certified Information Systems Security Professional Official Study Guide, Indianapolis: John Wiley & Sons, Inc., 2018.

45. V. Yesin, M. Karpinski, M. Yesina, V. Vilihura та K. Warwas, «Ensuring Data Integrity in Databases with the Universal Basis of Relations,» *Applied Sciences,* т. 11, № 18, 2021.

46. A. Amer Alwan, H. Ibrahim та N. Izura Udzir, «Local Integrity Checking using Local Information in a Distributed Database,» в *The 3rd Malaysian Software Engineering Conference*, Malaysia, 2007.

47. N. Azizah, V. Hartajaya та S. Riady, «Comparison Of Replication Strategies On Distributed Database Systems,» *International Journal of Cyber and IT Service Management,* т. 2, № 1, pp. 20-29, 2022.

48. A. Kumar Singh та U. Shanker, «DATABASE REPLICATION TECHNIQUES: A REVIEW,» *INTERNATIONAL JOURNAL OF RESEARCH*

REVIEW IN ENGINEERING SCIENCE & TECHNOLOGY, т. 5, № 4, pp. 16-19, 2006.

49. S. Abdul Moiz, P. Sailaja, G. Venkataswamy та N. P. Supriya, «Database Replication: A Survey of Open Source and Commercial Tools,» *International Journal of Computer Applications* , т. 13, № 6, 2011.

50. M. Wiesmann, F. Pedone, A. Schiper, B. Kemme та G. Alonso, «Understanding replication in databases and distributed systems,» в *Proceedings 20th IEEE International Conference on Distributed Computing Systems*, Taipei, 2000.

51. B. Ciciani, D. M. Dias та P. S. Yu, «Analysis of replication in distributed database systems,» *EEE Transactions on Knowledge and Data Engineering*, т. 2, № 2, pp. 247-261, 1990.

52. A. Gorbenko, A. Romanovsky, O. Tarasyuk та O. Biloborodov, «From Analyzing Operating System Vulnerabilities to Designing Multiversion Intrusion-Tolerant Architectures,» *IEEE Transactions on Reliability*, т. 69, № 1, pp. 22-39, 2020.

53. A. Gorbenko, A. Romanovsky та O. Tarasyuk, «Fault tolerant internet computing: Benchmarking and modelling trade-offs between availability, latency and consistency,» *Journal of Network and Computer Applications*, т. 146, pp. 1-27, 2019.

54. J. Domaschka, C. B. Hauser та B. Erb, «Reliability and Availability Properties of Distributed Database Systems,» в *IEEE 18th International Enterprise Distributed Object Computing Conference*, Ulm, 2014.

55. A. Gorbenko та A. Romanovsky, «Time-Outing Internet Services,» *IEEE Security and Privacy Magazine*, 2013.

56. A. Gorbenko та O. Tarasyuk, «Exploring timeout as a performance and availability factor of distributed replicated database systems,» *Radioelectronic and Computer Systems*, т. 4, № 96, pp. 98-105, 2020.

57. D. Dawson, «The Causes of Network Outages: Underlying Causes, Growing Threats and Industry Implications,» CircleID, 6 September 2023. [Онлайновий]. Available: https://circleid.com/posts/20230906-the-causes-of-network-outages. [Дата звернення: 24 вересня 2023].

58. В. Лямець та А. Тевашев, Системний аналіз, Харків: ХТУРЄ, 1998.

59. О. Пушкар, Інформатика, комп'ютерна техніка. Комп'ютерні технології., Київ: Академія, 2004.

60. С. Никаноров, Системний аналіз: етап розвитку методології рішення, Київ, 2001.

61. В. Маценко, Математичне моделювання: навчальний посібник, Чернівці: Чернівецький національний університет, 2014.

62. О. Станжицький, Є. Таран та Л. Гординський, Основи математичного моделювання: Навчальний посібник, Київ: Видавничополіграфічний центр "Київський університет", 2006.

63. І. Васильків, Основи теорії ймовірностей і математичної статистики, Львів: ЛНУ ім. Івана Франка , 2020.

64. А. Литвинов, Теорія систем масового обслуговування, Харків: ХНУМГ ім. О. М. Бекетова,, 2018.

65. K. Loren та P. Garg, «The threats to our products,» Microsoft Interface, 1 April 1999. [Онлайновий]. Available: http://blogs.msdn.com/sdl/attachment/9887486.ashx. [Дата звернення: 25 вересня 2021].

66. A. Shostack, «"STRIDE," in Threat Modeling: Designing for Security,» в *Threat Modeling: Designing for Security*, Wiley, 2014, pp. 61-86.

67. R. Khan, K. McLaughlin, D. Laverty та S. Sezer, «STRIDE-based threat modeling for cyber-physical systems,» в *IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, Turin, 2017.

68. M. Samadi, NoSQL: Database for Storage and Retrieval of Data in Cloud, 2017.

69. «Database Security Cheat Sheet,» OWASP, [Онлайновий]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Database_Security_Cheat_Sheet.html. [Дата звернення: 26 грудня 2020].

70. V. Pevnev та S. Kapchynskyi, «Database security: threats and preventive measures,» *Advanced Information Systems,* т. 2, № 1, p. 69–72, 2018.

71. «Database Security Guideline,» Database Security Consortium Security Guideline WG, 1 February 2009. [Онлайновий]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://www.db-security.org/report/dbsc_guideline_ver2.0_e.pdf. [Дата звернення: 15 січня 2021].

72. S. Preeti, «Database Security: Attacks and Techniques,» *International Journal of Scientific & Engineering Research,* т. 7, № 12, pp. 313-319, 2016.

73. A. M. Gamundani та L. M. Nekare, «A Review of New Trends in Cyber Attacks: A Zoom into Distributed Database Systems,» в *IST-Africa Week Conference (IST-Africa)*, Gaborone, 2018.

74. A. Zahid, R. Masood та M. A. Shibli, «Security of sharded NoSQL databases: A comparative analysis».

75. «Terraform, HCL,» HashiCorp, [Онлайновий]. Available: https://developer.hashicorp.com/terraform. [Дата звернення: 12 July 2021].

76. «Ansible,» RedHat, [Онлайновий]. Available: https://www.ansible.com/. [Дата звернення: 15 червня 2021].

77. GitHub, «YCSB,» 11 October 2019. [Онлайновий]. Available: https://github.com/brianfrankcooper/YCSB/wiki. [Дата звернення: 5 лютого 2020].

78. S. Gokhale, S. Noonan, N. Agrawal та C. Ungureanu, «KVZone and the search for a write-optimized key-value store,» в *Proceedings of the 2nd USENIX conference on Hot topics in storage and file systems,* 2010.

79. S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs та B. Rinaldi, «YCSB++: Benchmarking and performance debugging advanced features in scalable table stores,» в *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011.

80. A. Dey, A. Fekete, R. Nambiar та U. Röhm, «YCSB+T: Benchmarking web-scale transactional databases,» в *IEEE 30th International Conference on Data Engineering Workshops*, Chicago, 2014.

81. S. P. Kumar, S. Lefebvre, R. Chiky та E. G. Soudan, «Evaluating consistency on the fly using YCSB,» в *International Workshop on Computational Intelligence for Multimedia Understanding (IWCIM)*, Paris, 2014.

82. S. Taylor, «R-Squared,» CFI, 17 May 2020. [Онлайновий]. Available: https://corporatefinanceinstitute.com/resources/data-science/r-squared/. [Дата звернення: 14 грудня 2021].

83. W. Berg та G. Henningsson, «Mitigating HTTP Denial-of-Service Attacks on Self-Hosted Web Applications,» KHT Royal institute of technology, 5 June 2022. [Онлайновий]. Available: https://www.diva-portal.org/smash/get/diva2:1701965/FULLTEXT01.pdf. [Дата звернення: 15 грудня 2022].

84. А. Литвинов, Теорія систем масового обслуговування, Харків: нац. ун-т міськ. госп-ва ім. О. М. Бекетова, 2018.

85. А. Горбенко, «Оцінка виду та ступеня невизначеності нефункціональних характеристик web-додатків,» *Радіоелектронні і комп'ютерні системи,* № 4, pp. 30-34, 2013.

86. В. Харченко, О. Тарасюк, А. Горбенко та В. Скляр, «Метод та інструментальні засоби комплексної метрико-ймовірнісної оцінки надійності програмного забезпечення,» *Електронне моделювання,* т. 27, № 5, pp. 85-98, 2005.

87. А. Гультяев, MATLAB імітаційне моделювання, 1999.

88. А. Кобзарь, Прикладна математична статистика. Довідник для інженерів та нукових співробітників, 2006.

89. Н. Цейтлін, З досвіду аналітичної статистики, 2006.

90. «Kubernetes,» Kubernetes community, 3 January 2023. [Онлайновий]. Available: https://kubernetes.io/docs/concepts/overview/. [Дата звернення: 4 лютого 2023].

91. «Temporal developer's guide,» Temporal, [Онлайновий]. Available: https://docs.temporal.io/dev-guide. [Дата звернення: 14 січня 2023].

92. «Kubernetes Documentation,» Cloud Native Computing Foundation, 9 October 2022. [Онлайновий]. Available: https://kubernetes.io/docs/home/. [Дата звернення: 13 грудня 2022].

93. «Elastic Enterprise Search: Search across business systems and software,» Elasticsearch, [Онлайновий]. Available: https://www.elastic.co/getting-started/enterprise-search/search-across-business-systems-and-software. [Дата звернення: 13 березня 2023].

94. Amazon, «Amazon EC2,» Amazon Web Services, [Онлайновий]. Available: https://aws.amazon.com/ec2/. [Дата звернення: 3 червня 2022].

95. QCon, «NoSQL: past, present, future,» [Онлайновий]. Available: https://www.infoq.com/presentations/NoSQL-History/. [Дата звернення: 20 листопада 2019].

96. A. Presley та D. Liles, «The Use of IDEF0 for the Design and Specification of Methodologies,» в *Proceedings of the 4th industrial engineering research conference,* Fort Worth, 1995.

97. М. Фаулер, UML основи, Символ-Т, 2013.

98. E. Brewer, «Towards Robust Distributed Systems,» в *19th Annual ACM Symposium on Principles of Distributed Computing*, Portland, USA, 2000.

99. Q. Gu та P. Liu, «Denial of Service Attacks,» Pennsylvania State University, 7 June 2019. [Онлайновий]. Available: https://s2.ist.psu.edu/paper/ddos-chap-gu-june-07.pdf. [Дата звернення: 13 жовтня 2021].

100. D. B. Terry, K. Petersen, M. J. Spreitzer та M. M. Theimer, «Quorum Systems in Replicated Databases:Science or Fiction?,» в *The 16th International Conference onData Engineering*, San Diego, 2000.

ДОДАТОК А.
РЕЗУЛЬТАТИ ТЕСТУВАННЯ ГЛОБАЛЬНО-РОЗПОДІЛЕНОГО КЛАСТЕРУ

Таблиця А.1 – Часові затримки операції читання для патрену $\{[C], ([N_1]\}, \{[N_2],[N_3]\})$

К-сть потоків	Часова затримка, мс			Коефіцієнт варіації, %		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	6008	128132	166086	23	23	21
200	8872	174065	263505	26	15	21
300	12325	259843	376879	39	12	10
400	11709	336046	514455	45	8	10
500	12930	432965	635506	46	8	8
600	14324	510569	790233	36	68	79
700	18500	609466	893571	35	57	67
800	19146	689630	1043835	41	57	8
900	20992	791191	1131833	38	7	6
1000	25537	859495	1224197	34	5	4

Таблиця А.2 – Часові затримки операції запису для патрену $\{[C], ([N_1]\}, \{[N_2],[N_3]\})$

К-сть потоків	Часова затримка, мс			Коефіцієнт варіації, %		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	3986	123941	204113	20	21	19
200	5680	177012	271295	55	17	18
300	9147	269538	422127	28	13	15
400	10486	366567	560222	30	10	12
500	14816	451655	713362	45	8	13
600	15474	542011	829897	44	7	13
700	18456	627421	1004690	52	5	10
800	18856	722408	1140976	55	5	8
900	20512	816881	1281370	62	5	9
1000	31071	896612	1390675	62	5	8

Таблиця А.3 – Продуктивність патрену {[C], ([N_1]}, {[N_2],[N_3]})

К-сть потоків	Операція читання, опр/с			Операція запису, опр/с		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	7860	353	266	8683	349	213
200	8801	499	332	9909	472	300
300	10292	506	341	12620	475	293
400	11705	512	337	13112	472	303
500	12539	505	343	13804	473	305
600	12917	512	339	14428	474	305
700	13579	506	337	14551	476	308
800	13428	512	346	15630	475	303
900	14258	506	353	15781	478	312
1000	15160	517	355	15906	477	308

Таблиця А.4 – Часові затримки операції читання для патрену {[C]}, ({[N_1]},{[N_2]},{[N_3]})

К-сть потоків	Часова затримка, мс			Коефіцієнт варіації, %		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	126405	237817	299333	0.2	11	11
200	126502	246774	310453	0.7	9	13
300	126773	262582	374315	0.3	9	11
400	126738	314229	446182	2	13	8
500	126966	425842	578242	0.9	9	8
600	126869	500338	775985	0.9	7	9
700	127314	595304	902325	3	7	6
800	127030	698664	1007110	0.8	5	7
900	127376	754920	1159050	3	5	7
1000	128032	857244	1299007	3	5	6

Таблиця А.5 – Часові затримки операції запису для патрену {[C]}, ({[N_1]},{[N_2]},{[N_3]})

К-сть потоків	Часова затримка, мс			Коефіцієнт варіації, %		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	126653	245554	320857	1	13	11
200	126598	252074	322845	0.3	11	15
300	126907	270237	381095	1	9	13
400	126957	359274	544990	1	10	11
500	126948	447262	648055	3	8	11
600	126970	531854	728439	2	6	8
700	127072	614306	893938	1	6	8
800	127580	706083	1099533	1	6	9
900	127669	798297	1285334	2	5	9
1000	127673	875723	1492448	4	5	6

Таблиця А.6 – Продуктивність патрену {[C]}, ({[N_1]},{[N_2]},{[N_3]})

К-сть потоків	Операція читання, опр/с			Операція запису, опр/с		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	347	178	140	335	172	133
200	685	363	292	668	338	262
300	1041	468	344	1007	454	312
400	1378	498	342	1342	477	314
500	1729	511	345	1657	482	338
600	2085	508	374	2016	479	315
700	2384	512	346	2356	484	343
800	2752	515	376	2688	485	354
900	3104	517	386	3049	483	349
1000	3381	518	390	3313	491	357

Таблиця А.7 – Часові затримки операції читання для патрену {[C], ([N_1],[N_2],[N_3])}

К-сть потоків	Часова затримка, мс			Коефіцієнт варіації, %		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	5454	7001	8365	48	65	66
200	6519	8371	9828	76	79	64
300	8139	10762	11724	89	61	68
400	11670	13666	15738	91	65	54
500	13000	15702	19208	77	52	52
600	14571	16638	20529	81	57	57
700	16364	18400	21390	74	55	43
800	18079	19540	22906	78	45	37
900	19814	22194	25735	86	54	41
1000	23802	27593	32340	70	48	56

Таблиця А.8 – Часові затримки операції запису для патрену {[C], ([N_1],[N_2],[N_3])}

К-сть потоків	Часова затримка, мс			Коефіцієнт варіації, %		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	4398	4732	4980	61	52	65
200	6258	6907	7708	86	65	69
300	8458	9495	10101	86	52	62
400	9584	11254	12050	70	83	50
500	11461	13092	13906	86	64	49
600	12940	14886	15655	95	61	58
700	15764	16795	18239	74	71	63
800	17395	18637	20511	96	75	49
900	18955	20697	23400	71	50	55
1000	21053	23360	27383	93	55	59

Таблиця А.9 – Продуктивність патрену {[C], ([N_1],[N_2],[N_3])}

К-сть потоків	Операція читання, опр/с			Операція запису, опр/с		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	6360	5489	4128	6444	6140	5618
200	9357	8949	8155	9947	9310	7775
300	12001	10564	9820	11817	11016	10664
400	12759	11111	10738	13435	12955	11790
500	14296	12571	11186	14627	13977	12719
600	15365	13415	12146	16015	14980	13961
700	14564	14561	13018	16770	15680	14764
800	15920	15649	12675	16377	15861	14730
900	16830	14640	13735	18338	17384	15434
1000	14733	16074	12945	17873	16055	15016

Таблиця А.10 – Часові затримки операції читання для патрену {[C]}, ({[N_1],[N_2],[N_3]})

К-сть потоків	Часова затримка, мс			Коефіцієнт варіації, %		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	126675	127226	127960	0.8	2.6	0.7
200	126902	127624	128505	0.4	4.3	3.3
300	126811	127435	128220	1.5	2.2	3.4
400	127004	127843	128746	1.4	3.1	5.4
500	126999	127773	128353	2.3	1.1	2.2
600	126978	127814	128467	1.5	2.8	2.2
700	126963	127655	128832	1.4	1	3.4
800	127075	128226	129022	1.3	4	2.4
900	126781	127760	128601	1	1.3	1.3
1000	127261	127963	129187	1	1	6

Таблиця А.11 – Часові затримки операції запису для патрену {[C]}, ({[N_1],[N_2],[N_3]})

К-сть потоків	Часова затримка, мс			Коефіцієнт варіації, %		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	127010	127698	128339	1.5	0.4	1.5
200	126825	127670	128435	0.5	1.1	6.1
300	127003	127428	128090	1.2	1.4	1.8
400	126784	127635	128251	1.9	1	1.2
500	127141	127769	128565	1.5	1.5	3.8
600	127116	127524	128075	1.4	1.6	3.2
700	127287	128024	128714	2.5	1.9	3.5
800	126931	127719	128537	3.3	2.9	5.1
900	127535	128054	129115	2.6	2.4	4.3
1000	127467	128252	129027	3	2	4

Таблиця А.12 – Пропускна здатність патрену {[C]}, ({[N_1],[N_2],[N_3]})

К-сть потоків	Операція читання, опр/с			Операція запису, опр/с		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
100	345	344	342	335	335	335
200	692	685	680	674	671	663
300	1036	1035	1028	1010	1010	1000
400	1382	1366	1364	1352	1343	1341
500	1730	1724	1722	1687	1680	1663
600	2066	2056	2054	2014	2014	1998
700	2424	2404	2388	2357	2343	2336
800	2770	2733	2717	2715	2684	2658
900	3115	3100	3076	3022	3012	2980
1000	3450	3424	3320	3357	3340	3319

ДОДАТОК Б.

ЛІСТИНГ ПРОГРАМНОГО КОДУ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

Б.1. Скрипт розгортання глобально-розподіленого кластеру у хмарному провайдеру

```
 #Create Amazon VPC with specific CIDR block
 resource "aws_vpc" "vpc" {
   cidr_block           = var.vpc_cidr
   enable_dns_hostnames = true

   tags = merge(
     local.common_tags, {
       "Name" = "${var.project_name}-vpc"
   })
 }

 #Create Amazon DHCP
 resource "aws_vpc_dhcp_options" "dhcp" {
   domain_name_servers = ["AmazonProvidedDNS"]

   tags = merge(
     local.common_tags, {
       "Name" = "${var.project_name}-dhcp"
   })
 }

 #Associate Amazon DHCP and Amazon VPC
 resource "aws_vpc_dhcp_options_association" "vpc_dhcp" {
   vpc_id          = aws_vpc.vpc.id
   dhcp_options_id = aws_vpc_dhcp_options.dhcp.id
 }

 #Create Amazon IGW
 resource "aws_internet_gateway" "igw" {
   vpc_id = aws_vpc.vpc.id

   tags = merge(
     local.common_tags, {
       "Name" = "${var.project_name}-igw"
   })
 }

 #Create `main` Amazon Route Table
 resource "aws_route_table" "main" {
```

```
  vpc_id = aws_vpc.vpc.id

  tags = merge(
    local.common_tags, {
      "Name" = "${var.project_name}-rt"
  })
}

#Create default route table rule
resource "aws_route" "default" {
  route_table_id         = aws_route_table.main.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.igw.id
}

#Create Amazon EC2 key pair
resource "aws_key_pair" "ec2_keypair" {
  key_name   = "${var.project_name}-pub-key"
  public_key = var.ec2_keypair_public_key
}


#Create public subnet
resource "aws_subnet" "pb_sn_az" {
  count = length(var.aws_az)

  vpc_id            = aws_vpc.vpc.id
  cidr_block        = cidrsubnet(aws_vpc.vpc.cidr_block, 12, count.index)
  availability_zone = "${var.aws_region}${var.aws_az[count.index]}"

  map_public_ip_on_launch = true

  tags = merge(
    local.common_tags, {
      "Name" = "${var.project_name}-pb-sn-${var.aws_az[count.index]}"
  })
}

#Associate public subnet with main route table
resource "aws_route_table_association" "pb_sn_az" {
  count = length(var.aws_az)

  subnet_id      = aws_subnet.pb_sn_az[count.index].id
  route_table_id = aws_route_table.main.id
}
```

```
#Create public subnet
resource "aws_subnet" "pb_sn_cm_az" {
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = cidrsubnet(aws_vpc.vpc.cidr_block, 12, 12)
  availability_zone = "${var.aws_region}a"

  map_public_ip_on_launch = true

  tags = merge(
    local.common_tags, {
      "Name" = "${var.project_name}-pb-sn-cm-a"
  })
}

#Associate public subnet with main route table
resource "aws_route_table_association" "pb_sn_cm_az" {
  subnet_id      = aws_subnet.pb_sn_cm_az.id
  route_table_id = aws_route_table.main.id
}

    variable "project_name" {
  default = "karpenko-test"
}

variable "aws_region" {
  default = "us-east-1"
}

variable "aws_az" {
  default = ["a", "b", "c", "a", "b", "c", "a","b","c","a","b"]
}
#263215456332
variable "vpc_cidr" {
  default = "10.168.0.0/16"
}
variable "vm" {
  default = [
    {
      ami_id        = "ami-071fe47787cd40412"
      instance_type = "t2.xlarge"
    },
    {
      ami_id        = "ami-071fe47787cd40412"
```

```
      instance_type = "t2.xlarge"
    },
    {
      ami_id        = "ami-071fe47787cd40412"
      instance_type = "t2.xlarge"
    },
    # {
    #   ami_id        = "ami-071fe47787cd40412"
    #   instance_type = "t2.xlarge"
    # },
    # {
    #   ami_id        = "ami-071fe47787cd40412"
    #   instance_type = "t2.xlarge"
    # },
    # {
    #   ami_id        = "ami-071fe47787cd40412"
    #   instance_type = "t2.xlarge"
    # },
    # {
    #   ami_id        = "ami-071fe47787cd40412"
    #   instance_type = "t2.xlarge"
    # },
    # {
    #   ami_id        = "ami-071fe47787cd40412"
    #   instance_type = "t2.xlarge"
    # },
    # {
    #   ami_id        = "ami-071fe47787cd40412"
    #   instance_type = "t2.xlarge"
    # },
    # {
    #   ami_id        = "ami-071fe47787cd40412"
    #   instance_type = "t2.xlarge"
    # },
    # {
    #   ami_id        = "ami-071fe47787cd40412"
    #   instance_type = "t2.xlarge"
    # }
  ]
}

variable "cm" {
  default = {
    ami_id        = "ami-071fe47787cd40412"
    instance_type = "t2.xlarge"
```

```
  }
}

variable "ec2_keypair_public_key" {
  default = "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCQ+mEXKAMUMSsdTGGdEubzB
IpMkzErn51XqC1vTKZqVJojae2mwvPCD5baHYqZ7jV5fWjM4OirHhrhvVTH4miv
PzM+G2ibjHUmx6eGw2/n12OrMJyXxtVup67Jo9lxlXGDnHFytUTEi+deCHYghtftj
s5CZaiKlVf2G7p8ja1fe0FjWRmIIe7MmYjwK4Zr9goH42HBNc1JsroV1/qcFdvOh6e
DFq64xJdqHpja2fDIa7nhGglybC35G5Yn3EQ71CWVGF0zTXZ2JxfvgJ4pLQ5C9ap
kbA4rem3M+/xsfrVKbLAd2LAr7b62fUlPEB2uGsLJPFKWpwRJXYEI0WDfL5gf"
}

locals {
  common_tags = {
    "Owner"   = "andrii.karpenko@yum.com"
    "Project" = "Training"
    "Purpose" = "Training"
    "Removal" = "24/12/2022"
  }
}

locals {
  nodes = [
    for node in aws_instance.vm.* :
    {
      instance   = index(aws_instance.vm.*, node)
      public_ip  = node.public_ip
      private_ip = node.private_ip
    }
  ]

  # inventory = {
  #     all = {
  #         children = [
  #             for node in aws_instance.vm.*:
  #             {
  #                 index(aws_instance.vm.*, node) = {
  #                     hosts = {
  #                         node.public_ip
  #                     }
  #                 }
  #             }
  #         ]
  #     }
```

```
  # }

}
resource "aws_security_group" "vm" {
  name        = "${var.project_name}-vm-sg"
  description = "Controls in/out traffic for vm networking"
  vpc_id      = aws_vpc.vpc.id

  tags = merge(
    local.common_tags, {
      "Name" = "${var.project_name}-vm-sg"
  })
}

resource "aws_security_group_rule" "inbound_internal_vm" {
  security_group_id = aws_security_group.vm.id
  description       = "Allow all traffic within vm"
  type              = "ingress"
  from_port         = "0"
  to_port           = "65535"
  protocol          = "all"
  self              = "true"
}

resource "aws_security_group_rule" "inbound_ssh_to_vm" {
  security_group_id = aws_security_group.vm.id
  description       = "Allow ssh protocol for any"
  type              = "ingress"
  from_port         = "22"
  to_port           = "22"
  protocol          = "tcp"
  cidr_blocks       = ["0.0.0.0/0"]
}


resource "aws_security_group_rule" "vm_outbound_any" {
  security_group_id = aws_security_group.vm.id
  type              = "egress"
  from_port         = 0
  to_port           = 0
  protocol          = "all"
  cidr_blocks       = ["0.0.0.0/0"]
}
```

```
resource "aws_network_interface" "vm" {
  count = length(var.vm)

  subnet_id       = aws_subnet.pb_sn_az[count.index].id
  private_ips     = [cidrhost(cidrsubnet(aws_vpc.vpc.cidr_block, 12, count.index), 4)]
  security_groups = [aws_security_group.vm.id]

  tags = merge(
    local.common_tags, {
      "Name" = "${var.project_name}-vm-${var.aws_az[count.index]}-eni"
  })
}

resource "aws_instance" "vm" {
  count = length(var.vm)

  ami               = var.vm[count.index].ami_id
  instance_type     = var.vm[count.index].instance_type
  availability_zone = "${var.aws_region}${var.aws_az[count.index]}"
  key_name          = aws_key_pair.ec2_keypair.key_name
  user_data = templatefile("./files/ec2/nix-userdata.tpl", {
    hostname = "${var.project_name}-vm-${var.aws_az[count.index]}-
${count.index}"
  })
  disable_api_termination = false

  network_interface {
    device_index         = 0
    network_interface_id = aws_network_interface.vm[count.index].id
  }

  root_block_device {
      # volume_type          = "io1"
      volume_size          = "100"
      delete_on_termination = true
  }

  lifecycle {
    #[TEMP] Due to user-data forces new resource when you move terraform apply
from windows to linux and vice versa
    ignore_changes = [user_data]
  }

  tags = merge(
    local.common_tags, {
```

```
    "Name" = "${var.project_name}-vm-${var.aws_az[count.index]}-
${count.index}"
  })

  # volume_tags = merge(
  #     local.common_tags, {
  #       "Name"   = "${var.project_name}-vm-${var.aws_az[count.index]}-ebs"
  # })
}

resource "aws_network_interface" "cm" {
  subnet_id       = aws_subnet.pb_sn_cm_az.id
  private_ips     = [cidrhost(cidrsubnet(aws_vpc.vpc.cidr_block, 12, 12), 4)]
  security_groups = [aws_security_group.vm.id]

  tags = merge(
    local.common_tags, {
      "Name" = "${var.project_name}-cm-a-eni"
  })
}

resource "aws_instance" "cm" {
  ami               = var.cm.ami_id
  instance_type     = var.cm.instance_type
  availability_zone = "${var.aws_region}a"
  key_name          = aws_key_pair.ec2_keypair.key_name
  user_data = templatefile("./files/ec2/nix-userdata.tpl", {
    hostname = "cassandra-client"
  })
  disable_api_termination = false

  network_interface {
    device_index         = 0
    network_interface_id = aws_network_interface.cm.id
  }

  # root_block_device {
  #     volume_type           = var.cm.root_block_device.volume_type
  #     volume_size           = var.cm.root_block_device.volume_size
  #     delete_on_termination = var.cm.root_block_device.is_delete_on_termination
  #     encrypted             = var.cm.root_block_device.is_encrypted
  # }

  lifecycle {
```

```
    #[TEMP] Due to user-data forces new resource when you move terraform apply
from windows to linux and vice versa
    ignore_changes = [user_data]
  }

  tags = merge(
    local.common_tags, {
      "Name" = "client-a"
  })

  # volume_tags = merge(
  #     local.common_tags, {
  #         "Name"   = "${var.project_name}-vm-${var.aws_az[count.index]}-ebs"
  # })
}

resource "local_file" "host" {
  content = templatefile("files/templates/hosts", {
    node_1 = local.nodes[0].public_ip
    node_2 = local.nodes[1].public_ip
    node_3 = local.nodes[2].public_ip
    # node_4 = local.nodes[3].public_ip
    # node_5 = local.nodes[4].public_ip
    # node_6 = local.nodes[5].public_ip
    # node_7 = local.nodes[6].public_ip
    # node_8 = local.nodes[7].public_ip
    # node_9 = local.nodes[8].public_ip
    # node_10 = local.nodes[9].public_ip
    # node_11 = local.nodes[10].public_ip
    cm     = aws_instance.cm.public_ip
  })
  filename = "../os/inventory/hosts"
}

resource "local_file" "node_1" {
  content = templatefile("files/templates/node.yaml", {
    cluster_name   = "Amazon cassandra cluster"
    seeds          = join(",", local.nodes.*.private_ip)
    listen_address = local.nodes[0].private_ip
    private_ip     = local.nodes[0].private_ip
  })
  filename = "../os/inventory/group_vars/node-1/conf.yaml"
}

resource "local_file" "node_2" {
```

```
  content = templatefile("files/templates/node.yaml", {
    cluster_name   = "Amazon cassandra cluster"
    seeds          = join(",", local.nodes.*.private_ip)
    listen_address = local.nodes[1].private_ip
    private_ip     = local.nodes[1].private_ip
  })
  filename = "../os/inventory/group_vars/node-2/conf.yaml"
 }

 resource "local_file" "node_3" {
  content = templatefile("files/templates/node.yaml", {
    cluster_name   = "Amazon cassandra cluster"
    seeds          = join(",", local.nodes.*.private_ip)
    listen_address = local.nodes[2].private_ip
    private_ip     = local.nodes[2].private_ip
  })
  filename = "../os/inventory/group_vars/node-3/conf.yaml"
 }
```

Б.2. Скрипт налаштування глобально-розподіленого кластеру

```
 ---
 - name: Stop cassandra service
   systemd:
     name: cassandra
     state: stopped

 - name: Remove old cassandra data
   file:
     path: /var/lib/cassandra/data/system
     state: absent

 - name: Set cassandra configuration
   template:
     src: files/cassandra.yaml
     dest: /etc/cassandra/cassandra.yaml
     owner: root
     group: root
     mode: '0644'

 - name: Remove cassandra-topology.properties
   file:
     path: /etc/cassandra/cassandra-topology.properties
     state: absent
```

```
- name: Start cassandra service
  systemd:
    name: cassandra
    state: started

- name: Install DataDog
  shell: DD_AGENT_MAJOR_VERSION=7
DD_API_KEY=9155fbe97419eca72d5662a092b7e75c DD_SITE="datadoghq.eu"
bash -c "$(curl -L https://s3.amazonaws.com/dd-agent/scripts/install_script.sh)"
...
- name: Download YCSB repository
  unarchive:
    src: https://github.com/brianfrankcooper/YCSB/releases/download/0.17.0/ycsb-
0.17.0.tar.gz
    dest: /home/ubuntu/
    remote_src: yes

- name: Delete test table
  shell: cqlsh 10.168.0.4 -e "DROP TABLE IF EXISTS ycsb.usertable"

- name: Delete keyspace (if exist)
  shell: cqlsh 10.168.0.4 -e "DROP KEYSPACE IF EXISTS ycsb"

- name: Create keyspace
  shell: |
    cqlsh 10.168.0.4 -e "CREATE KEYSPACE ycsb WITH REPLICATION = {'class'
: 'NetworkTopologyStrategy', 'eu-west-2': {{ replication_factor }} };"
    cqlsh 10.168.0.4 -e "CREATE TABLE ycsb.usertable (
            y_id varchar primary key,
            field0 varchar,
            field1 varchar,
            field2 varchar,
            field3 varchar,
            field4 varchar,
            field5 varchar,
            field6 varchar,
            field7 varchar,
            field8 varchar,
            field9 varchar);"

############################ READ TEST ############################
- name: Set cassandra configuration
  template:
    src: files/workloadc
```

```
    dest: /home/ubuntu/ycsb-0.17.0/workloads/workloadc
    owner: ubuntu
    group: ubuntu
    mode: '0644'
    force: yes
  vars:
    consistency_level: ONE
    operationcount: 100000
    is_write_test: 0
    is_read_test: 1

- name: Load data to DataBase
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb load cassandra-cql -P workloads/workloadc -p hosts=10.168.0.4

- name: Consistency level ONE
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb.sh run cassandra-cql -s -threads 1000 -P workloads/workloadc  -p
hosts=10.168.0.4 -p cassandra.readconsistencylevel=ONE -p measurementtype=raw -
p measurement.raw.output_file=/home/ubuntu/1000_ONE_read.csv

- name: Set cassandra configuration
  template:
    src: files/workloadc
    dest: /home/ubuntu/ycsb-0.17.0/workloads/workloadc
    owner: ubuntu
    group: ubuntu
    mode: '0644'
    force: yes
  vars:
    consistency_level: TWO
    operationcount: 100000
    is_write_test: 0
    is_read_test: 1

- name: Load data to DataBase
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb load cassandra-cql -P workloads/workloadc -p hosts=10.168.0.4

- name: Consistency level TWO
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
```

```
    ./bin/ycsb.sh run cassandra-cql -s -threads 1000 -P workloads/workloadc  -p
hosts=10.168.0.4 -p cassandra.readconsistencylevel=TWO -p measurementtype=raw -
p measurement.raw.output_file=/home/ubuntu/1000_TWO_read.csv


- name: Set cassandra configuration
  template:
    src: files/workloadc
    dest: /home/ubuntu/ycsb-0.17.0/workloads/workloadc
    owner: ubuntu
    group: ubuntu
    mode: '0644'
    force: yes
  vars:
    consistency_level: THREE
    operationcount: 100000
    is_write_test: 0
    is_read_test: 1


- name: Load data to DataBase
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb load cassandra-cql -P workloads/workloadc -p hosts=10.168.0.4


- name: Consistency level THREE
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb.sh run cassandra-cql -s -threads 1000 -P workloads/workloadc  -p
hosts=10.168.0.4 -p cassandra.readconsistencylevel=THREE -p
measurementtype=raw -p
measurement.raw.output_file=/home/ubuntu/1000_THREE_read.csv


- name: Set cassandra configuration
  template:
    src: files/workloadc
    dest: /home/ubuntu/ycsb-0.17.0/workloads/workloadc
    owner: ubuntu
    group: ubuntu
    mode: '0644'
    force: yes
  vars:
    consistency_level: ALL
    operationcount: 100000
```

```
    is_write_test: 0
    is_read_test: 1

- name: Load data to DataBase
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb load cassandra-cql -P workloads/workloadc -p hosts=10.168.0.4

- name: Consistency level ALL
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb.sh run cassandra-cql -s -threads 1000 -P workloads/workloadc -p
hosts=10.168.0.4 -p cassandra.readconsistencylevel=ALL -p measurementtype=raw -
p measurement.raw.output_file=/home/ubuntu/1000_ALL_read.csv

- name: Set cassandra configuration
  template:
    src: files/workloadc
    dest: /home/ubuntu/ycsb-0.17.0/workloads/workloadc
    owner: ubuntu
    group: ubuntu
    mode: '0644'
    force: yes
  vars:
    consistency_level: QUORUM
    operationcount: 100000
    is_write_test: 0
    is_read_test: 1

- name: Load data to DataBase
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb load cassandra-cql -P workloads/workloadc -p hosts=10.168.0.4

- name: Consistency level QUORUM
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb.sh run cassandra-cql -s -threads 1000 -P workloads/workloadc -p
hosts=10.168.0.4 -p cassandra.readconsistencylevel=QUORUM -p
measurementtype=raw -p
measurement.raw.output_file=/home/ubuntu/1000_QUORUM_read.csv

############################ WRITE TEST ############################
- name: Set cassandra configuration
  template:
```

```
    src: files/workloadc
    dest: /home/ubuntu/ycsb-0.17.0/workloads/workloadc
    owner: ubuntu
    group: ubuntu
    mode: '0644'
    force: yes
  vars:
    consistency_level: ONE
    operationcount: 100000
    is_write_test: 1
    is_read_test: 0

- name: Load data to DataBase
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb load cassandra-cql -P workloads/workloadc -p hosts=10.168.0.4

- name: Consistency level ONE
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb.sh run cassandra-cql -s -threads 1000 -P workloads/workloadc -p
hosts=10.168.0.4 -p cassandra.writeconsistencylevel=ONE -p measurementtype=raw
-p measurement.raw.output_file=/home/ubuntu/1000_ONE_write.csv

- name: Set cassandra configuration
  template:
    src: files/workloadc
    dest: /home/ubuntu/ycsb-0.17.0/workloads/workloadc
    owner: ubuntu
    group: ubuntu
    mode: '0644'
    force: yes
  vars:
    consistency_level: TWO
    operationcount: 100000
    is_write_test: 1
    is_read_test: 0


- name: Load data to DataBase
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb load cassandra-cql -P workloads/workloadc -p hosts=10.168.0.4
```

```
- name: Consistency level TWO
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb.sh run cassandra-cql -s -threads 1000 -P workloads/workloadc -p
hosts=10.168.0.4 -p cassandra.writeconsistencylevel=TWO -p measurementtype=raw
-p measurement.raw.output_file=/home/ubuntu/1000_TWO_write.csv


- name: Set cassandra configuration
  template:
    src: files/workloadc
    dest: /home/ubuntu/ycsb-0.17.0/workloads/workloadc
    owner: ubuntu
    group: ubuntu
    mode: '0644'
    force: yes
  vars:
    consistency_level: THREE
    operationcount: 100000
    is_write_test: 1
    is_read_test: 0


- name: Load data to DataBase
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb load cassandra-cql -P workloads/workloadc -p hosts=10.168.0.4


- name: Consistency level THREE
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb.sh run cassandra-cql -s -threads 1000 -P workloads/workloadc -p
hosts=10.168.0.4 -p cassandra.writeconsistencylevel=THREE -p
measurementtype=raw -p
measurement.raw.output_file=/home/ubuntu/1000_THREE_write.csv


- name: Set cassandra configuration
  template:
    src: files/workloadc
    dest: /home/ubuntu/ycsb-0.17.0/workloads/workloadc
    owner: ubuntu
    group: ubuntu
    mode: '0644'
    force: yes
```

```
  vars:
    consistency_level: ALL
    operationcount: 100000
    is_write_test: 1
    is_read_test: 0


- name: Load data to DataBase
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb load cassandra-cql -P workloads/workloadc -p hosts=10.168.0.4


- name: Consistency level ALL
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb.sh run cassandra-cql -s -threads 1000 -P workloads/workloadc -p
hosts=10.168.0.4 -p cassandra.writeconsistencylevel=ALL -p measurementtype=raw -
p measurement.raw.output_file=/home/ubuntu/1000_ALL_write.csv


- name: Set cassandra configuration
  template:
    src: files/workloadc
    dest: /home/ubuntu/ycsb-0.17.0/workloads/workloadc
    owner: ubuntu
    group: ubuntu
    mode: '0644'
    force: yes
  vars:
    consistency_level: QUORUM
    operationcount: 100000
    is_write_test: 1
    is_read_test: 0


- name: Load data to DataBase
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
    ./bin/ycsb load cassandra-cql -P workloads/workloadc -p hosts=10.168.0.4


- name: Consistency level QUORUM
  shell: |
    cd /home/ubuntu/ycsb-0.17.0
```

```
    ./bin/ycsb.sh run cassandra-cql -s -threads 1000 -P workloads/workloadc -p
hosts=10.168.0.4 -p cassandra.writeconsistencylevel=QUORUM -p
measurementtype=raw -p
measurement.raw.output_file=/home/ubuntu/1000_QUORUM_write.csv

############################INIT CLUSTER############################
- hosts: node-1
  gather_facts: false
  any_errors_fatal: true
  become: yes
  tasks:
    - include: tasks/init-cluster.yaml

- hosts: node-2
  gather_facts: false
  any_errors_fatal: true
  become: yes
  tasks:
    - include: tasks/init-cluster.yaml

- hosts: node-3
  gather_facts: false
  any_errors_fatal: true
  become: yes
  tasks:
    - include: tasks/init-cluster.yaml

- hosts: node-4
  gather_facts: false
  any_errors_fatal: true
  become: yes
  tasks:
    - include: tasks/init-cluster.yaml

- hosts: node-5
  gather_facts: false
  any_errors_fatal: true
  become: yes
  tasks:
    - include: tasks/init-cluster.yaml

- hosts: node-6
  gather_facts: false
  any_errors_fatal: true
  become: yes
```

```
  tasks:
    - include: tasks/init-cluster.yaml

- hosts: node-7
  gather_facts: false
  any_errors_fatal: true
  become: yes
  tasks:
    - include: tasks/init-cluster.yaml

############################TEST DB############################
- hosts: cm
  gather_facts: false
  any_errors_fatal: true
  tasks:
    - include: tasks/test-db.yaml
      # loop: "{{ replication_factor }}"
      # loop_control:
      #   loop_var: replication
```

Б.3. Скрипт аналізу та корекції налаштувань глобально-розподіленого кластеру

```
import csv

import numpy as np

from prettytable import PrettyTable
from scipy.stats import variation
import math


def calc_params(filename, threads, operation):
    "This function calculate parameters"
    tmp = []
    latency_values = []
    avr_latency = 0
    length = 0
    delta = 0
    time = 0
    items = 0
    coefficient = 0.3
    with open(filename, 'r') as data:
```

```
        content = csv.DictReader(data)
        tmp = list(content)
        if operation == 'read':
            length = len(tmp) - (threads + 2)
            delta = int((length * coefficient) / 2)  # /2
            for i in range(delta, (length - delta)):
                avr_latency += int(tmp[i][' latency(us)'])
                latency_values.append(int(tmp[i][' latency(us)']))
                items = items + 1
            time = float(tmp[(length)][' timestamp(ms)']
                         ) - float(tmp[0][' timestamp(ms)'])
            # print(length)
            # print(tmp[0][' timestamp(ms)'])
            # print(tmp[(length)][' timestamp(ms)'])
        else:
            length = len(tmp) - (threads + 2)
            delta = int((length * coefficient) / 2)
            for i in range((delta + (threads + 2)),  (length - delta)):
                avr_latency += int(tmp[i][' latency(us)'])
                latency_values.append(int(tmp[i][' latency(us)']))
                items = items + 1
            time = float(tmp[(len(tmp) - delta)][' timestamp(ms)']
                         ) - float(tmp[delta][' timestamp(ms)'])
        return items, avr_latency/items, variation(latency_values), items/(time * 0.001),
latency_values


filePath = "/Users/akarpenko/Desktop/client32"
resultFilePath = "/Users/akarpenko/Desktop/results/theSameDatacenter"
read_latency = PrettyTable()
read_latency.field_names = ["Threads", "ONE Latency", "ONE Variation",
                            "ONE Ops/s", "QUORUM Latency", "QUORUM Variation",
"QUORUM Ops/s",
                            "ALL Latency", "ALL Variation", "ALL Ops/s"]
# read_latency.field_names = ["Threads",
#                             "ONE ops/s"]

############### READ###############
for i in [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]:
    one_count, one_latency, one_variation, one_ops, one_values = calc_params(
        "%s/%s_%s_%s.csv" % (filePath, i, "ONE", "read"), i, "read")
    quorum_count, quorum_latency, quorum_variation, quorum_ops, quorum_values =
calc_params(
        "%s/%s_%s_%s.csv" % (filePath, i, "QUORUM", "read"), i, "read")
    all_count, all_latency, all_variation, all_ops, all_values = calc_params(
```

```
      "%s/%s_%s_%s.csv" % (filePath, i, "ALL", "read"), i, "read")
   read_latency.add_row([i, one_latency,
                  one_variation, one_ops, quorum_latency,
                  quorum_variation, quorum_ops,all_latency,
                  all_variation, all_ops])
   # read_latency.add_row([i, one_ops])

   np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "ONE", "read"), one_values,
delimiter=',')
   np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "QUORUM", "read"),
quorum_values, delimiter=',')
   np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "ALL", "read"), all_values,
delimiter=',')
   # read_latency.add_row([i, one_latency, one_variation,one_ops])

print(read_latency)

write_latency = PrettyTable()
write_latency.field_names = ["Threads", "ONE Latency", "ONE Variation",
                  "ONE Ops/s", "QUORUM Latency", "QUORUM Variation",
"QUORUM Ops/s",
                  "ALL Latency", "ALL Variation", "ALL Ops/s"]

# ###############WRITE###############
for i in [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]:
   one_count, one_latency, one_variation, one_ops, one_values = calc_params(
      "%s/%s_%s_%s.csv" % (filePath, i, "ONE", "write"), i, "write")
   quorum_count, quorum_latency, quorum_variation, quorum_ops, quorum_values =
calc_params(
      "%s/%s_%s_%s.csv" % (filePath, i, "QUORUM", "write"), i, "write")
   all_count, all_latency, all_variation, all_ops, all_values = calc_params(
      "%s/%s_%s_%s.csv" % (filePath, i, "ALL", "write"), i, "write")
   write_latency.add_row([i, one_latency,
                  one_variation, one_ops, quorum_latency,
                  quorum_variation, quorum_ops,all_latency,
                  all_variation, all_ops])
   # write_latency.add_row([i, one_ops])
   np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "ONE", "write"), one_values,
delimiter=',')
   np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "QUORUM", "write"),
quorum_values, delimiter=',')
   np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "ALL", "write"), all_values,
delimiter=',')
```

```
print(write_latency)
```

Б.4. Скрипт аналізу та корекції налаштувань глобально-розподіленого кластеру

```
import csv

import numpy as np

from prettytable import PrettyTable
from scipy.stats import variation
import math


def calc_params(filename, threads, operation):
    "This function calculate parameters"
    tmp = []
    latency_values = []
    avr_latency = 0
    length = 0
    delta = 0
    time = 0
    items = 0
    coefficient = 0.3
    with open(filename, 'r') as data:
        content = csv.DictReader(data)
        tmp = list(content)
        if operation == 'read':
            length = len(tmp) - (threads + 2)
            delta = int((length * coefficient) / 2)  # /2
            for i in range(delta, (length - delta)):
                avr_latency += int(tmp[i][' latency(us)'])
                latency_values.append(int(tmp[i][' latency(us)']))
                items = items + 1
            time = float(tmp[(length)][' timestamp(ms)']
                         ) - float(tmp[0][' timestamp(ms)'])
            # print(length)
            # print(tmp[0][' timestamp(ms)'])
            # print(tmp[(length)][' timestamp(ms)'])
        else:
            length = len(tmp) - (threads + 2)
            delta = int((length * coefficient) / 2)
            for i in range((delta + (threads + 2)),  (length - delta)):
                avr_latency += int(tmp[i][' latency(us)'])
                latency_values.append(int(tmp[i][' latency(us)']))
```

```
            items = items + 1
        time = float(tmp[(len(tmp) - delta)][' timestamp(ms)']
                ) - float(tmp[delta][' timestamp(ms)'])
    return items, avr_latency/items, variation(latency_values), items/(time * 0.001),
latency_values


filePath = "/Users/akarpenko/Desktop/client32"
resultFilePath = "/Users/akarpenko/Desktop/results/theSameDatacenter"
read_latency = PrettyTable()
read_latency.field_names = ["Threads", "ONE Latency", "ONE Variation",
                    "ONE Ops/s", "QUORUM Latency", "QUORUM Variation",
"QUORUM Ops/s",
                    "ALL Latency", "ALL Variation", "ALL Ops/s"]
# read_latency.field_names = ["Threads",
#                      "ONE ops/s"]

############### READ###############
for i in [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]:
   one_count, one_latency, one_variation, one_ops, one_values = calc_params(
      "%s/%s_%s_%s.csv" % (filePath, i, "ONE", "read"), i, "read")
   quorum_count, quorum_latency, quorum_variation, quorum_ops, quorum_values =
calc_params(
      "%s/%s_%s_%s.csv" % (filePath, i, "QUORUM", "read"), i, "read")
   all_count, all_latency, all_variation, all_ops, all_values = calc_params(
      "%s/%s_%s_%s.csv" % (filePath, i, "ALL", "read"), i, "read")
   read_latency.add_row([i, one_latency,
                  one_variation, one_ops, quorum_latency,
                  quorum_variation, quorum_ops,all_latency,
                  all_variation, all_ops])
   # read_latency.add_row([i, one_ops])

   np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "ONE", "read"), one_values,
delimiter=',')
   np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "QUORUM", "read"),
quorum_values, delimiter=',')
   np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "ALL", "read"), all_values,
delimiter=',')
   # read_latency.add_row([i, one_latency, one_variation,one_ops])

print(read_latency)

write_latency = PrettyTable()
write_latency.field_names = ["Threads", "ONE Latency", "ONE Variation",
```

```
                    "ONE Ops/s", "QUORUM Latency", "QUORUM Variation",
"QUORUM Ops/s",
                    "ALL Latency", "ALL Variation", "ALL Ops/s"]

# ###############WRITE###############
for i in [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]:
    one_count, one_latency, one_variation, one_ops, one_values = calc_params(
        "%s/%s_%s_%s.csv" % (filePath, i, "ONE", "write"), i, "write")
    quorum_count, quorum_latency, quorum_variation, quorum_ops, quorum_values =
calc_params(
        "%s/%s_%s_%s.csv" % (filePath, i, "QUORUM", "write"), i, "write")
    all_count, all_latency, all_variation, all_ops, all_values = calc_params(
        "%s/%s_%s_%s.csv" % (filePath, i, "ALL", "write"), i, "write")
    write_latency.add_row([i, one_latency,
                    one_variation, one_ops, quorum_latency,
                    quorum_variation, quorum_ops,all_latency,
                    all_variation, all_ops])
    # write_latency.add_row([i, one_ops])
    np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "ONE", "write"), one_values,
delimiter=',')
    np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "QUORUM", "write"),
quorum_values, delimiter=',')
    np.savetxt("%s/%s_%s_%s.csv" % (resultFilePath, i, "ALL", "write"), all_values,
delimiter=',')


print(write_latency)
```

Б.5. Скрипт гібридної імітаційної моделі

```
from numpy.random import default_rng
import numpy as np
import random
import matplotlib.pyplot as plt


def model_behavior(replicasCount=3, distributionName="gamma",
distributionShape=10000, distributionScale=6, consistencyLevel="one",
speculativeRetry="false", speculativeRetryCalls=0, minDelay=0):
    delays = []
    resultDelay = 0
    rng = default_rng()
```

```
    if distributionName == "gamma":
        delays = rng.gamma(distributionShape, distributionScale, replicasCount)
    elif distributionName == "weibull":
        delays = rng.weibull(distributionShape, replicasCount)
    else:
        print(f"Entered distribution [{distributionName}] does not support")
        raise ValueError(
            "The distribution should be one of [gamma or weibull]")

    if consistencyLevel == "one" or consistencyLevel == "One":
        if speculativeRetry == "false" and speculativeRetryCalls == 1:
            # 1 - 1 - 3: get random value from array of delays
            resultDelay = delays[random.randint(0, replicasCount - 1)]
            print(f"1-1-3:{resultDelay}")
        elif speculativeRetry == "true" and (speculativeRetryCalls > 1 and
speculativeRetryCalls < replicasCount):
            # delays = delays.sort() # 1 - 2 - 3 and 1 - 3 - 3: get min delay from array of
delays
            tmp = []
            j = 0
            for i in range(0, int(replicasCount/2)+1):
                j = random.randint(0, replicasCount - 1)  # random index
                while 1:
                    if delays[j] in tmp:
                        j = random.randint(
                            0, replicasCount - 1)  # random index
                    else:
                        break
                tmp.append(delays[j])
            sortedDelay = sorted(tmp)

            resultDelay = sortedDelay[0]
            # print("Result:")
            # print(resultDelay)
            print(f"1-2-3:{resultDelay}")
        elif speculativeRetry == "true" and (speculativeRetryCalls > 1 and
speculativeRetryCalls == replicasCount):
            sortedDelay = sorted(delays)
            resultDelay = sortedDelay[0]
            print("Delays:")
            print(delays)
            print("Sorted:")
            print(sortedDelay)
            print(f"1-2-3:{resultDelay}")
        else:
```

```
            print(f"1-2(3)-3:error")
            resultDelay = "error"  # Not valid params
        elif consistencyLevel == "quorum" or consistencyLevel == "Quorum":
            if speculativeRetry == "false" and (int(replicasCount/2)+1) ==
speculativeRetryCalls:
                tmp = []
                j = 0
                for i in range(0, int(replicasCount/2)+1):
                    j = random.randint(0, replicasCount - 1)  # random index
                    while 1:
                        if delays[j] in tmp:
                            j = random.randint(
                                0, replicasCount - 1)  # random index
                        else:
                            break
                    tmp.append(delays[j])
                # 2 - 2 - 3:
                sortedDelay = sorted(tmp)
                resultDelay = sortedDelay[(int(replicasCount/2)+1)-1]
                print("Delays:")
                print(delays)
                print("Sorted:")
                print(sortedDelay)
                print(f"2-2-3:{resultDelay}")
            elif speculativeRetry == "true" and (speculativeRetryCalls >
(int(replicasCount/2)+1) and speculativeRetryCalls <= replicasCount):
                # 2 - 3 - 3: get min delay from array of delays
                sortedDelay = sorted(delays)
                resultDelay = sortedDelay[(int(replicasCount/2)+1)-1]
                print(f"2-3-3:{resultDelay}")
            else:
                print(f"2-2(3)-3:error")
                resultDelay = "error"  # Not valid params
        else:
            # 3 - 3 - 3: get max value form array of delays
            sortedDelay = sorted(delays)
            resultDelay = sortedDelay[replicasCount - 1]
            print(f"3-3-3:{resultDelay}")

        return resultDelay + minDelay, delays


shape_100 = 1.97
scale_100 = 1014.40
minDelay_100 = 619
```

```
scale_200 = 2.41
shape_200 = 1516.77
minDelay_200 = 1759

scale_300 = 10.33
shape_300 = 219.38
minDelay_300 = 3751

scale_400 = 14
shape_400 = 734

scale_600 = 17
shape_600 = 865

scale_800 = 20
shape_800 = 946

scale_900 = 24
shape_900 = 1079

scale_1200 = 27
shape_1200 = 1185


##### 100, 200, 300
# 1-1-3 100 req/s shape = 2.554678, scale = 1126.388333
temp_1_1_3_100 = []
res_delays = []
for i in range(0, 1000):
    res, delays = model_behavior(3, "gamma", shape_100,
                                 scale_100, "one", "false", 1,minDelay_100)
    res_delays.append(delays)
    temp_1_1_3_100.append(res)

np.savetxt("1_1_3_100.csv", temp_1_1_3_100, delimiter=',')
np.savetxt("1_1_3_100_delays.csv", res_delays, delimiter=',')

temp_1_1_3_200 = []
res_delays = []
for i in range(0, 1000):
    res, delays = model_behavior(3, "gamma", shape_200,
                                 scale_200, "one", "false", 1,minDelay_200)
    res_delays.append(delays)
    temp_1_1_3_200.append(res)
```

```
np.savetxt("1_1_3_200.csv", temp_1_1_3_200, delimiter=',')
np.savetxt("1_1_3_200_delays.csv", res_delays, delimiter=',')


temp_1_1_3_300 = []
res_delays = []
for i in range(0, 1000):
    res, delays = model_behavior(3, "gamma", shape_300,
                                 scale_300, "one", "false", 1,minDelay_300)
    res_delays.append(delays)
    temp_1_1_3_300.append(res)

np.savetxt("1_1_3_300.csv", temp_1_1_3_300, delimiter=',')
np.savetxt("1_1_3_300_delays.csv", res_delays, delimiter=',')

# 1-2-3 200 req/s shape = 2.255489, scale = 2847.010385
temp_1_2_3_200 = []
res_delays = []
for i in range(0, 1000):
    res, delays = model_behavior(3, "gamma", shape_200,
                                 scale_200, "one", "true", 2,minDelay_200)
    res_delays.append(delays)
    temp_1_2_3_200.append(res)

np.savetxt("1_2_3_200.csv", temp_1_2_3_200, delimiter=',')
np.savetxt("1_2_3_200_delays.csv", res_delays, delimiter=',')

# 1-3-3 300 req/s shape =  0.713627, scale 12075.153294
temp_1_3_3_300 = []
res_delays = []
for i in range(0, 1000):
    res, delays = model_behavior(3, "gamma", shape_300,
                                 scale_300, "one", "true", 3, minDelay_300)
    res_delays.append(delays)
    temp_1_3_3_300.append(res)

np.savetxt("1_3_3_300.csv", temp_1_3_3_300, delimiter=',')
np.savetxt("1_3_3_300_delays.csv", res_delays, delimiter=',')

# 2-2-3 200 req/s shape = 2.255489, scale = 2847.010385
temp_2_2_3_200 = []
res_delays = []
for i in range(0, 1000):
    res, delays = model_behavior(3, "gamma", shape_200,
```

```
                                scale_200, "quorum", "false", 2,minDelay_200)
    res_delays.append(delays)
    temp_2_2_3_200.append(res)

np.savetxt("2_2_3_200.csv", temp_2_2_3_200, delimiter=',')
np.savetxt("2_2_3_200_delays.csv", res_delays, delimiter=',')


# # 2-3-3 300 req/s shape =  0.713627, scale = 12075.153294
temp_2_3_3_300 = []
res_delays = []
for i in range(0, 1000):
    res, delays = model_behavior(3, "gamma", shape_300,
                                scale_300, "quorum", "true", 3,minDelay_300)
    res_delays.append(delays)
    temp_2_3_3_300.append(res)

np.savetxt("2_3_3_300.csv", temp_2_3_3_300, delimiter=',')
np.savetxt("2_3_3_300_delays.csv", res_delays, delimiter=',')

# # 3-3-3 300 req/s shape =  0.713627, scale = 12075.153294
temp_3_3_3_300 = []
res_delays = []
for i in range(0, 1000):
    res, delays = model_behavior(3, "gamma", shape_300,
                                scale_300, "all", "false", 3,minDelay_300)
    res_delays.append(delays)
    temp_3_3_3_300.append(res)

np.savetxt("3_3_3_300.csv", temp_3_3_3_300, delimiter=',')
np.savetxt("3_3_3_300_delays.csv", res_delays, delimiter=',')

plt.hist([temp_1_1_3_100, temp_1_1_3_200, temp_1_1_3_300], alpha=0.5,
bins=np.arange(
    1000, 8000, 250), edgecolor='black', label=["1_1_3_100", "1_1_3_200",
"1_1_3_300"])


plt.hist([temp_1_1_3_100, temp_1_2_3_200, temp_1_3_3_300, temp_2_2_3_200,
temp_2_3_3_300, temp_3_3_3_300], alpha=0.5, bins=np.arange(
    1000, 50000, 1000), edgecolor='black', label=["1_1_3_100", "1_2_3_200",
"1_3_3_300", "2_2_3_200", "2_3_3_300", "3_3_3_300"])

plt.hist([temp_1_1_3_100, temp_1_2_3_200, temp_1_3_3_300, temp_2_2_3_200,
temp_2_3_3_300, temp_3_3_3_300], alpha=0.5, bins=np.arange(
```

```
    1000, 8000, 100), edgecolor='black', label=["1_1_3_100", "1_2_3_200",
"1_3_3_300", "2_2_3_200", "2_3_3_300", "3_3_3_300"])

plt.hist([temp_1_1_3_300, temp_1_2_3_600, temp_1_3_3_900, temp_2_2_3_600,
temp_2_3_3_900, temp_3_3_3_900], alpha=0.5, bins=np.arange(
    1000, 8000, 100), edgecolor='black', label=["1_1_3_300", "1_2_3_600",
"1_3_3_900", "2_2_3_600", "2_3_3_900", "3_3_3_900"])

plt.hist(temp_1_1_3_100,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_1_3_100")
plt.hist(temp_1_2_3_200,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_2_3_200")
plt.hist(temp_1_3_3_300,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_3_3_300")
plt.hist(temp_2_2_3_200,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="2_2_3_200")
plt.hist(temp_2_3_3_300,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="2_3_3_300")
plt.hist(temp_3_3_3_300,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="3_3_3_300")

plt.hist(temp_1_1_3_200,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_1_3_200")
plt.hist(temp_1_2_3_400,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_2_3_400")
plt.hist(temp_1_3_3_600,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_3_3_600")
plt.hist(temp_2_2_3_400,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="2_2_3_400")
plt.hist(temp_2_3_3_600,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="2_3_3_600")
plt.hist(temp_3_3_3_600,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="3_3_3_600")

plt.hist(temp_1_1_3_300,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_1_3_300")
plt.hist(temp_1_2_3_600,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_2_3_600")
plt.hist(temp_1_3_3_900,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_3_3_900")
plt.hist(temp_2_2_3_600,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="2_2_3_600")
plt.hist(temp_2_3_3_900,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="2_3_3_900")
```

```
plt.hist(temp_3_3_3_900,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="3_3_3_900")

plt.hist(temp_1_1_3_400,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_1_3_400")
plt.hist(temp_1_2_3_800,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_2_3_800")
plt.hist(temp_1_3_3_1200,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="1_3_3_1200")
plt.hist(temp_2_2_3_800,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="2_2_3_800")
plt.hist(temp_2_3_3_1200,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="2_3_3_1200")
plt.hist(temp_3_3_3_1200,alpha=0.5,bins=np.arange(1000, 8000,
100),edgecolor='black',label="3_3_3_1200")
plt.xlabel('Count')
plt.ylabel('Delay')
plt.legend(loc='upper right')
plt.show()
```

ДОДАТОК В.

АКТИ ВПРОВАДЖЕНЬ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОЇ РОБОТИ

АКТ ВПРОВАДЖЕННЯ
наукових результатів дисертаційної роботи за темою
«Моделі, методи та інформаційна технологія оцінки та підвищення готовності і оперативності глобально-розподілених систем зберігання даних на основі контролюючої узгодженості»
Карпенка Андрія Сергійовича,
виконаної на здобуття наукового ступеня доктора філософії

Комісія у складі голови – Бігдан А.О., *Senior Cloud DevOps Engineer*, членів – Шакір'янова К.С., *Intermediate Software Engineer*, встановила, що результати наукових досліджень аспіранта кафедри 503 Національно аерокосмічного університету ім. М. Є. Жуковського «Харківський авіаційний інститут» Карпенка А.С., а саме:

1) метод оптимального вибору рівня узгодженості даних глобально-розподілених реплікованих систем зберігання великих обсягів інформації;

2) метод випереджаючих читань глобально-розподілених реплікованих систем зберігання великих обсягів інформації, що дозволяє зменшити екстремальні часові затримки при обмежені на зростання середнього часу обслуговування запитів або мінімізувати зростання середнього часу обслуговування при обмеженні на максимальний час обслуговування;

3) комплекс моделей для глобально-розподілених реплікованих систем зберігання даних, які вирішують задачі формалізації патернів розгортання у хмарному середовищі за допомогою апарату теоретико-множинного опису, деталізацію загроз для кібербезпеки та оцінки часу обслуговування при використанні запропонованого механізму випереджаючих читань за допомогою гібридного імітаційного підходу,

були реалізовані при проектуванні та впроваджені системи підтримки виконання асинхронних завдань, що дозволило підвищити готовність з 99,2% до 99,63% та оперативність, зменшивши максимальну затримку з 85689.2 мс до 53896.27 мс.

Голова комісії Бігдан А.О.
Члени комісії Шакір'янова К.С.

Директор ТОВ "СофтСерв Інновації" Копанський Т.Б.

28 серпня 2023 р.

ЗАТВЕРДЖУЮ
Проректор з наукової роботи
Національного аерокосмічного
університету ім. Н.Є. Жуковського
«Харківський авіаційний інститут»

д.т.н., професор
_____ В.В. Павліков
«16» 03 2023 р.

АКТ ВПРОВАДЖЕННЯ

наукових результатів дисертаційної роботи Карпенка Андрія Сергійовича, виконаної на здобуття наукового ступеня доктора філософії при виконанні держбюджетних проектів

Комісія у складі голови – декана факультету радіоелектроніки, комп'ютерних систем та інфокомунікацій, к.т.н. Одокієнка О.В., членів – доцента кафедри комп'ютерних систем, мереж і кібербезпеки, к.т.н., доцента, Фесенка Г.В., професора кафедри, д.т.н., професора, Пєвнева В.Я., доцента кафедри, к.т.н., доцента, Орєхова О.О., встановила, що результати наукових досліджень Карпенка Андрія Сергійовича, а саме:

1) метод оптимального вибору рівня узгодженості даних глобально-розподілених реплікованих систем зберігання великих обсягів інформації;

2) метод випереджаючих читань глобально-розподілених реплікованих систем зберігання великих обсягів інформації;

3) комплекс моделей для глобально-розподілених реплікованих систем зберігання даних,

було впроваджено у науково-дослідницкій роботі кафедри комп'ютерних систем, мереж і кібербезпеки у вигляді теоретичних положень, використаних при виконанні НДР: «Методологічні засади та технології оцінювання та забезпечення безпеки (захисту) критичних інформаційних інфраструктур» (№ ДР 0119U100979, 2019-2021) та «Методи, моделі та інформаційні технології підвищення надійності та безпечності складних ІТ-систем на етапах розроблення та впровадження» (№ Д/Р 0121U113842, 2021-2023).

Це дозволило підвищити якість виконання НДР щодо розроблення, оцінки та впровадження глобально-розподілених реплікованих систем зберігання даних

Голова комісії О. В. Одокієнко

Члени комісії Г. В. Фесенко
В. Я. Пєвнев
О. О. Орєхов

16 03 2023 р.

LEEDS BECKETT UNIVERSITY

Headingley Campus
Leeds
United Kingdom
LS6 3QS

Tel: 0113 812 0000
Web: www.leedsbeckett.ac.uk

To: whom it may concern

20 July 2022

Act of Implementation
of the results of PhD doctoral research
by Andrii Karpenko

This is to confirm that results of PhD doctoral research by Andrii Karpenko, namely:

(i) method of optimal selection of the data consistency level of globally distributed replicated storage systems aimed at reducing system latency and increasing its availability while preserving the strong data consistency;

(ii) method of speculative reading aimed at reducing extreme delays and improving availability of distributed databases via exploiting redundancy of data replicas

have been adopted in the teaching process of Leeds Beckett University ('Cloud Computing Development' module) and been used to define topics of MSc studies within the GrEen NetworkIng And cLoud computing (GENIAL) project of Erasmus Mundus Joint Master Degrees program (610619-EPP-1-2019-1-FR-EPPKA1-JMD-MOB).

Yours Sincerely

Professor Tawfik Hissam

School of Built Environment, Engineering and Computing
Leeds Beckett University
Headingley Campus
Leeds LS6 3QS, UK
Tel: 0044(0)113 8127591
Email: T.Hissam@leedsbeckett.ac.uk