НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.051

«До захисту допущено»
В.о. завідувача кафедри
_____ Едуард ЖАРІКОВ
«___»_____2021 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Метод та інструментальні засоби досягнення консенсусу в розподілених системах побудованих на базі платформи .Net »

Виконав (-ла):
студент (-ка) ІІ курсу, групи ІП-02мп
Шелудько Дмитро Максимович _____

Керівник:
Старший викладач
Ковтунець Олесь Володимирович _____

Рецензент:
доцент кафедри інформаційних
систем та технологій ФІОТ,
к.т.н., доцент
Жданова Олена Григорівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Едуард ЖАРІКОВ

«___»_____2021р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Шелудьку Дмитру Максимовичу

1. Тема дисертації «Метод та інструментальні засоби досягнення консенсусу в розподілених системах побудованих на базі платформи .Net», науковий керівник дисертації Ковтунець Олесь Володимирович, старший викладач, затверджені наказом по університету від «27» жовтня 2021 р. № 3587-с
2. Термін подання студентом дисертації «6» грудня 2021 р.
3. Об’єкт дослідження – процес досягнення консенсусу в розподілених системах.
4. Предмет дослідження – методи та алгоритми досягнення консенсусу в розподілених системах.
5. Перелік завдань, які потрібно розробити – аналіз проблеми та існуючих рішень; розробка програмного забезпечення; дослідження ефективності розробленого програмного забезпечення.
6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 5 плакатів
7. Орієнтовний перелік публікацій: Порівняння фреймворків для вирішення проблеми маштабування розподілених систем з використанням моделі акторів.

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9.Дата видачі завдання «30» вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Вивчення та аналіз предметної області	14.09	
2	Пошук та аналіз існуючих методів розв’язання задачі	20.09	
3	Постановка та формалізація математичної моделі задачі	02.10	
4	Модифікація існуючих методів розв’язання задачі	15.10	
5	Розробка програмного забезпечення	01.11	
6	Виконання експериментальних досліджень	13.11	
7	Оформлення пояснювальної записки	17.11	
8	Подання дисертації на попередній захист	22.11	
9	Подання дисертації на захист	6.12	

Студент Дмитро ШЕЛУДЬКО

Науковий керівник Олесь КОВТУНЕЦЬ

РЕФЕРАТ

Магістерська дисертація: 81 с., 10 рис., 5 табл., 6 додатків, 8 джерел.

Актуальність. Використання розподілених систем в якості альтернативи типовим клієнт-серверним рішеням постійно набирає популярності. Навантаження на сучасні інформаційні системи постійно зростає, тому побудова та використання розподілених рішень, інколи є єдиним виходом для досягнення поставлених вимог.

На сьогоднішній день можливість горизонтального маштабування стало вимогою до кожного модулю системи. Основна проблема, що виникає при маштабувані системи – це синхронізація її роботи. Наразі є декілька інструментів та алгоритмів для вирішення даної проблеми, але всі вони пагано інтегруються в системи, що побудовані на базі платформи .Net. Також на даний момент немає бібліотеки або модулю для платформи .Net, що можна інтегрувати в систему та використовувати для вирішення проблем неухгодженості. Також, одною із актуальних проблем – є складність імплементації існуючих алгоритмів на базі платформи .Net.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська робота виконувалась згідно з планом досліджень кафедри інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського»

Мета дослідження – розробка методу та алгоритму, ціллю якого буде вирішення проблеми консенсусу. Порівняння розробленого методу із існуючими та реалізація алгоритму у вигляді бібліотеки для платформи .Net.

Для досягнення мети необхідно виконати наступні завдання:

а) провести аналіз існуючих рішень, методів та алгоритмів для вирішення проблеми консенсусу;

б) створити та описати метод та алгоритм, побудований на базі розробленого методу досягнення консенсусу;

в) порівняти розроблений метод досягнення консенсусу з існуючими рішеннями;

г) створити бібліотеку на базі платформи .Net, що реалізує розроблений алгоритм;

д) розробити розподілену системи для валідації та аналізу роботи бібліотеки;

е) провести аналіз та валідацію коректності роботи бібліотеки.

Об'єкт дослідження – процес досягнення консенсусу між сервісами в розподілених системах.

Предмет дослідження – методи та алгоритм досягнення консенсусу між сервісами в розподілених системах.

Методи дослідження – основним методом дослідження, який використовується в даній роботі є моделювання.

Наукова новизна одержаних результатів полягає в розробці методу та алгоритму досягнення консенсусу та порівнянні його з існуючими аналогами з точки зору оптимальності, простоти імплементації та валідності роботи.

Публікації. Матеріали роботи опубліковані у журналі «Innovative solutions in modern science» (випуск №2/29, а також одна стаття подана до публікації), на Міжнародній науково-практичній конференції «Математичне та імітаційне моделювання систем. МОДС 2018» (25-29 червня 2018р.) та на II Всеукраїнській науково-практичній конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2019).

РОЗПОДІЛЕНІ СИСТЕМИ, PAXOS, .NET, МІКРОСЕРВІСНА АРХІТЕКТУРА, КОНСЕНСУС

ABSTRACT

Master's dissertation: 81 pp., 10 figs., 5 tables, appendix, 8 sources.

Topicality. The use of distributed systems as an alternative to typical client-server solutions is constantly gaining popularity. The load on modern information systems is constantly growing, so the construction and use of distributed solutions is sometimes the only way to achieve the requirements.

To date, the possibility of horizontal scaling has become a requirement for each module of the system. The main problem that arises when scaling the system is the synchronization of its operation. There are currently several tools and algorithms to solve this problem, but they are all poorly integrated into systems built on the .Net platform. There is also currently no library or module for the .Net platform that can be integrated into the system and used to resolve inconsistencies. Also, one of the current problems is the difficulty of implementing existing algorithms based on the .Net platform.

Connection of work with scientific programs, plans, themes. The master's thesis was performed according to the research plan of the Department of Informatics and Software Engineering of the National Technical University of Ukraine "Kyiv Polytechnic Institute. Igor Sikorsky »

The purpose of the study is to develop a method and algorithm, the purpose of which will be to solve the problem of consensus. Comparison of the developed method with the existing ones and implementation of the algorithm in the form of a library for the .Net platform.

To achieve this goal you must perform the following tasks:

a) analyze existing solutions, methods and algorithms to solve the problem of consensus;
b) create and describe a method and algorithm based on the developed method of reaching consensus;
c) compare the developed method of reaching consensus with existing solutions;

The object of research is the process of reaching consensus between services in distributed systems.

The subject of research - methods and algorithms for reaching consensus between services in distributed systems.

Research methods - the main research method used in this work is modeling.

The scientific novelty of the obtained results lies in the development of a method and algorithm for reaching consensus and comparing it with existing analogues in terms of optimality, ease of implementation and validity of work.

Publications. Materials of the work were published in the journal "Innovative solutions in modern science" (issue №2 / 29, as well as one article submitted for publication), at the International scientific-practical conference "Mathematical and simulation modeling of systems. MODS 2018 ”(June 25-29, 2018) and at the II All-Ukrainian scientific-practical conference of young scientists and students“ Information systems and management technologies ”(ISTU-2019).

Research methods - the main research method used in this work is modeling.

Relationship of work with scientific programs, plans, themes. The work was carried out at the Department of Computer-Aided Management and Data Processing Systems of the National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute» within the framework of the topic «Creation of simulation tools for discrete-event systems».

The purpose of the research - to identify possible ways of abuse by the community of free-ride users in decentralized networks with reward distribution [10] and compare them in terms of the amount of reward received by the community as well as the possibilities of preventing.

To achieve this purpose, it is needed to accomplish the following **tasks:**

a) carry out an overview of modern research in the area of abuse in decentralized networks;
b) formulate hypotheses about possible behavior of the community of free-ride users;
c) perform modelling of reward distribution using the proposed patterns of free-ride users community behavior;

d) analyze results of the experiments.

The object of research – the process of reward distribution in decentralized networks with 2 types of users.

Subject of research – users strategies of abusing in decentralized networks with reward distribution.

The research methods – main research method, that is used in this paper is modelling.

The scientific novelty of received results consists in identifying the most effective strategies for abuse by the community of free-ride users in decentralized networks with reward distribution [10], in terms of the amount of reward received by the community, and to compare these strategies in terms of the severity of the prevention by the main network.

Publications. Materials of the research were published in the journal “Innovative solutions in modern science” (vol. 2 / 29, and one article submitted for publication), at the International scientific and practical conference “Mathematical and simulation modeling of systems. MODS 2018” (June 25-29, 2018) and at the II All-Ukrainian Scientific and Practical Conference of Young Scientists and Students “Information Systems and Technologies of Management” (ISTM-2019).

DISTRIBUTED SYSTEMS, PAXOS, .NET, MICROSERVICE ARCHITECTURE, CONSENSUS

ЗМІСТ

ВСТУП

Існує чимало сфер діяльності, в яких при автоматизації їх бізнес-процесів за умов використання мікросервісної архітектури на перший план виходить необхідність гарантувати високий рівень консистентності та злагодженості роботи, загальний порядок операцій на всіх етапах виконання, коректну поведінку при часткових збоях системи чи затримках мережі для обміну данними, можливість автономного вирішення помилок та самовідновленя після збоїв. Більшість сучасних систем потребують високого рівня доступності та мають високі нефункціональні вимоги до продукту та окремих модулів. Чудовим прикладом може слугувати банківська сфера з її потребою в високому рівні злагодженості роботи сервісів, необхідністю підтримувати максимальний рівень валідності данних та чітко відслідковувати послідовність операцій з рахунком клієнта. Сюди ж можна віднести будь-які інші облікові операції в бухгалтерії великих компаній та корпорацій, в системах для проведення та організації аукціонів, брокерах та біржах, тендерах тощо.

В розподілених системах випадок, при якому існує потреба в досягненні консенсусу роботи системи, є найскладнішим. Будь-яке рішення, яке дає можливість синхронізації компонентів, негативно впливає на продуктивність та пропускну здатність системи. Більше того, для забезпечення збереження порядку операцій та гарантій повної консистентності, важливо також забезпечувати відмовостійкість системи за рахунок реплікації даних в кластері. Вузли кластера за умови відмови вузла-лідера повинні продовжити роботу без перебоїв та втрат вже зафіксованих лідером значень. Таким чином вибір найбільш оптимального методу та алгоритму досягнення консенсусу і його модифікація для вирішення практичних проблем системи при роботі з розподіленими системами та мікросервісною архітектурою є актуальним питанням і потребує дослідження.

1 ОГЛЯД ДОСЛІДЖЕНЬ І ТЕОРЕТИЧНИХ МАТЕРІАЛІВ В ОБЛАСТІ ДОСЯГНЕННЯ КОНСЕНСУСУ В РОЗПОДІЛЕНИХ СИСТЕМАХ

1.1 Огляд розподілених систем

Спочатку було розглянуто модель розподілених систем, та архітектуру мікросервісних додатків і її особливості. Основними джерелами, які було розглянуто в процесі вивчення даних систем були [1] і [3]. Розподілені системи є альтернативою централізованій або клієнт-серверній моделі будування програмного забезпечення, якій притаманна особливість горизонтального маштабування.

Сучасним розподіленим системам притаманний фактор можливості відмови одного чи більше вузлів мережі. Цей фактор накладає дуже великі вимоги під час побудови децентралізованих систем високого рівня доступності та відмовостійкості. Спочатку розглянемо розподілену систему обробки інформації, що є кластером із x86-серверів. Якщо для системи, побудованої на базі одного сервера, ймовірність відмови досить мала, і найчастіше при впровадженні простих систем можна знехтувати даною проблемою, то для кластера серверів ймовірність відмови одного із серверів стає в рази більшою. Для розподілених систем параметр середньої тривалості роботи між відмовами (MTBF, mean time between failures) є дуже важливим на всіх стадіях та життєвих циклах системи, оскільки характеризує надійність вузла системи. Середня тривалість роботи пристрою між відмовами показує, який рівень функціонування в середньому припадає на одну відмову:

$$T = \frac{\sum m t_i}{m}$$

де t_i – напрацювання до настання відмови i; m – кількість відмов.

Важливим фактором також є ненадійність мережі у вигляді відмови мережевого обладнання та втрати пакетів, відмови жорстких дисків, збої серверного програмного забезпечення на рівні ОС та на рівні додатків. Візьмемо для прикладу систему роботи Google – для кластера з 1800 машин існує висока ймовірність виявлення 1000

непрацездатних серверів протягом першого року експлуатації кластера, тобто близько 3 відмов щодня – і це ще не взято до уваги відмови жорстких дисків, проблеми з мережею і живленням тощо. В результаті, якщо не закладати високий рівень стійкості обладнання, доступності та відмовостійкості програмного забезпечення розподіленої системи, отримаємо систему, в якій кожна з зазначених вище проблем призводить до втрати роботоспроможності системи.

1.2 Огляд проблем досягнення консенсусу в розподілених системах

Дуже важливим питанням в галузі мікросервісів є формулювання правил і політик, за якими працює децентралізована система. Необхідно гарантувати всім користувачам єдине бачення стану системи в кожен конкретний момент часу. Саме цю проблему і вирішують консенсус-протоколи. По суті, консенсус-система підтримує журнал з фактами (реєстр фактів), який реплікується (копіюється) на декілька учасників системи, об'єднаних в мережу рівноправних вузлів.

Члени мережі – анонімні особи, звані вузлами. Коли учасник системи хоче додати факт в журнал, в мережі формується консенсус, щоб визначити, де цей факт повинен з'явитися в журналі. Надійне забезпечення доступності інформації щодо стану системи досягається за допомогою децентралізованих протоколів консенсусу. Децентралізовані протоколи досягнення консенсусу мають дуже широкий спектр застосування:

а) формування транзакційного журналу цифрових валют;
б) роботи розподілених баз даних;
в) програми керування авіаційним обладнанням;
г) критичні секції чи сервіси технічних систем;
д) обчислювальні системи високої надійності;
е) космічні системи, тощо.

Пропозиції (факти) в децентралізованій мережі одночасно формуються безліччю вузлів. Такі факти, що задовольняють поставленим критеріям, відправляються в розподілену мережу та додаються в розподілений лог фактів.

Виникають ситуації, коли кілька нових фактів в різних частинах розподіленої мережі протирічать один одному, тому потрібно вирішити, який із запропонованих фактів додати в розподілений лог. На допомогу приходять консенсус-протоколи, що дають таку можливість.

Загалом сценарії нештатного функціонування компонентів розподілених систем можна розділити на два класи:

а) повна відмова компонента. Характеризується цей клас проблем тим, що даний тип відмови призводить до недоступності одного з компонентів розподіленої системи (або сегментації мережі у разі відмови комутатора). До цього класу проблем належить відмова сервера, відмова системи зберігання даних, відмова комутатора, відмова операційної системи, відмова програми;

б) візантійська помилка. Характеризується тим, що вузол системи продовжує функціонувати, але може повертати некоректну інформацію. Наприклад, при використанні оперативної пам'яті без ECC можна отримати зчитування некоректних даних з пам'яті, помилки мережного обладнання можуть призводити до пошкодження пакетів тощо.

Помилки другого класу складніші для виявлення та виправлення. Надійна комп'ютерна система повинна бути в змозі впоратися з виходом із ладу одного або кількох її компонентів. Несправний компонент може проявляти особливий тип поведінки у вигляді надсилання суперечливої інформації в різні частини системи. Проблема подолання цього виду несправностей викладена абстрактно як проблема візантійських генералів. Проблема описана у вигляді історії, як кілька підрозділів візантійської армії розташувалися табором з різних сторін міста-суперника, з керуванням кожним підрозділом окремим генералом. Генерали можуть спілкуватися один з одним тільки за допомогою обміну повідомленнями, оскільки вони знаходяться з різних боків міста. Після спостереження за ворогом вони повинні прийняти загальний план дій. Проте деякі з генералів можуть бути зрадниками, які намагаються перешкодити вірним генералам досягти згоди. У генералів повинен бути алгоритм, який гарантує:

а) усі вірні генерали приймають один і той же план дій. Вірні генерали будуть робити те, що накаже алгоритм, але зрадники можуть робити все, що заманеться. Алгоритм повинен гарантувати виконання умови 1 незалежно від того, що роблять зрадники. Вірні генерали повинні не тільки досягти згоди, а й домовитися про розумний план;

б) невелика кількість зрадників не може змусити вірних генералів прийняти неправильне рішення.

Умову 2 важко формалізувати, оскільки вона вимагає точного визначення, що поганий план є поганим. Кожен генерал спостерігає за ворогом і передає свої спостереження іншим. Нехай v(i) — інформація, яку повідомляє і-ий загальний. Кожен генерал використовує певний метод для поєднання значень v (1) ..... v (n) в єдиний план дій, де n — кількість генералів. Умова 1 виконується завдяки тому, що всі генерали використовують один і той же метод для об'єднання інформації, а умова 2 досягається за допомогою надійного методу. Наприклад, якщо єдине рішення, яке необхідно прийняти – це атакувати чи відступати, тоді можна задіяти вибір за більшістю голосів серед усіх генералів. Невелика кількість зрадників може вплинути на рішення лише в тому випадку, якщо вірні генерали будуть майже порівну розділені між всіма можливими варіантами.

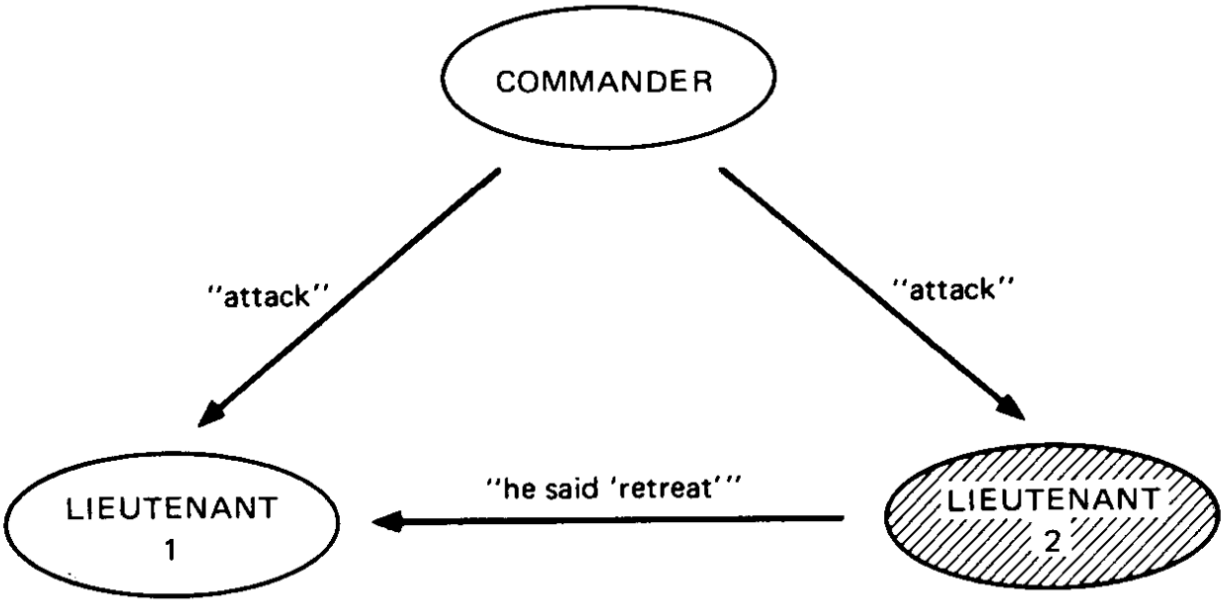


Рисунок 1.1 – Приклад проблеми візантійських генералів

На рисунку 1.1 зображено приклад проблеми візантійських генералів, за умови, що зрадник – це другий лейтенант.

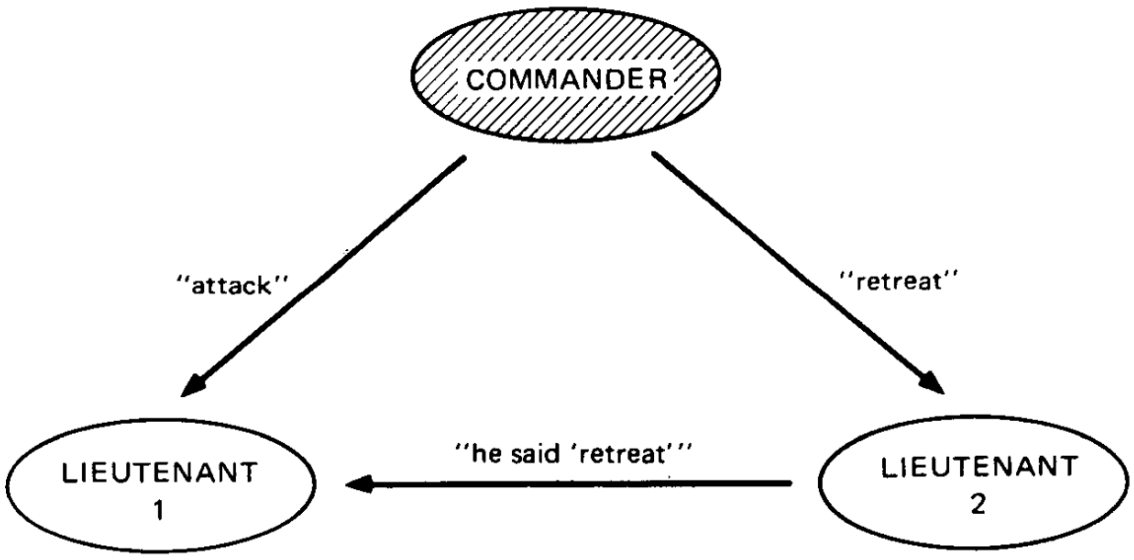


Рисунок 1.2 – Приклад проблеми візантійських генералів

На рисунку 1.2 зображено приклад проблеми візантійських генералів, за умови, що зрадник – це головний генерал.

1.3 Огляд альтернативних рішень в галузі консенсус протоколів

Останім часом все більшої популярності набуває використання та побудова розподілених систем на базі моделі акторів. В загальному випадку це дозволяє вирішити проблеми розподілених систем та особливо проблему досягнення консенсусу, за умови правильного використання даної моделі.

Модель акторів побудована на ідеї, що все навколо є акторами. Це схоже на філософію об’єкта Everything is, яку використовують деякі об’єктно-орієнтовані мови програмування.

Актор - це обчислювальна сутність, яка у відповідь на отримане повідомлення може одночасно:

а) надсилати кінцеву кількість повідомлень іншим акторам;

б) створити кінцеву кількість нових акторів;

в) визначити поведінку, яка буде використовуватися для наступного отриманого повідомлення.

Передбачуваної послідовності вищезазначених дій немає, і їх можна виконувати паралельно.

Відокремлення відправника від надісланого повідомлення було фундаментальним прогресом моделі акторів, що дозволило легко використати асинхронні комунікації та структури управління як шаблони передачі повідомлень. Споживачі повідомлень ідентифікуються за адресою, яку іноді називають «поштова адреса». Таким чином, актор може спілкуватися лише з акторами, чиї адреси він має. Він може отримати такі адреси із повідомлення. Модель акторів характеризується притаманною паралельністю обчислень всередині та між акторами, динамічним створенням акторів, включенням адрес акторів у повідомлення та взаємодією лише через пряму асинхронну передачу повідомлень без обмежень щодо порядку надходження повідомлення. Ключовим нововведенням було використання поведінки, визначеної як математична функція, щоб виразити те, що робить актор, коли він обробляє повідомлення, включаючи визначення нової поведінки для обробки

наступного повідомлення, яке надходить. Введення поняття поведінки забезпечило механізм для математичного моделювання спільного доступу в паралельному режимі.

Модель акторів можна використовувати для вирішення проблем консенсусу. Для цього необхідно корректно спроектувати систему та накласти вимогу, що кожен модуль повинен інтегруватися в систему акторів. Дана вимога не завжди є зручною, тому що інколи сервіс чи модуль погано проектуються на моделі акторів. Особливо це стосується сервісів, що виконують агрегацію та аналітику для великих обсягів даних.

1.4 Сімейство протоколів Paxos

Paxos — це набір протоколів для розв’язування задач консенсусу в мережі, де можливі збої та помилки в роботі обчислювальних приладів.

Сімейство протоколів Paxos розглядає варіанти розв'язання задачі консенсусу в залежності від кількості вузлів системи, кількості затримок для отримання результату, активності учасників, кількості надісланих повідомлень та типів можливих відмов. Результат відмовостійкого консенсусу не визначений (тобто, за певних умов обчислювачі не зможуть дійти згоди), проте гарантується безпека, іншими словами – консистентність, а умови, за яких консенсус неможливий, дуже рідкісні.

Алгоритми даного сімейства гарантують 3 наступні показники:

а) нетривіальність – рішення може бути прийняте тільки із заздалегідь заданої множини результатів;

б) консистентність – прийняте рішення збігається у всіх обчислювачів, які в ньому брали участь;

в) живучість(C,L) – для запропонованого рішення C рано чи пізно вузол L прийме якесь рішення (за умови наявності достатньої кількості працюючих вузлів).

Щоб спростити представлення базового варіанта алгоритму Paxos, опишемо такі визначення та передумови:

а) кожен окремий вузол розподіленої системи складається з обчислювального елемента, що має наступні властивості:
   1) працює на довільній швидкості;
   2) має можливість збою;
   3) має можливість приєднатися та продовжити роботу після збою (відповідно до моделі збою аварійного відновлення);
   4) не може вступати в об’єднання або групи з іншими обчислювальними елементами;
   5) не намагається навмисне оперувати неконсистентними даними (проблема візантійських генералів).

б) обмін даними відбувається через мережу Інтернет та має наступні властивості:
   1) обмін даними та повідомленнями із будь-яким вузлом системи;
   2) обмін даними відбувається в асинхронному режимі;
   3) можливі затримки даних в мережі;
   4) дані доставляються без пошкоджень (проблема візантійських генералів).

Як правило, алгоритм консенсусу може досягти прогресу при наявності 2F+1 вузлів, незважаючи на одночасний збій будь-яких F учасників системи. Однак, використовуючи переналаштування, можна використовувати протокол, який витримує будь-яку кількість повних збоїв доти, доки не відбудеться збій не більше F вузлів одночасно.

Сімейство протоколів Paxos визначає дії процесорів за їхніми ролями в протоколі: клієнт, акцептор, пропоузер, фолловер та лідер. У типових реалізаціях один процесор може одночасно грати одну або кілька ролей. Це не впливає на правильність протоколу — зазвичай поєднуються ролі для зменшення затримки та/або кількості повідомлень у протоколі.

Розглянемо кожну роль окремо:

а) клієнт – видає запит розподіленій системі та чекає відповіді. Наприклад, запит на запис у файл на розподіленому файловому сервері;

б) акцептор – виступає відмовостійкою «пам'яттю» протоколу. Акцептори збираються в групи, які називають кворумами. Будь-яке повідомлення, відправлене акцептору, має бути відправлене до кворуму акцепторів. Будь-яке повідомлення, отримане від акцептора, ігнорується, доки не буде отримано копію від кожного акцептора в кворумі;

в) пропоузер – захищає запит клієнта, намагаючись переконати акцепторів узгодити цей запит, і діє як координатор для просування протоколу у разі виникнення конфліктів;

г) фолловер – виступає фактором реплікації протоколу. Після узгодження клієнтського запиту акцепторами фолловер може вжити дій (тобто виконати запит і надіслати відповідь клієнту). Для покращення доступності обробки можна додати додаткових фолловерів;

д) лідер – протокол вимагає єдиного заявника (так званого лідера), щоб досягти прогресу. Багато процесів можуть вірити, що вони є лідерами, але протокол гарантує прогрес лише в тому випадку, якщо тільки один з них, зрештою, буде обраний. Якщо два процеси вважають, що вони є лідерами, вони можуть зупинити протокол, постійно пропонуючи конфліктуючі оновлення. Однак у цьому випадку властивості безпеки зберігаються.

Розглянемо поняття кворуму. Кворумом є більшість із усіх членів кластера.

$$Q = \frac{N}{2} + 1$$

Наприклад, якщо кластер має 6 учасників, кворумом буде 4. Кворуми виражають властивості безпеки алгоритму, забезпечуючи збереження інформації про результати принаймні в деяких процесорах, що є працездатними на даний момент часу.

Кворуми визначаються як підмножини безлічі акцепторів таким чином, щоб будь-які дві підмножини (тобто будь-які кворуми) мали принаймні один загальний елемент. Як правило, кворум є будь-якою більшістю акцепторів, що беруть участь в прийнятті рішення. Наприклад, розглянемо множину акцепторів {A,B,C,D}, кворумом більшості для якої будуть підмножини з трьох довільних акцепторів:

{A,B,C}, {A,C,D}, {A,B,D} або {B,C,D}. У загальному плані акцепторам та кворуму можна присвоїти довільні додатні вагові коефіцієнти так, щоб кожна підмножина акцепторів мала сумарну вагу більшу від половини загальної ваги всіх акцепторів.

Кожна спроба визначення узгодженого значення V здійснюється за допомогою пропозицій, які можуть бути ухвалені або не прийняті акцепторами. Кожна пропозиція є унікальною для цього заявника. Значення, що відповідає пронумерованій пропозиції, може бути обчислене в рамках виконання протоколу Paxos, але не обов'язково.

Розглянемо кроки базового алгоритму Paxos:

а) prepare (пропозиція). На цій фазі пропоузер генерує «пропозицію» з порядковим номером N і відправляє її всім акцепторам. Для кожної з наступних «пропозицій» номер N повинен бути більшим за обраний раніше;

б) promise («обіцянка»). Кожен акцептор отримує «пропозицію» з порядковим номером N та значенням V. Якщо номер «пропозиції» більший, ніж усі прийняті раніше даним аксептором, він зобов'язаний відповісти на це повідомлення «обіцянкою» не приймати більше «пропозицій» з порядковим номером менше N. Якщо даний акцептор вже приймав якусь «пропозицію», він повинен повернути номер N_i цієї «пропозиції» і прийняте значення V_i, інакше він повертає порожнє значення;

в) accept («Прийняти»). Якщо пропоузер отримав «обіцянки» від кворуму акцепторів, він вважає запит готовим до подальшої обробки. У випадку, якщо з «обіцянками» від аксептору прийшли також значення N_i та V_i, пропоузер вибирає V рівну значенню V_i «обіцянки» з максимальним N_i. Потім пропоузер відправляє запит «прийняти» всім акцепторам, який містить значення N та V;

г) accepted («прийнято»). Коли акцептор отримує повідомлення «прийняти» зі значеннями N і V, він приймає його тільки в тому випадку, якщо раніше він не «обіцяв» приймати пропозиції з номерами строго більше за N. В іншому

випадку він набуває значення і відповідає повідомленням «прийнято» всім фоловерам.

Рисунок 1.3 – Приклад роботи базового алгоритму сімейства Paxos

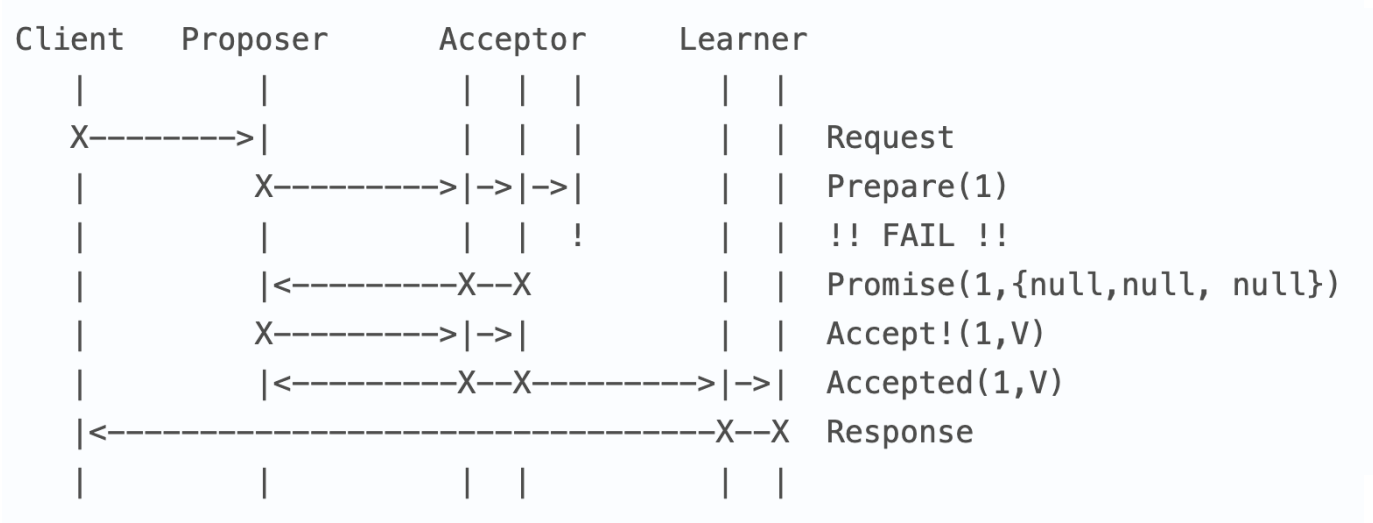


Рисунок 1.4 – Приклад роботи базового алгоритму сімейства Paxos, за умови відмови акцептора

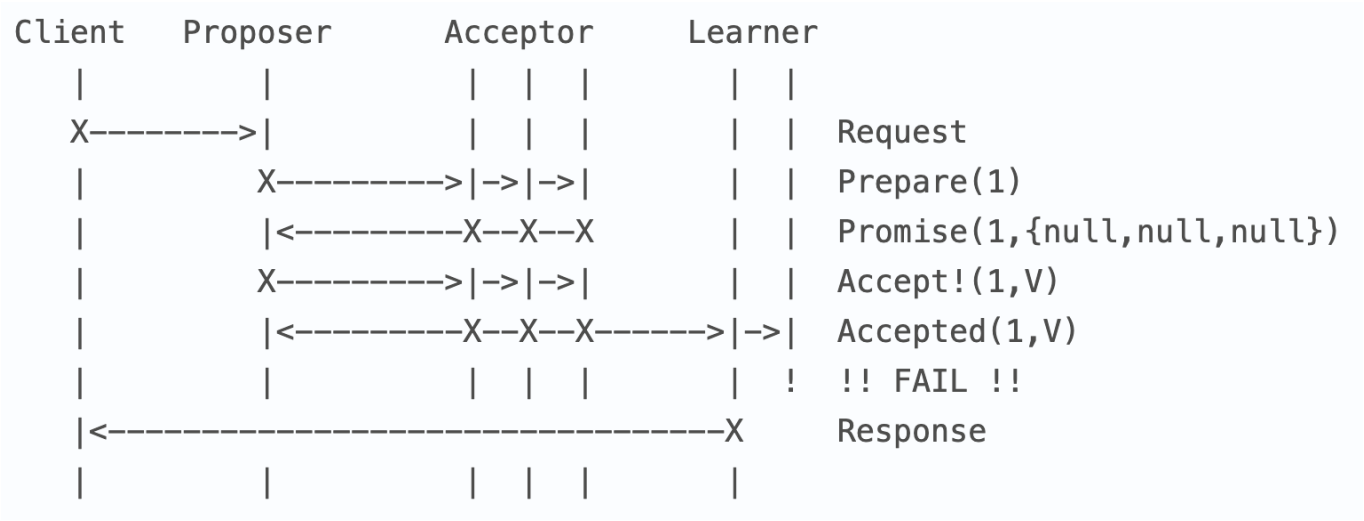


Рисунок 1.5 – Приклад роботи базового алгоритму сімейства Paxos, за умови відмови акцептора

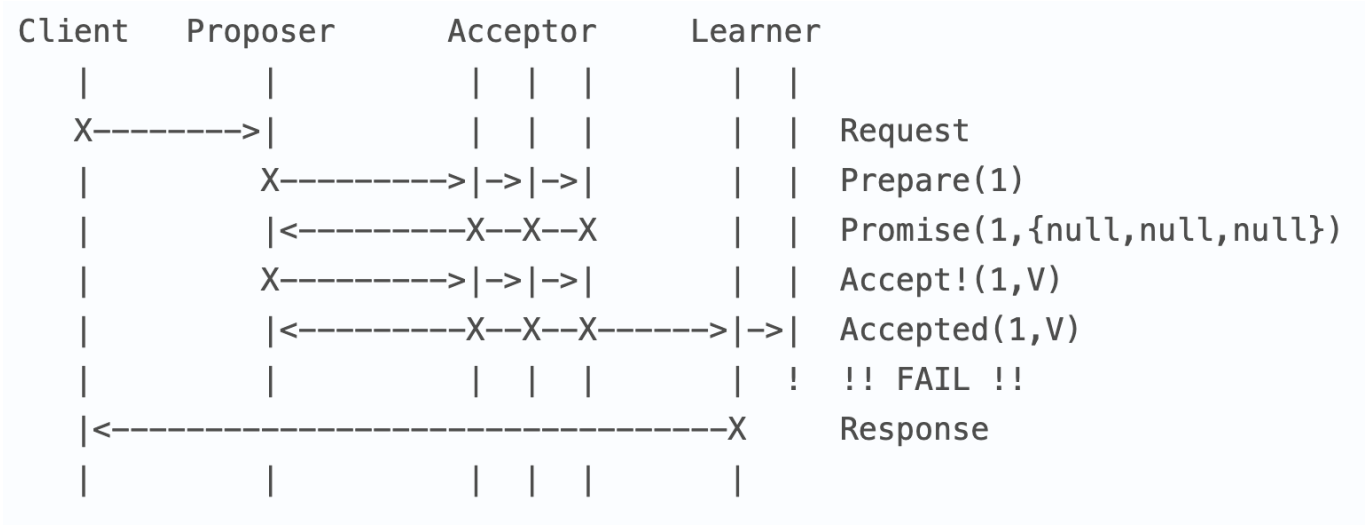


Рисунок 1.6 – Приклад роботи базового алгоритму сімейства Paxos, за умови відмови фолловера

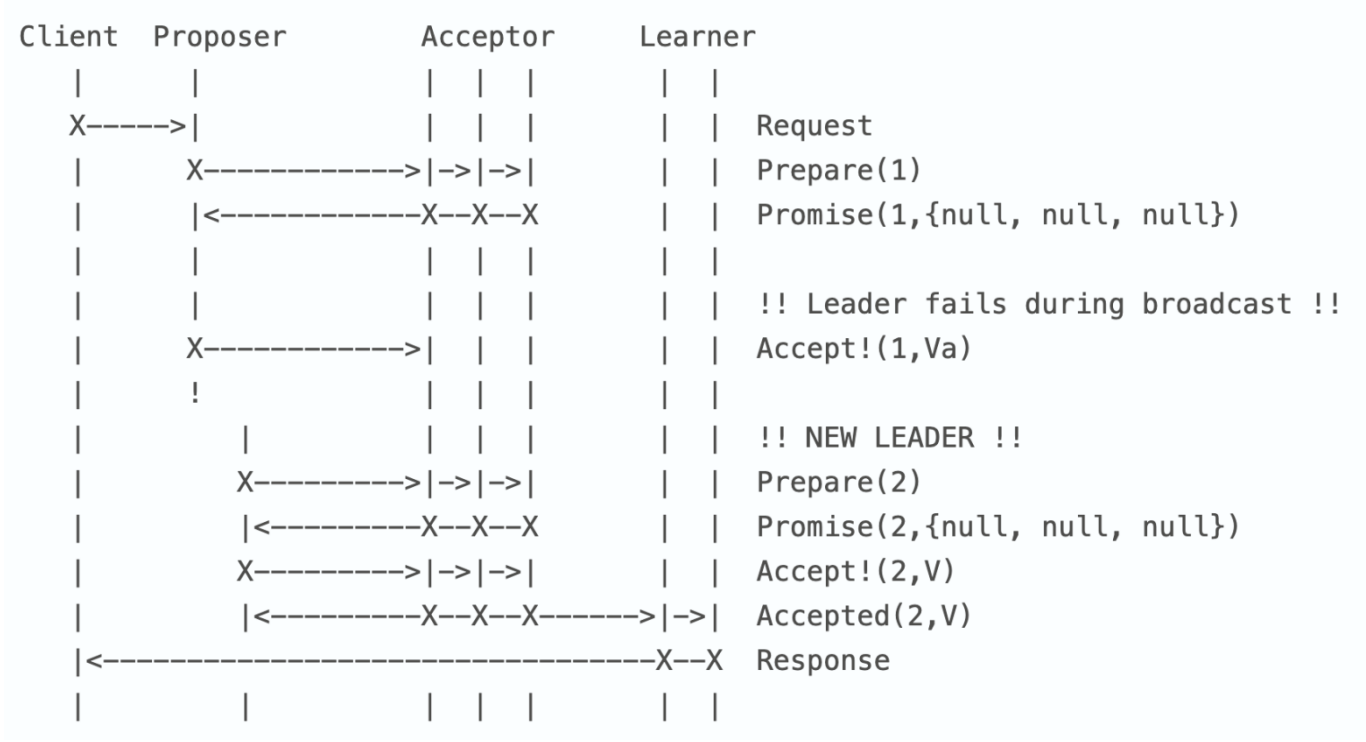


Рисунок 1.7 – Приклад роботи базового алгоритму сімейства Paxos, за умови відмови пропоузера

У разі відмови пропоузера, система повинна вибрати нового пропоузера, зазвичай це здійснюється шляхом голосування після закінчення тайм-ауту очікування повернення старого пропоузера. У випадку, якщо після вибору нового пропоузера старий повертається до життя, між лідерами може виникнути конфлікт, який може призвести до зациклювання системи (рисунок 1.8).

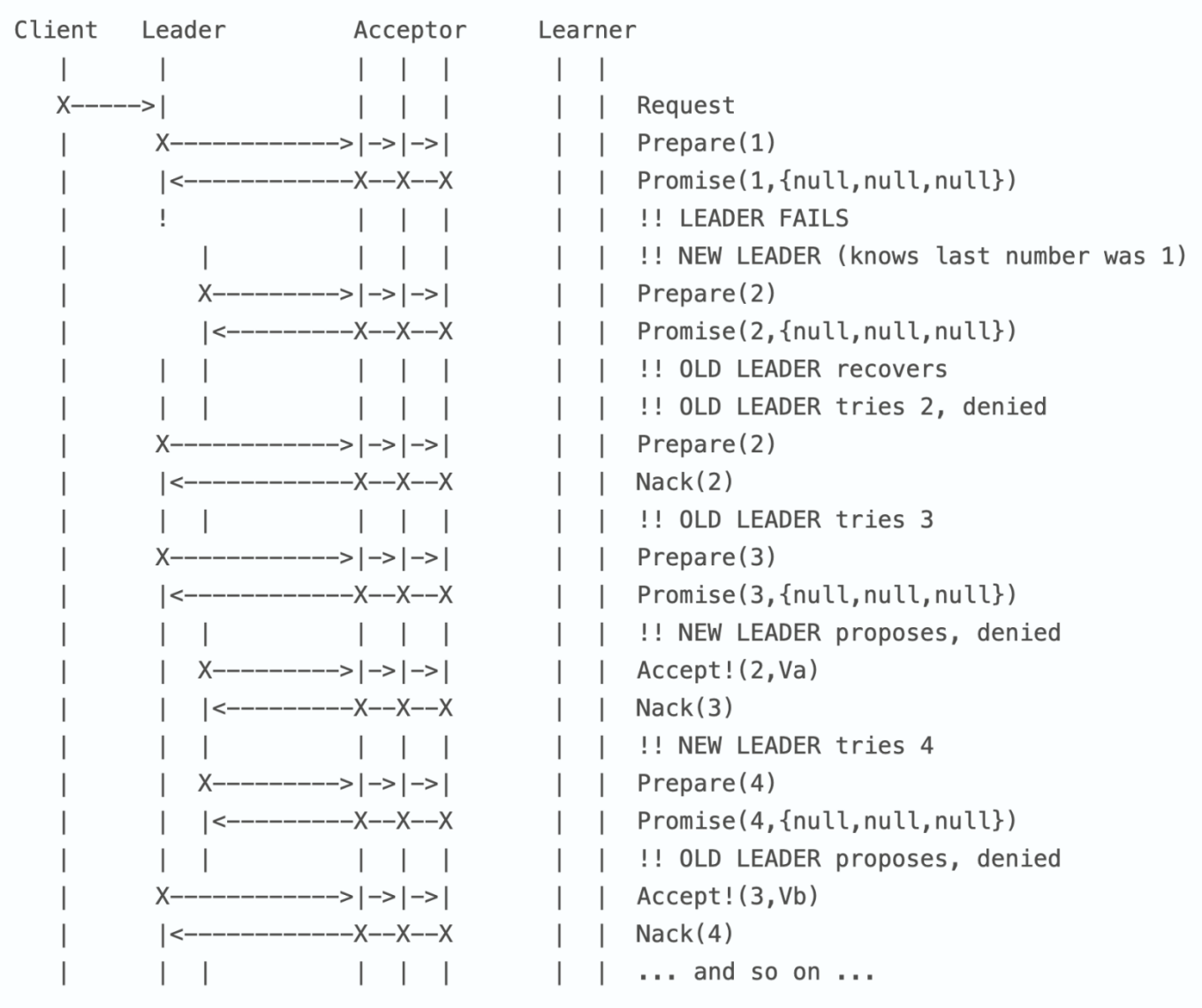


Рисунок 1.8 – Приклад роботи алгоритму під час відмови пропоузера

Окрім базового алгоритма, існують ще багато модифікацій протоколу Paxos. Кожна з таких модифікацій спрямована на покращення в певних умовах використання та вирішення певних специфічних класів проблем.

Таблиця 1.1 — Модифікації алгоритму Paxos

Алгоритм	Опис
Disk Paxos	Кожен реплікований компонент у мульти-декретному Paxos є процесом, який активно надсилає та отримує повідомлення. Disk Paxos — це варіант Paxos, який замінює акцепторні процеси на диски, які підтримують лише операції блоку читання та запису для збереження стану кворуму. Таким чином, Disk Paxos не потребує окремих процесів-акцепторів. У Disk Paxos кожен лідер володіє блоком на кожному диску, на який він може записувати свої повідомлення. Щоб запустити фазу 1 протоколу Synod, лідер виконує наступне для кожного диска. Лідер записує повідомлення play свій власний блок на диску та читає блоки інших лідерів на тому самому диску, щоб перевірити, чи є повідомлення p1a з більшим порядковим номером. Якщо лідер не виявить більшого порядкового номера на більшості дисків, його пропозиція приймається. Якщо він виявляє повідомлення p1a з більшим порядковим номером, він починає знову з більшим порядковим номером. Для фази 2 лідер повторює той самий процес із повідомленнями p2a, щоб визначити, чи прийняті його пропозиції.

Продовження таблиці 1.1

Cheap Paxos	Багатодекретний Paxos вимагає 2f+1 акцепторів і f+1 реплік, щоб витримувати f збоїв. Дешевий Paxos використовує тільки f+1 акцептори в поєднанні з f дешевими допоміжними акцепторами, які залишаються бездіяльними під час нормального виконання, якщо не відбувається збій. Дешевий Paxos — це різновид реконфігурованого алгоритму Paxos з кількома декретами, який спирається на спостереження, що лідер може надсилати свої повідомлення p1a та p2a фіксованому кворуму акцепторів. Поки всі приймачі в цьому кворумі працюють і можуть спілкуватися з лідерами, немає потреби, щоб приймачі, які не в кворумі, нічого не робили. Дешевий Paxos керує протоколом Paxos з кількома декретами з фіксованим кворумом f+1 акцепторів. При підозрі на збій акцептора у фіксованому кворумі, допоміжний акцептор стає активним, так що знову є f+1 реагуючих акцепторів, які можуть сформувати кворум. Цей новий кворум завершує виконання будь-яких нерозглянутих пропозицій і переналаштовує систему, замінюючи акцептор, що мів вийти із ладу на новий, повертаючи систему до нормального виконання.

Продовження таблиці 1.1

Fast Paxos	У Paxos кожне рішення приймає принаймні три затримки повідомлення між тим, коли репліка пропонує команду, і коли деяка репліка дізнається, яка команда була обрана. Більше того, якщо кілька лідерів працюють одночасно, потрібно чотири затримки повідомлення, щоб команди, що зіткнулися, були призначені різним вузлами. Швидкий Paxos – це варіант Paxos, який зменшує наскрізні затримки повідомлень, що виникають в репліці, щоб дізнатися вибране значення. Навчання відбувається через дві затримки повідомлення, коли немає зіткнення, і через три затримки повідомлення, якщо затримка є. Для досягнення такої поведінки, необхідно 3f + 1 акцепторів замість 2f + 1. У Fast Paxos репліка надсилає свою пропозицію безпосередньо акцепторам, минаючи лідера. Fast Paxos розрізняє швидкі та класичні пропозиції. У швидких пропозиціях, після завершення першого етапу лідер надсилає приймачам повідомлення p2a без значення. Одночасно репліки надсилають свої пропозиції безпосередньо акцепторам. Коли акцептор отримує таке повідомлення p2a, він може вибрати будь-яку пропозицію репліки як значення для прийняття. За відсутності колізій акцептори в кінцевому підсумку приймають ті самі пропозиції, і наскрізна затримка повідомлення для репліки зменшується з трьох затримок повідомлення до двох затримок повідомлення. Ця схема може вийти з ладу, якщо репліки одночасно надсилають різні пропозиції, а акцептори приймають конфліктні значення. У цьому випадку для відновлення можна використовувати класичний підхід до пропозицій, який працює так само, як і в звичайному Paxos.

Продовження таблиці 1.1

Stoppable Paxos	Paxos з можливістю зупинки реалізує реплікований кінцевий автомат в стані зупинки. Зупинки виконання реплікованого кінцевого автомата можна досягти за допомогою спеціальної команди — як тільки буде прийнято таке рішення, команди більше не виконуються. Цей варіант передбачає альтернативний варіант реконфігурації: зупинити поточний реплікований кінцевий автомат і запустити новий, використовуючи кінцевий стан програми попереднього. Таким чином, реплікований кінцевий автомат можна розглядати як послідовність зупинених автоматів, кожен з яких працює з фіксованою конфігурацією.

Продовження таблиці 1.1

Mencius	Менцзи намагається вирішити вузьке місце з одним лідером, яке виникло в Паксосі з багатьма декретами. У Mencius репліки, акцептори та лідери розміщені разом, організовані в логічне кільце та попередньо призначені номери слотів, щоб пропонувати свої команди. Таким чином, лідери по черзі пропонують команди, збільшуючи пропускну здатність, особливо коли система пов'язана з процесором. Оскільки початковий лідер для слоту відомий, у фазі 1 немає потреби, а лідеру потрібно виконати лише фазу 2. Якщо у нього немає команди, яку можна запропонувати, лідер пропонує команду без операції. Якщо лідер несправний або повільний, інші лідери можуть взяти на себе слот, виконавши фазу 1, але вони можуть лише запропонувати команду без операції. Використовуючи ці знання, Mencius дозволяє реплікам дізнаватися про команду без операції, запропоновану несправним лідером, за одну затримку повідомлення. При нормальному виконанні без збоїв процесів продуктивність Mencius подібна до Paxos з кількома декретами зі стабільним лідером, без вузького місця лідера. Однак бездіяльні та повільні лідери можуть знизити загальну продуктивність. Щоб обмежити кількість неопераційних повідомлень, які генерують лідери, Mencius використовує договори оренди між лідерами, коли лідери можуть передавати свої пропозиції іншим лідерам для всіх слотів, менших за вказане число слотів. Це може бути зроблено добровільно, якщо непрацюючий лідер відмовляється від своїх бюлетенів, або агресивно, якщо швидкий лідер забирає пропозиції повільного лідера.

Продовження таблиці 1.1

Vertical Paxos	Вертикальний Paxos — це варіант Paxos, який дозволяє реконфігурувати процес виконання, поки реплікований кінцевий автомат активний і приймає рішення щодо команд. Vertical Paxos використовує допоміжний провідний для прийняття рішення щодо операцій реконфігурації, який визначає набір акцепторів і лідера для кожної конфігурації. У Vertical Paxos лідер встановлюється для кожної конфігурації, а різні номери пропозицій мають різні конфігурації. На відміну від стандартних «горизонтальних» алгоритмів Paxos, кожна конфігурація має рівно один номер пропозиції та пов’язаний з ним лідер, а слот може мати більше однієї конфігурації. У горизонтальних алгоритмах Paxos конфігурації можуть змінюватися лише тоді, коли відбувається горизонтальний перехід від одного слота до іншого. У Vertical Paxos конфігурації змінюються, коли відбувається рух по вертикалі, але залишаються такими ж, як при русі по горизонталі.

Кінець таблиці 1.1

Egalitarian Paxos	Egalitarian Paxos (EPaxos) — це ще один варіант Paxos, який намагається вирішити вузьке місце з єдиним лідером, дозволяючи всім реплікам отримувати значення від клієнтів і дозволяючи їм пропонувати значення з однією затримкою повідомлення, коли немає залежності між командами. Це дозволяє системі рівномірно розподіляти навантаження на всі репліки; крім того, він пропонує кращу продуктивність для геореплікованих кінцевих автоматів, дозволяючи клієнтам надсилати командні запити до найближчих до них реплік. На відміну від інших варіантів Paxos, EPaxos впорядковує команди динамічно та децентралізовано. EPaxos передбачає, що репліки, акцептори та лідери розташовані спільно. У процесі пропонування команди для номера слота репліка додає до цієї команди обмеження впорядкування, тому кожна репліка може використовувати ці обмеження, щоб самостійно досягти того ж порядку локально. EPaxos працює у дві фази: попередньо прийняти та прийняти. Фаза попереднього прийняття встановлює обмеження впорядкування для команди c. Отримавши c від клієнта, репліка стає лідером команд і надсилає повідомлення попереднього прийняття, включаючи c, його залежності depc і порядковий номер seqc до 2f реплік. depc — це список усіх екземплярів, які містять команди, які заважають c, а seqc — це порядковий номер, більший за порядковий номер усіх команд, що заважають у depc.

1.5 Висновок до розділу

Теорія побудови розподілених систем, вивчення та визначення їхніх проблем вивчалися в багатьох роботах та наразі продовжують бути актуальною темою для досліджень. Задачі для розробників таких систем стоять досить складні, тому що проблеми відмовостійкості, консистентності та високого рівня доступності є актуальними проблемами і на теперішній день.

Розглядаючи проблему досягнення консенсусу в розподілених системах, можна виділити дві основні проблеми даного протоколу – це проблема візантійських генералів та відмова одного з вузлів системи. Кожна із цих проблем має різні підходи до її вирішення та кожне із розв’язків має свої переваги та недоліки. Оптимального підходу, що задовольнить вимоги всіх існуючих кластерних систем, немає, тому кожна розподілена система вибирає саме той підхід, що є найбільш підходящим для вирішення поставлених перед нею задач та цілей.

Наразі існує сімейство протоколів та методів вирішення проблеми консенсусу в розподілених системах – Paxos. Кожен з даних підходів націлений на певні оптимізації стандартного протоколу і має певні недоліки, що були розглянуті вище.

2 МАТЕМАТИЧНА МОДЕЛЬ

2.1 Модель розподіленої системи

Розподілені системи можна ідентифікувати в рамках різних контекссстів та сценаріїв. Комп'ютерну мережу великої корпорації, можна розглядати як розподілену систему. Надання програмних додатків для підтримки електронної комерції, дистанційної освіти та електронного уряду, через широковикористовані комп’ютерні мережі, такі як мережа Інтернет. Крім того, поштові системи, в яких здійснюється відправлення, обробка та доставка листів за своєю суттю також є розподіленими.

Отже, можна сказати, що розподілена система є множиною слабо пов’язаних автономних об’єктів, фізично розташованих у різних місцях світу. В даній ситуації, термін об’єкт відповідає абстракції, яка може представляти різні типи сутностей, такі як люди, інтелектуальні агенти, компоненти програмного забезпечення або одиниці обробки інформації.

Термін розташування означає посилання на фізичне або віртуальне положення кожного об'єкта. Досить зручно припустити, що існує 2 види розподілених систем - системи які за своєю природою є розподіленими та системи, які розподілені внаслідок рішення прийнятого при побудові таких систем.

2.1.1 Розподілена в часі система

Центральне питання формального моделювання розподілених систем полягає у ідентифікації структури і ролі часу, оскільки різні сімейства розподілених систем вимагають чітких визначень структур часу для вираження поведінки їх складових об'єктів. Розглянемо роль часу в розподілених системах:

а) T – непорожня множина, що визначає часову область;

б) T×T – відношення (n порядку), при цьому $T, \preccurlyeq$ задовольняє наступним аксіомам:

1) (R1) Рефлексивність: для кожного $x \in T$, $x \leq x$;

2) (R2) Антисиметрія: для кожного $\{x, y\} \subseteq T$, $x \leq y \wedge y \leq x \rightarrow x = y$;

3) (R3) Транзитивність: для кожного $\{x, y,z\} \subseteq T$ $x \leq y \wedge y \leq z \rightarrow x \leq z$.

в) L — набір лінійних потоків часу: кожен $\lambda \in L$ — функція з $\mathrm{dom}\ \lambda \in P+ (T)$ і $\lambda \in I+ (L)$, для деяких фіксованих hL, що задовольняє (R1, R2, R3) та наступні вимоги:

1) лінійність: для кожного $\{x, y,z\} \subseteq L$, $x\ y \rightarrow z\ x \vee y\ z$;

2) ін'єктивність: для кожного $\{x, y\} \subseteq \mathrm{dom}\ \lambda\ \lambda(x) = \lambda(y) \rightarrow x = y$;

3) юр'єктивність: для кожного $y \in \mathrm{cod}\ \lambda$, $\exists x \cdot x \in \mathrm{dom}\ \lambda \wedge \lambda(x) = y$;

4) монотонність: для кожного $\{x, y\} \subseteq \mathrm{dom}\ \lambda$, $x \leq y \rightarrow \lambda(x)\ \lambda(y)$.

Дискретні часові рамки, які зазвичай використовуються при розробці паралельних і розподілених систем, можна класифікувати відповідно до їх лінійної або розгалуженої природи. Розгалуження часу вважалося зручним каркасом не тільки для реалізації інструментів перевірки моделі, а також для специфікації одного типу властивості для активностой, які виражають те, що може відбуватися в поточний момент часу. З іншого боку, лінійний час, завдяки своїй простоті, є основним припущенням теорії, але незважаючи на свою простоту дозволяє дуже добре виразити спроможність для задоволення мінімальних достатніх передумов для конкретної неминучої події.

Дане визначення легко відображає лінійні або розгалужені часові рамки. Зокрема, припущення про лінійність фіксується, коли L (потік часу) є єдиним для всіх учасників. Як факт того, що L і T залишаються невизначеними в такій теорії, дає можливість висловити додаткові припущення щодо дискретності та щільності часу. Однак немає сенсу вимагати, щоб $hT,\leq$і підпорядковувалося лише обмеженням з урахуванням властивостей фізично розподілених систем для яких ідеалізовані часові рамки є уявними.

Насправді, прийняття функцій лінеаризації в L можна розглядати саме як механізм для відображення структур часу вищих вимірів у четвертий вимір простір-час, що відповідає стандартному припущенню щодо часових потоків.

2.1.2 Формальна модель розподіленої системи

Роль часу в розвитку розподілених систем полягає саме в тому, щоб дозволити провести асоціацію автономних об'єктів з відповідними способами поведінки та розміщення. Розглянемо детальніше дану ідею:

Структура об’єкта — це кортеж із 5-ти елементів (T,V,S,B,P), визначений як непорожні множини T, Ev та Loc, що ніколи не перетинаються, де:

а) T = hhT,≤i,Li – часовий проміжок;

б) V — структура події, визначена в термінах області подій Ev;

в) S є структурою розташування, визначеною в термінах домену розташування Loc;

г) B : Ev → P I+ (cod L) визначає поведінку;

д) P : Loc → P I+ (cod L) визначає місце розташування.

Максимальність: для кожного e ∈ Ev i s ∈ Loc, ((∩ B(e)) = { })∧((∩ P(s)) = { }.

Структури, визначені вище, називаються частковим порядком, оскільки вони покладаються на визначення базових відносин часткового порядку як їх часових рамок, T . Потоки часу, отримані з данних кадрів, L, використовуються для призначення часу залежної інтерпретації подій і місць, через B і P відповідно. Наразі залишаються невизначеними події і розташування структур V і S, оскільки вони цілком залежні від області застосування: події можуть містити значення та розташування, що можуть мати ієрархічну структуру, наприклад, представляти реалістичні моделі взаємодії і вкладені географічні регіони відповідно.

2.2 Моделювання проблеми консенсусу

2.2.1 Змістовна постановка задачі

Нехай, процес починається із вхідного значення, припустимо, це значення задається при старті самого процесу. Усі процеси повинні узгоджувати загальне вихідне значення, яке має бути вхідним значенням деякого іншого процесу.

Для простоти постановки задачі, розглянемо двійкову систему. У двійковій системі вхідними значеннями можуть бути 1 або 0. Уявімо, що існує n + 1 процесів. Представимо вхідну систему P у вигляді симплексу n-порядку, що має n вершин, де кожне значення вершине відображає запропоноване значення (вхідне значення). Для вирішення проблеми консенсусу, потрібно створити таку функцію, яка перетворить симплекс n-порядку на симплекс n+1-порядку, де n+1 і буде консенсус значенням.

$$P^n \rightarrow P^{n+1}$$

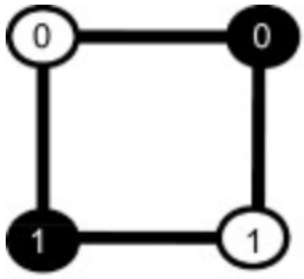


Рисунок 2.1 – Симплекс n-порядку

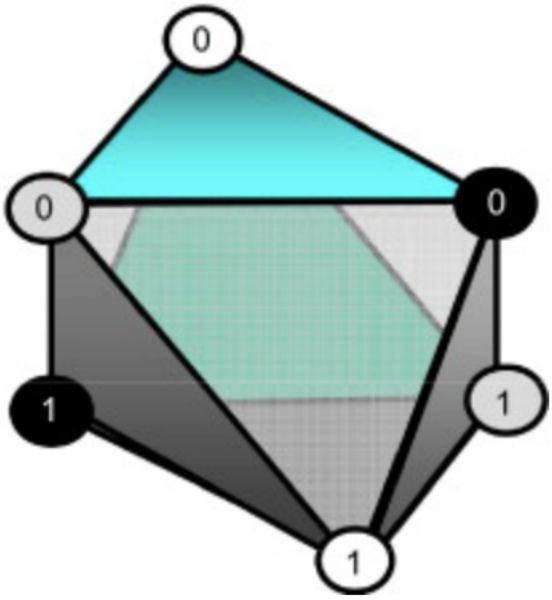


Рисунок 2.2 – Симплекс n+1-порядку

2.2.2 Побудова методу для досягнення консенсусу для вирішення поставленої задачі

Для вирішення поставленої задачі, будемо базуватися на підході із виділенням лідера кластеру, що буде виконувати обов’язки оркестрування системою та контролювати роботу сервісів.

Введемо поняття кандидат – це будь-який вузол, що не є лідером, але знаходится в консистентному стані і не відстає на задану велечину від журналу стану лідера. Ідея кандидата полягає в тому, щоб відокремити неконсистенті та відстаючі вузли від вузлів, що співпадають з лідером і у випадку його виходу з ладу спростити та пришвидшити процес вибору нового лідера, завдяки відокремленню актуальних вузлів від неконсистентних.

Розглянемо процес вибору лідера серед наявних вузлів. При першому запуску системи, кожен із її учасників знаходится у стані послідовника. Коли виходить час на отримання запиту, що називається heartbeat, кожен із учасників переходить в стан

кандидата та пропонує свою кандидатуру, передаючи всім членам свій останній лог із копії журналу стану. Учасник із найбільшим порядковим номером логу стає лідером кластеру та синхронізує свій журнал стану із всіма вузлами.

Лідер підтримує механізм серцебиття (heartbeat) та надсилає запит кожному кандидату та послідовнику, якщо даний запит не прийшов вчасно, запускається механізм вибору лідера. Метод надає можливість гнучко задати проміжки часу та періодичність надсилання даних запитів. Це дозволяє вирішити проблему часткових затримок мережі. Також необхідно розглянути умову, що мережа, в якій знаходится лідер стає недоступною та лідер кластеру не отримує відповіді протягом встановленого проміжку часу. В таких ситуаціях потрібно гарантувати максимальний рівень доступності і для цього лідер переходить в стан кандидата та система вибирає нового лідера.

Важливим нововведеням даного методу є механізм перевибору лідера для підтримки максимальної консистентності. Завдяки підтримки порядку логів та пошуку найбільшого порядкового номеру логу, стає можливим вибір максимально актуального вузла, як лідера кластера.

Також даний метод дає можливість виконувати налаштування для вибору між швидкістю роботи та максимальним рівнем консистентності. Для гарантування максимального рівня консинтентності використавується підхід 2х фазного коміту, а для збільшення швидкості роботи алгоритму буде використанно поняття кворуму та необхідної наявності, лише більшості голосів.

Процес поширення повідомлень по системі відбувається за ініціативою лідера, після отримання більшості голосів від кандидатів та послідовників, лідер додає значення до журналу стану та поширює його для всіх вузлів системи.

Також, даний метод дає дуже зручну можливість динамічно додавати або видаляти члена кластера без необхідності перезапуску системи. Це дуже важлива можливість, оскільки для систем реального часу, високий рівень доступності – це нообхідність. Динамічне додавання або видалення член кластера може статися в будь-який момент роботи системи. Це можливо завдяки динамічному переконфігуруванню кластера. Додавання чи видалення саме по собі є спеціальним повідомленням, яке

лідер повинен транслювати аналогічно іншим повідомленням. Це особливе повідомлення, тому що замість клієнтського запиту на реплікацію логу, повідомлення містить нову конфігурацію системи, яку лідер генерує та передає всім членам кластера атомарно.

2.2.3 Формальна модель розподіленої системи Аналіз та порівняння розробленого методу із існуючими аналогами

Розроблений метод побудований на базі понять, що використовуються в базовій реалізації алгоритму, але володіє перевагами у вигляді гнучкої можливості конфігурації, що дозволяє точно налаштовувати систему під вимоги продукту та середи виконання. Також важливим нововведенням є особливий тип повідомлень, що дозволяє динамічно додавати вузли до системи не зупиняючи роботи продукту. Це дуже важлива особливість, яку можна використовувати під час роботи та розробки розподілених систем реального часу. Також метод дає можливість вибрати механізм роботи націлений на гарант консистентності та побудувати максимально стабільну систему, що буде основана на базі 2х фазного коміту.

2.3 Висновки до розділу

Була розглянута математична модель розподіленої системи та детально описана задача консенсус протоколу та проблеми, що можуть виникнути при реалізації та побудові методу досягнення узгодженості.

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

В даній роботі була проведена низка експериментально-дослідницьких випробовувань для визначення основних характеристик досліджуваної системи та виділення основних якостей поведінки в різних умовах та станах досліджуваної системи. Фокус експериментів націлений на визначення характеру та поведінки системи в умовах відсутності консенсусу. Дослідницька робота проводилася на базі розробленої тестової системи для синхронізації виконання роботи по таймеру.

Основною метою експерименту є виявлення властивостей розподіленої системи в умовах відсутності консенсусу та можливими станами під час досягнення консенсусу за різних умов функціонування системи.

Розглянемо основні цілі експерименту. Основною ціллю – є необхідність отримати знання про стани системи під час різних умов та станів системи. Важливим станом системи буде випадок мережевих проблем та низької пропускної здатності. Наразі мережа інтернет та використання стеку протоколів TCP/IP набуло великої популярності, хмарні системи також використовуються все активніше, також дають високий рівень гарантій, щодо надійності та стабільності мережі та інфраструктури, проте, проблеми за мережею бувають і в таких випадках систем. Тому дуже важливим буде перевірити стани та процес переходу та досягнення консенсусу, вибору лідера в таких умовах.

Для досягнення бажаних умов та передумов були використані сервіси, що штучним чином навантажили мережу інтернет та дозволили зменшити пропускну здатність, завдяки постійному потоку передачі даних по мережі. Також була проведена емуляція довгої роботи системи та відповідно уповільнене отримання повідомлень та ініціація перевибору лідера.

Розглянемо систему, що функціонує в нормальних умовах та кількість часу, що система знаходиться в різних проміжних та кінцевих станах.

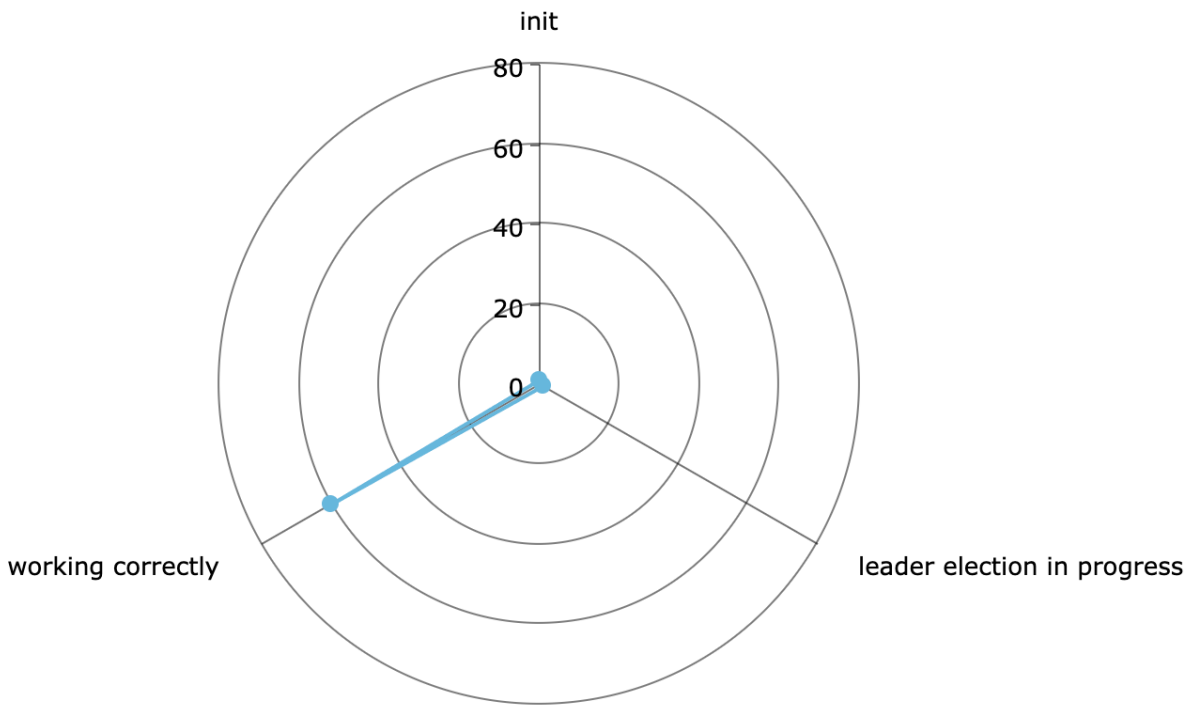


Рисунок 3.1 – Час знаходження системи в кінцевих та проміжних станах

При нормальній роботі мережі та інфраструктури, система більшість свого часу знаходиться в стані роботи та нормально функціонує.

Для зручності, за квант часу була взята 1 секунда. Згідно з результатами експерименту, система знаходилася 60 квантів часу в стані коректної роботи та по одному кванту часу в стані ініціалізації та процесу вибору лідера відповідно. Це можна пояснити стартом системи, під часу старту, система автоматично знаходится в стані ініціалізації та потім переходить у стан вибору лідера, а після успішного закінчення даного процесу, переходить в кінцевий стан.

Розглянемо ситуації при яких мережа працює некоретно та виникають збої під час обміну даними.

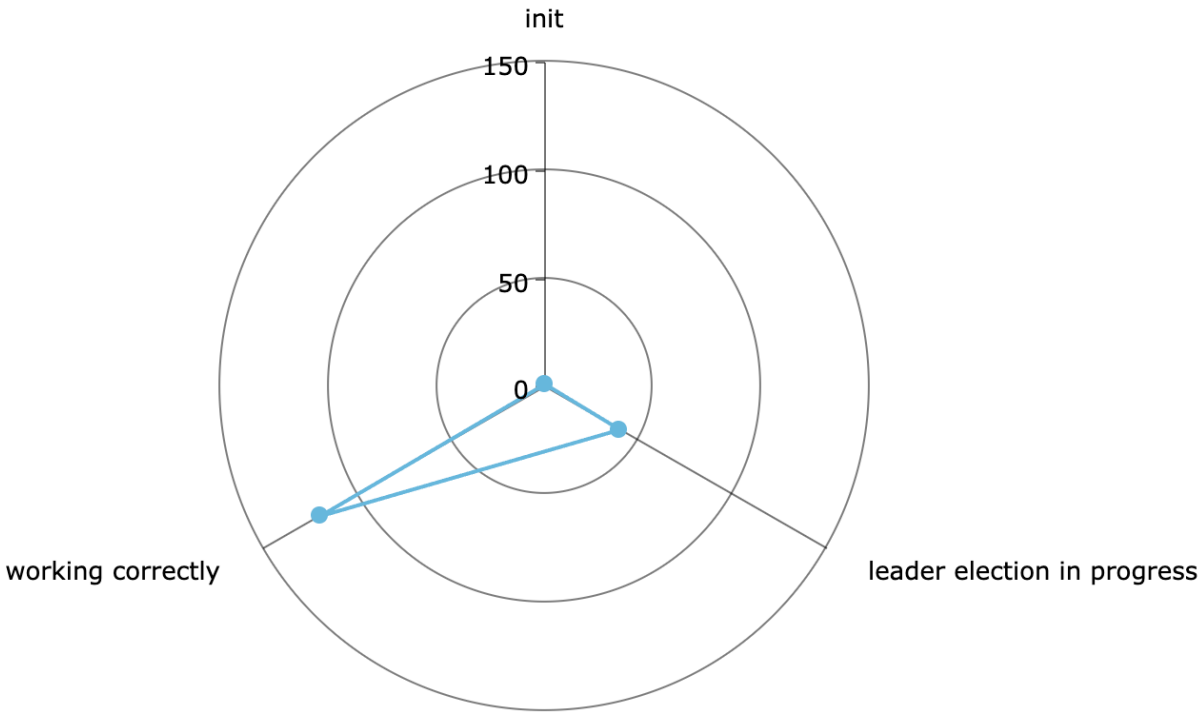


Рисунок 3.2 – Час знаходження системи в кінцевих та проміжних станах

Згідно з графіком час роботи алгоритму та час перебування в проміжних станах зростає. Як можна бачити на графіку, час, що система знаходилась в проміжних станах виріс майже до 50 квантів в порівнянні із попереднім експер. Ці результати показують пряму залежність методу та розробленого інструменту від мережі обміну данними. Під час навантаження, алгоритму потрібно більше часу на встановлення з'єднань та обміну повідомленнями, що напряму впливає на час досягнення консенсусу та переходу системи в кінцевий стан.

3.1 Висновки до розділу

Були проведені експерименти та аналіз роботи системи в різних умовах та конфігураціях мережі. Як можна бачити, під час навантаження, алгоритму потрібно

більше часу на встановлення з'єднань та обміну повідомленнями, що напряму впливає на час досягнення консенсусу та переходу системи в кінцевий стан.

4 ОПИС ПРОГРАМНОГО ТА ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Розробку програмного продукту (бібліотеки), який використовується для проведення аналізу та вирішення проблеми досягнення консенсусу в розподілених системах побудованих на базі платформи .Net, було вирішено проводити за допомогою мови програмування C#.

C# - це мова програмування, яка базується на об’єкто-орієнтованій парадигмі програмування. Це дозволяє створювати оптимальні рішення поставлених задач, які будуть використовувати основні переваги об’єктно орієнованої парадигми. Також дана мова програмування дуже динамічно розвивається і дає дуже потужну платформу з безліччю готових рішень стандартних проблем, що позитивно впливає на швидкість розробки, тестування та інтеграції бібліотеки. Також мова програмування C# надає дуже зручні засоби для роботи з мовою незалежно від операційної системи. Дана мова програмування є дуже популярною на данний момент і має дуже велику ком’юніті, що також є перевагою та допоможе при вирішенні технічних проблем під час розробки продукту.

Для розробки модулю бібліотеки, що буде виконувати роль транспорту, було використано протокол HTTP. Це дозволило спростити модель комунікації і скористатися всіми плюсами даного протоколу, таких як можливість легко перейти на використання HTTPS та досягти високого рівня безпеки, відсутність стану під час обміну даними, самоопис, що полегшує процес розробки, виправлення помилок та тестування.

Формат повідомлень представлений у вигляді JSON-документу, що полегшує процес розробки та тестування.

Для розробки модуля, що відповідає за форматування повідомлень, була використана бібліотека Newtonsoft.Json, що наразі є основним JSON серіалізатором платформи .Net. Даний інструмент надає багато переваг, таких як легка можливість конфігурування та легкість використання.

Для побудови бібліотеки був використаний фреймворк .Net 5. Наразі дана версія платформи є актулаьною, та дає величезну базу готових рішень, таких як вбудований di container, можливість конфігурування з використанням файлів, значень середи та агрументів. Також даний фрейморк дає IOC, що дозволяє будувати легкі в підтримці додатки.

Для розробки і тестування створеної бібліотеки, була створена розподілена система та було використано технологію docker-compose, що дозволило легко підняти кластер із розроблених сервісів та налагодити комунікацію між членами даного кластеру.

Для розробки модулів розподіленої системи, був використаний фреймворк Asp.Net Core, що дозволив легко та просто побудувати систему на платформі .Net із вбудованими рішенням для Інтернет-додатків.

4.2 Архітектура програмного забезпечення

На рисунку 4.1 зображено схему структурну компонентів розробленого додатку.

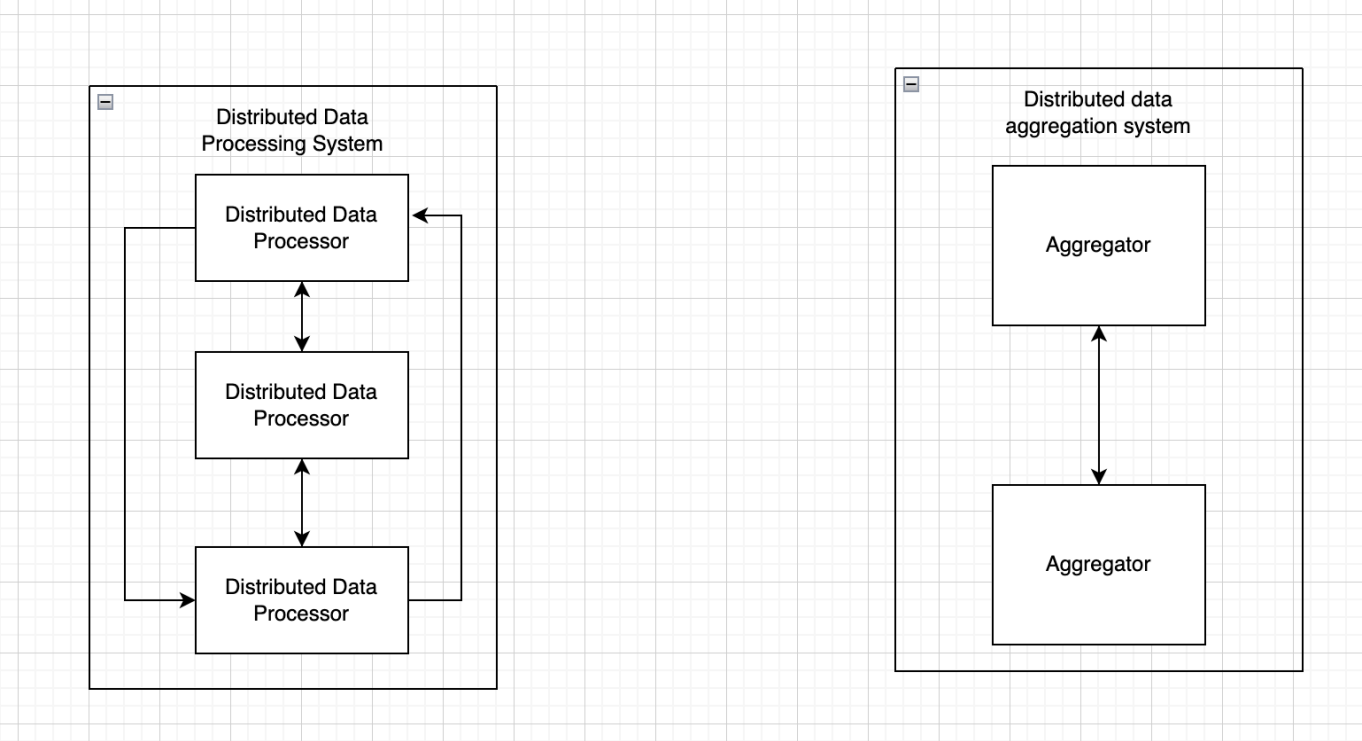


Рисунок 4.1 – Схема структурна компонентів розробленої системи для тестування

Розглянемо призначення кожної компоненти.

Компонент Distributed Data Processor кожен квант часу (цей параметр можна конфігурувати) пропонує випадкове число, та намагається додати його до розподіленого коміт логу. Зважаючи на те, що даний компонент є горизонтально маштабованим в 3 єкземпляри, виникає проблема неузгодженості і потрібно узгодити в правильному порядку запропоновані значення. Для цього була виконана інтеграція зі створеною бібліотекою, що дозволило об’єднати дані сервіси у кластер та вирішити дану проблему.

Компонент Aggregator виконує класичну задачу програмування – запуск певної роботи по таймеру. Для того, щоб гарантувати, що в один момент часу не буде виконуватися більше ніж одна задача, потрібно об’єднати сервіси в кластер та використати протокол консенсусу, щоб домовитися, який саме єкземпляр сервісу буде виконувати роботу.

На рисунку 4.2 зображено схему структурну компонентів розробленої бібліотеки.

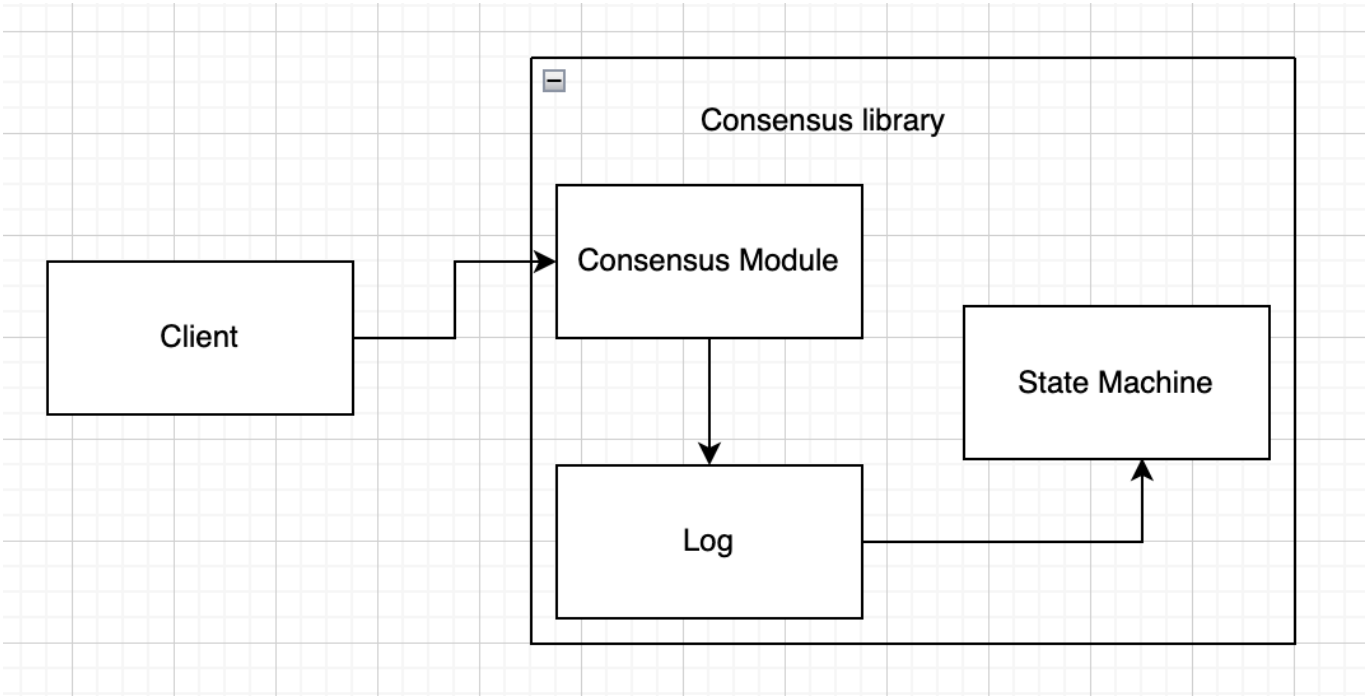


Рисунок 4.2 – Схема структурна компонентів розробленої системи для тестування

Компонент Consensus Module займаєтсья задачею оркестрування запитів клієнтів на додавання значення та обробкою пропозицій від сервісів.

Компонент Log відображає собою append-only структуру данних, яка зберігає всі запити на додавання даних, але які потенційно ще не є прийнятими кворумом чи 2х-фазним комітом.

Компонент State Machine являє собою стан системи та відображає дані, що були успішно зарепліковані та прийняті кворумом чи 2х-фазним комітом.

В таблиці 4.1 наведено опис класів бібліотеки, які було розроблено в рамках даної магістерської дисертації.

Таблиця 4.1 — Опис класів програмного забезпечення розробленої бібліотеки

Компонент	Клас	Опис
Library	AsyncManualResetEvent	Класс, що дозволяє синхронізувати мультипоточний додаток та надає можливість асинхронного інтерфейсу роботи.
	AsyncResponseHandler	Класс, що є генератором та гарантом унікальності повідомлень.
	ResponseCrate	Компонент, що виконує функції генерації повідомлення, також він використовує AsyncManualResetEvent для вирішення проблем, що можуть виникати при роботі в мультипоточному середовищі виконання.
	WaitHandleAsyncFactory	Компонент, що реалізовує шаблон проектування із GOF шаблонів – MethodFactory. Даний компонент дозволяє отримати об’єкт, що можна використовувати для синхронізації потоків.
	CandidateRequest	Контракт, що абстрагує запит кандидата, що наразі не є лідером та намагається стати повноцінним лідером кластеру.
	LeaderHeartbeat	Контракт, основною метою якого є надання можливісті надсилати запити і перевіряти чи працює лідер чи ні.
	StateLogEntryApplied	Контракт для відображення дати, коли пропозиція, була додана до журналу стану.
	StateLogEntrySimulateRequest	Використовується для визначення та скасування AppendLogEntryAsync нелідерним вузлом.

Продовження таблиці 4.1

	StateLogEntryRequest	Контракт, що надходить від послідовника до лідера під час синхронізації журналу стану.
	StatisticsCollectorSetup	Компонент інтерфейсу користувача, який відповідає за надання можливостей по налаштуванню параметрів збору необхідної статистичної інформації під час моделювання
	StateLogEntrySuggestion	Контракт, що відображає запит пропоузера на додавання нового значення до журналу стану.
	VoteOfCandidate	Класс, що відображає голос одного з кандидатів.
	AddLogEntryResult	Список можливих результатів та помилок, що можуть відбутися під час комунікації із лідером при запиті на додавання значення до лідеру.
	StateLog	Класс, що відображає зафіксоване в журналі стану значення. Призначений тільки для внутрішнього користування.
	StateLogEntry	Контракт, що відображає зафіксоване в реплікаційному лозі значення.
	IMessageReceiver	Контракт, що використовується для отримки повідомлень незалежно від каналу зв’язку.
	IMessageSender	Контракт, що використовується для відправки повідомлень незалежно від каналу зв’язку.
	NodeAddress	Клас, що відображає фізичну адресу члена кластера.
	ClusterEndpoint	Клас, що відображає точку, яка буде використовуватися для зв’язку та обміну даними та повідомленнями між даним членом кластеру.
	Msg	Базовий клас для повідомлення.

Кінець таблиці 4.1

	MsgHandshake	Клас, що відображає процес встановлення зв’язку перед початком обміну данними та повідомленнями при роботі з кожним вузлом системи.
	MessageReceiver	Клас, що реалізує всю логіку відправки та обміну данними. Саме цей клас знає про транспорт, що використовується для обміну поведомленнями.
	MessageSender	Клас, що реалізує всю логіку отримки та обміну данними. Саме цей клас знає про транспорт, що використовується для обміну поведомленнями.

В таблиці 4.2 наведено опис класів системи, яка була використана для тестування роботи бібліотеки, яка була розроблена в рамках даної магістерської дисертації.

Таблиця 4.2 — Опис класів програмного забезпечення розробленої системи для аналізу результатів роботи бібліотеки

Компонент	**Клас**	**Опис**
Aggregator	IAggregationService	Інтерфейс, що виступає абстракцією над сервісом для аггрегації даних.
	AggregationService	Клас, що емулює роботи агрегаційного сервісу та виконує роботу по сбору константних данних та логуванню інформації про збір цих даних.

Продовження таблиці 4.2

	IServiceConfiguration	Інтерфейс, що виступає абстракцією над конфігурацією сервіса та його правил роботи, в особливості правил запуску та збору данних.
	ServiceConfiguration	Компонент, що реалізує логіку конфігурації сервіса для агрегації данних та надає можливість для гнучкого задання налаштувань агрегаційної системи.
	IClusterSettings	Контракт, що виступає абстракцією над налаштування кластеру із декількох вузлів аггрегаційного сервісу та надає можливість гнучко налаштовувати принципи роботи кластеру та кожного із його вузлів.
	ClusterSettings	Клас, що реалізовує основні налаштування кластеру із декількох вузлів аггрегаційного сервісу та надає можливість гнучко налаштовувати принципи роботи кластеру та кожного із його вузлів.
Processor	ICurrentStateRepository	Клас, що виступає абстракцією над певним станом розподілених данних та надає можливість для оперування цими данними.
	CurrentStateRepository	Клас, що реалізує логіку оперування розподіленими данними завдяки консенсус протоколу.

Кінець таблиці 4.2

	IClusterConfiguration	Контракт, що виступає абстракцією над налаштування кластеру із декількох вузлів сервісу роботи із розподіленими данними та надає можливість гнучко налаштовувати принципи роботи кластеру та кожного із його вузлів.
	ClusterConfiguration	Клас, що реалізовує основні налаштування кластеру із декількох вузлів сервісу для роботи з розподіленими данними та надає можливість гнучко налаштовувати принципи роботи кластеру та кожного із його вузлів.

В таблиці 4.3 наведено специфікацію основних функцій, які було розроблено в рамках даної роботи.

Таблиця 4.3 — Специфікація функцій програмного забезпечення

Клас	**Метод**	**Призначення**	**Параметри**	**Семантика параметрів**	**Повертає результат**
AsyncManualResetEvent	WaitAsync()	Очікування отримки примітиву синхронізації.	Немає.	Параметрів немає.	Чи успішно отримано примітив синхронізації.
AsyncManualResetEvent	Set()	Встановити примітив в зачиненй стан, коли ніхто не може отримати примітив синхронізації.	Немає.	Параметрів немає.	Не повертає значення.

Кінець таблиці 4.3

AsyncManualResetEvent	Reset()	Встановити примітив в відчинений стан, коли будь-хто може отримати примітив синхронізації.	Немає.	Параметрів немає.	Не повертає значення.
IMessageReceiver	IncomingSinglaHandler()	Визначити процес обробки отриманих від інших вузлів сигналів.	NodeAddress nodeAddress, eSignalType signalType, byte[] data	Вузол, що надіслав повідомлення, тип повідомлення, повідомлення.	Не повертає значення.
IMessageSender	SendToAll()	Відправка повідомлення всім вузлам системи.	eRaftSignalType signalType, byte[] data, NodeAddress senderNodeAddress, string entityName, bool highPriority	Тип повідомлення, повідомлення, адреса відправника, ім’я логічної групи, вказівник на пріорітет.	Не повертає релуьтат.
IMessageSender	SendTo()	Відправка повідомлення всім вузлам системи.	NodeAddress nodeAddress,eRaftSignalType signalType, byte[] data, NodeAddress senderNodeAddress, string entityName	Адреса отримувача, тип повідомлення, повідомлення, адреса відправника, ім’я логічної групи.	Не повертає релуьтат.

4.3 Інструкція користувача

Розглянемо розроблену бібліотеку.

Приклад базової конфігурації вузлів кластеру зображено на рисунку 4.3.

Рисунок 4.3 – Конфігурація вузлів кластеру

Приклад базової конфігурації роботи алгоритму для досягнення консенсусу зображено на рисунку 4.3.

```
"EntitiesSettings":
[
        {
                "EntityName":"default",
                "DelayedPersistenceMs":10,
                "DelayedPersistenceIsActive":false,
                "InMemoryEntity":false,
                "InMemoryEntityStartSyncFromLatestEntity":false,
                "VerboseProtocol":true,
                "VerboseTransport":true
        }
],
```

Рисунок 4.4 – Конфігурація роботи алгоритму

Для того щоб створити кластер, необхідно вказати хоча б 3 адреси вузлів данної системи. Це необхідно для виділення кворуму та корректної роботи алгоритму. Як можна бачити на рисунку 4.2, необхідно вказати адресу кожного вузла системи, які буде використовувати алгоритм для роботи з сервісами та досягнення консенсусу в системі.

Також, як показано на рисунку 4.3, можна налагоджувати та гнучко конфігурувати процес роботи самого алгоритму. Наприлад, задати рівень логів чи затримки збереження коміт логу даного вузла системи.

Розглянемо роботу алгоритму на прикладі Data Processing сервісу, рисунок 4.4.

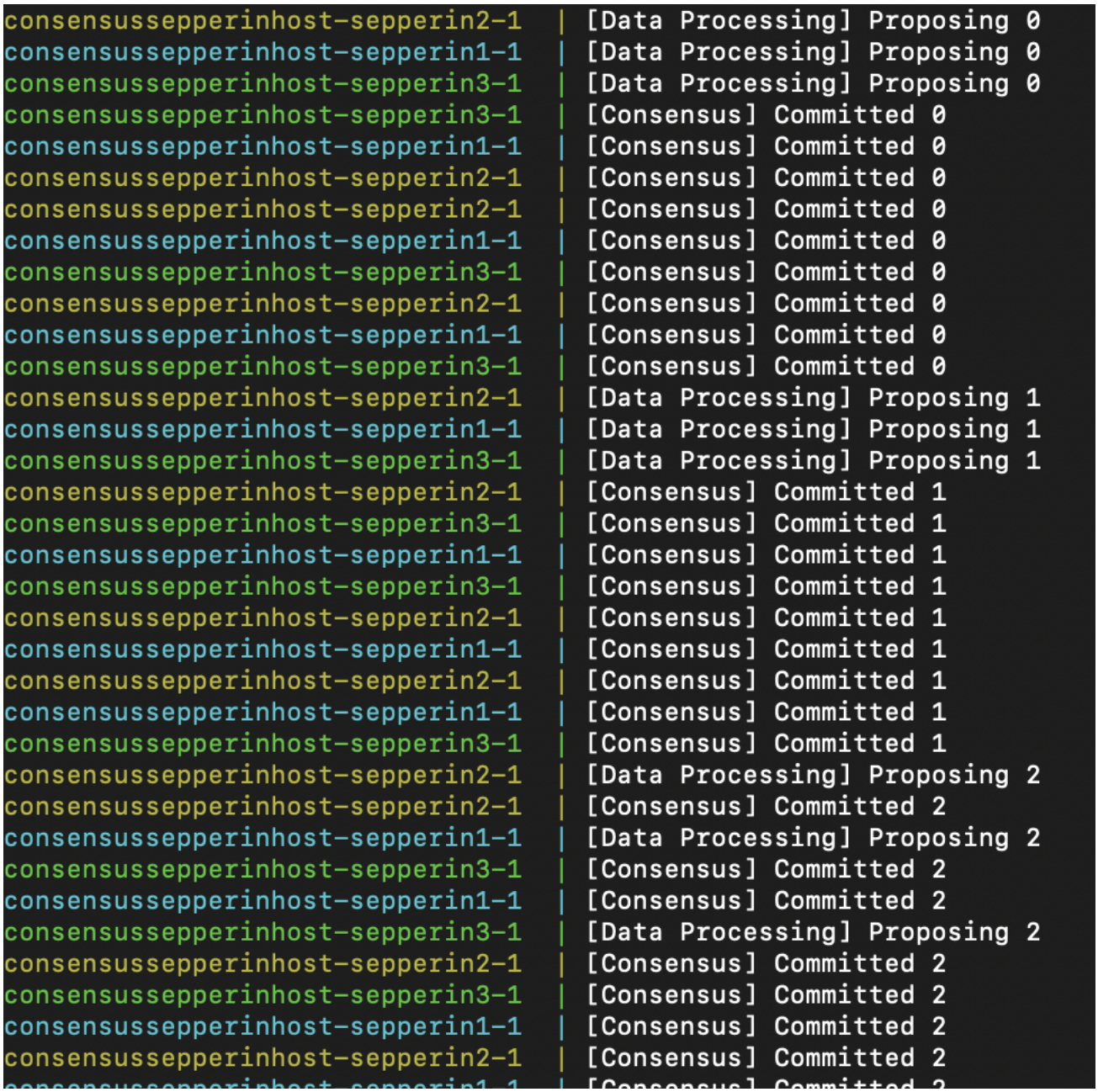


Рисунок 4.5 – Приклад роботи Data Processing сервісу

На рисунку 4.5 зображений приклад роботи Data Processing системи, що горизонтально замаштабована в 3 екземпляри.

4.4 Висновок до розділу

Для проведення процесу моделювання та тестуваня роботи алгоритму досягнення консенсусу в розподілених системах було розроблено спеціальне

програмне забезпечення з використанням мови програмування C#. Розроблене програмне забезпечення дозволяє зручно налаштовувати параметри алгоритму, легко задавати та конфігурувати учасників кластеру.

5 РОЗРОБКА СТАРТАП-ПРОЕКТУ

Розроблений стартап-проект на тему “Вирішення проблеми консенсусу в розподілених системах на базі платформи .Net”.

Проект дозволяє розробникам та компаніям пришвидчити процес написання та розробки програмних продуктів, а також значно підвищити явість та високий рівень надійності програми.

5.1 Аналіз ринкових можливостей запуску стартап-проекту

Проведемо аналіз конкуренції за Портером. Результати наведено в таблиці 5.1.

Таблиця 5.1 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	**Прямі конку-ренти в галузі**	**Потенційні конкурен-ти**	**Постачаль-ники**	**Клієнти**	**Товари-замінники**
	Raft.Net	Головним бар’єром є ефективна реалізація можливостей навіть базового протоколу, оскільки ще не завершена стадія розробки продукту конкурента	Відсутні	Підтримка з боку розробників, відкритий код, можливість підтримувати та розвивати продукт завдяки широкому ком’юніті	Немає

Кінець таблиці 5.1

Висновки:	Наразі існує лише один прокту-конкурент, але він не надає послуг в повній мірі, так як знаходится на стадії розробки та не є популярним на даний момент.	В даний момент поява нових конкурен-тів малоімовірна.	Постачальників немає.	Якщо користувачі не будуть отримувати достатніх переваг від користування продуктом, вони можуть перейти до конкурентів.	Наразі існує дуже мало товарів замінників, тому конкуренція невисока.

Проаналізувавши таблицю 5.1 можемо зробити висновок, що проект має можливість отримання популярності на ринку. Для забезпечення конкурентноспроможності потрібна наявність наступних характеристик:

а) реалізація та вирішення проблеми консенсусу в розподілених системах, а саме проблеми візантійських генералів та проблеми виходу вузла з ладу;

б) мати досвічену команду;

в) постійна адаптація під потреби ринку;

г) постійний зворотній зв’язок.

Виконаємо більш детальний аналіз сформульованих умов конкурентноспроможності. Результати аналізу наведено в таблицях 5.2 і 5.3.

Таблиця 5.2 — Обґрунтування факторів конкурентоспроможності

№ п/п	**Фактор конкурентоспроможності**	**Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)**
1	Наявність якісної реалізації	Наявність якісної реалізації протоколу досягнення консенсусу в децентралізованій мережі є необхідною умовою її популярності серед користувачів у порівнянні з конкурентами.
2	Мати досвічену команду	Великий досвід команди у роботі із передовими технологіями дозволить швидко реалізувати продукт та швидко адаптуватися під зміну потреб клієнтів.
ы	Постійна адаптація під потреби ринку	Необхідно постійно бути в курсі тенденцій на ринку розподілених систем та відповідно до вимог оновлювати проект.

Кінець таблиці 5.2

4	Постійний зворотній зв’язок	Оскільки продукт новий, необхідно постійно підтримувати зворотній зв’язок із користувачами та надавати їм послуги консультації. Також необхідно постійно збільшувати ком’юніті та покращувати документацію продукту.

Таблиця 5.3 — Порівняльний аналіз сильних та слабких сторін

№ п/п	**Фактор конкурентоспроможності**	**Бали 1-20**	**Рейтинг товарів-конкурентів у порівнянні з Sepperin**						
			–3	–2	–1	0	+1	+2	+3
1	Наявність якісної реалізації консенсус протоколу	20				+			
2	Мати досвічену команду	15					+		
3	Постійна адаптація під потреби ринку	20			+				
4	Постійний зворотній зв’язок	20	+						

Проведемо огляд висновків по ринку в таблиці 5.4, яка містить результати SWOT аналізу.

Таблиця 5.4 — SWOT-аналіз стартап-проекту

Сильні сторони: Підтримка, Якість реалізації та тестування, Постійне розширення можливостей	Слабкі сторони: Невелике ком’юніті Відсутність клієнтської бази

Кінець таблиці 5.4

Можливості: Самовідновлення під час падінь, Можливість корегувати швидкість роботи та констистентність данних	Загрози: Неправильна конфігурація, що негативно вплине на роботу системи

Проведення SWOT-аналізу дозволило виявити основні сильні і слабкі сторони запопронованого стартапу. Основними сильними сторонами є даного проекту є якість реалізації та тестування, підтримка розробників, а також постійне розширення та розвиток рішення. В той же час наявні і деякі слабкі сторони, до яких відноситься невелике ком’юніті, а також відсутність клієнтської бази. Також, ризиками для реалізації даного проекту є можливий відтік клієнтів до конкурентів, що може бути спричинено невірним використання та помилками під час конфігурації продукту, а також поява на ринку нових конкурентів.

5.2 Розроблення ринкової стратегії проекту

Аналіз цільових груп потенційних клієнтів, їх готовність до споживання продукту, рівень конкуренції та попиту в межах цільових груп та простота входження представлені у таблиці 5.5.

Таблиця 5.5 — Вибір цільових груп потенційних споживачів

№ п/п	**Опис профілю цільової групи потенційних клієнтів**	**Готовність споживачів сприйняти продукт**	**Орієнтовний попит в межах цільової групи (сегменту)**	**Інтенсивність конкуренції в сегменті**	**Простота входу у сегмент**
1	Великі компанії та корпорації	Середня	Середній	Висока	Сегмент є досить заповнений гтовими рішенями та внутрішніми продуктами, на які він орієнтуєнтується, тому вхід у сегмент не є простим
2	Стартап-проекти	Висока	Середній	Середня	Вхід у сегмент простий, тому що на початкових етапах розробники для підвищення темпу роботи, будуть використовувати багато готових рішень
3	Фрілансери	Висока	Високий	Невисока	Продукт дозволить менш кваліфікованим спеціалістам будувати складні системи та бути впевненими в консистентності та їх безпеці
Цільові групи: Великі компанії та корпорації, стартап-проекти та фрілансери					

З описів профілів цільових груп робимо висновок, що найбільш простим є вхід у сегменти стартап-проектів та фрілансерів. Це дасть можливість швидко наростити клієнтську базу та потім розширитися і на інші сегменти.

У таблицях 5.6, 5.7 та 5.8 розглянуті базові стратегії розвитку.

Для вирішення проблеми маркетингу, найбільше підійде стратегія диференційованого маркетингу. Розглянемо кожен із варіантів розвитку проекту та на основі цього розробляємо окремі програми впливу на ринку.

Будемо активно наглядати за продуктами конкурентів та відповідним чином оновлювати наш продукт, як в технічному так і продуктовому аспектах.

Таблиця 5.6 — Визначення базової стратегії розвитку

№	**Обрана альтернатива розвитку проекту**	**Стратегія охоплення ринку**	**Ключові конкуренто-спроможні позиції відповідно до обраної альтернативи**	**Базова стратегія розвитку**
1	Наш проект орієнтуєнтується на стартап-проекти та фріланс розробників, тому для кожного з цих сегментів потрібно вжити окремі заходи для підвищення привабливості проекту для них	Дифенеційований маркетинг	Наявність хорошої документації, можливість надати подомогу та підтримку в вирішенні поставленої проблеми, можливість консультації	Стратегія диференціації

Таблиця 5.7 — Визначення базової стратегії конкурентної поведінки

№	**Чи є проект «першопроходьцем» на ринку?**	**Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?**	**Чи буде компанія копіювати основні характеристики товару конкурента, і які?**	**Стратегія конку-рентної поведінки**
1	Ні	Компанія буде активно наглядати за продуктами конкурентів та відповідним чином оновлювати наш продукт, як в технічному так і продуктовому аспектах.	Компанія буде копіювати лише успішні та дійсно цікаві характеристики конкурентів.	Стратегія іновацій

Таблиця 5.8 — Визначення стратегії позиціонування

№	**Вимоги до товару цільової аудиторії**	**Базова стратегія розвитку**	**Ключові конкурентоспроможні позиції власного стартап-проекту**	**Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)**
1	Якісна реалізація вирішення проблеми консенсусу із підтримкою та легкою інтеграцією в існуючі рішення	Стратегія диференціації	наявність якісної реалізації алгоритму; постійна адаптація під потреби ринку; постійна підтримка та розвиток продукту	Вирішення проблеми консенсусу, Легка інтеграція в додатки на базі останніх версій платформи .Net

5.3 Розроблення маркетингової програми стартап-проекту

Проаналізуємо потреби потенційних груп користувачів. Результати аналізу наведені в таблиці 5.9.

Таблиця 5.9 — Визначення ключових переваг концепції потенційного товару

№	**Потреба**	**Вигода, яку пропонує товар**	**Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)**
1	Потреба в вирішенні проблеми інформаційних систем	Вирішення данної проблеми за максимально короткий проміжок часу та з мінімальними зусиллями	Якійсна та перевірена реалізація, постійна підтримка та консультація, гарна документація
2	Потреба в підвищеному рівні консистентності	Проект дозволить створювати захищені та безпечні з точки зору данних додатки	Полегшення написання високонадійних систем

Наприкінці проведемо аналіз концепції маркентингових комунікацій. Результати зведені у таблицю 5.10.

Таблиця 5.10 — Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комуніка-цій, якими Користую-ться цільові клієнти	Ключові позиції, обрані для позиціону-вання	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Вирішення проблем власних проектів завдяки використанню даного рішення	StackOverflow, GitHub	Якісний та простий протокол вирішення проблем консенсусу та гарантування високого рівня надійності системи	Підвищити рівень популярності	Спробуйте створити горизонтально маштабуємі додатки із високим рівнем консинтентності

5.4 Висновок до розділу

Було продовжено аналіз стартап-проекту децентралізованої соціальної мережі. Було виділено основні цільові групи користувачів мережі, що дозволило розробити відповідну маркетингову стратегію, що спирається на потреби цих користувачів.

ВИСНОВКИ

У результаті роботи над магістерською дисертацією, була проведена аналітична робота існуючих проблем розподілених систем. Були розглянуті часткові рішення та методи для подолання даних проблем

Також було проведено дослідження існуючих методів та алгоритмів досягнення консенсусу в розподілених системах, що є актуальний на даний момент. Описано та виділено особливості та аналіз різних модифікацій базового методу досягнення консенсусу, що націлені на подолання спецефічних проблем.

В роботі було розроблено власний метод для подолання проблеми консенсусу в розподілених системах, що вирішує як проблему виходу з ладу одного чи більше вузлів системи так і проблему візантійських генералів. Даний метод був описаний в роботі та було виконане його порівняння із існуючими альтернативами.

На базі розробленого методу був побудований алгоритм, що спростив процес реалізації та ліг в основу розробленої бібліотеки для платформи .Net.

Була розроблена бібліотека, що дає можливість вирішити основні проблеми досягнення консенсусу в розподілених системах, а саме, проблему виходу вузла з ладу, проблему візантійських генералів та проблему вибору одного значення із ряду запропонованих.

Був проведений аналіз та порівняння розробленого методу із існучими та виділені сильні та слабкі сторони данного підходу.

Також було розроблено спеціальне програмне забезпечення, що імітувало різні варіанти проблеми неузгодженості в децентралізованій системі. Для подолання даних проблем була використана розроблена бібліотека та проведено аналіз роботи алгоритму.

За матеріалами дисертації було опублікована 1 наукова робота: 1 стаття[48].

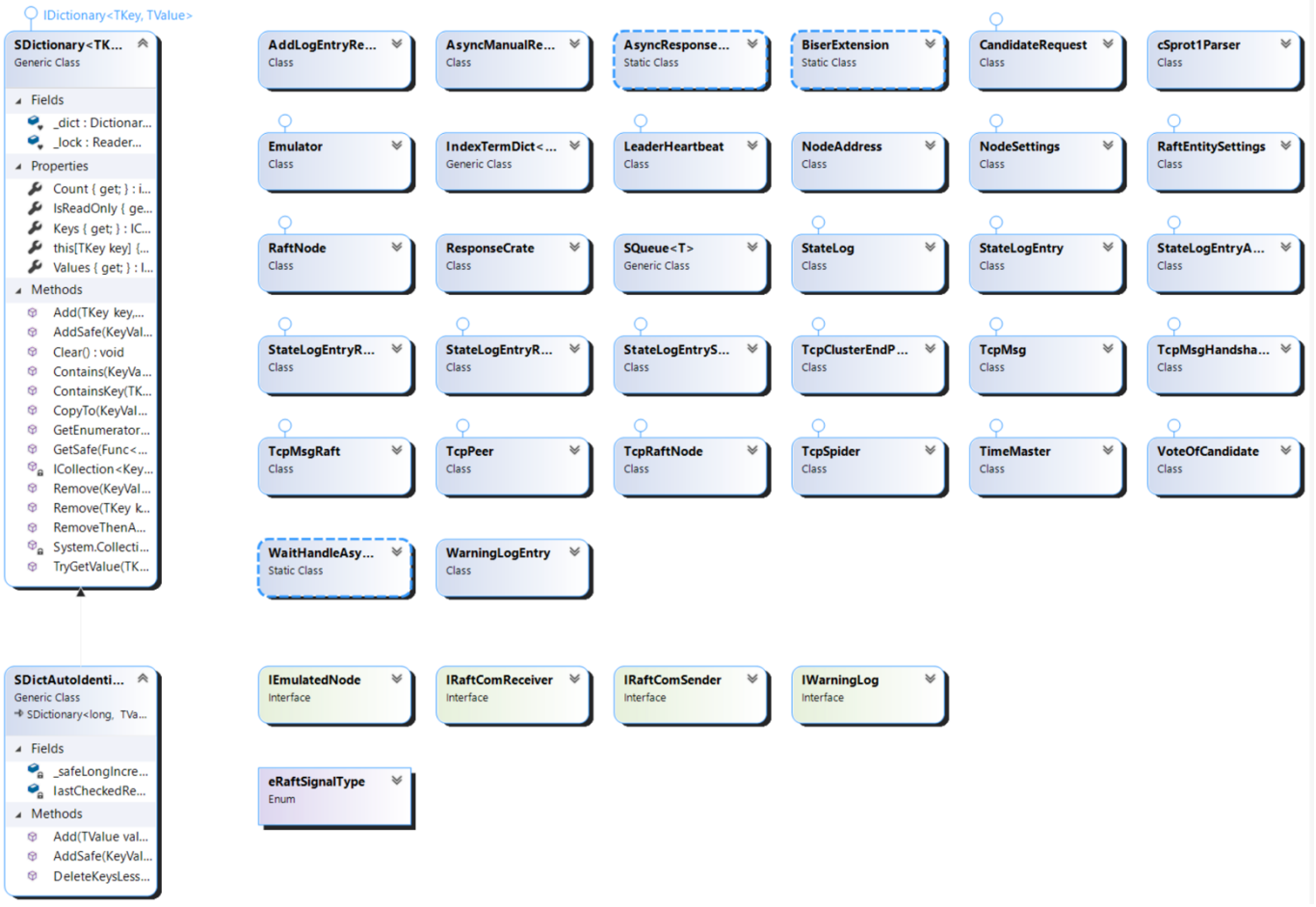
6 ПЕРЕЛІК ПОСИЛАНЬ

1) Failure Rates in Google Data Centers [Електронний ресурс] // Режим доступу: https://www.datacenterknowledge.com/archives/2008/05/30/failure-rates-in-google-data-centers
2) The Byzantine Generals Problem [Електронний ресурс] // Режим доступу: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fum%2Fpeople%2Flamport%2Fpubs%2Fbyz.pdf
3) The Byzantine Generals Problem [Електронний ресурс] // Режим доступу: https://www.microsoft.com/en-us/research/uploads/prod/2016/12/The-Byzantine-Generals-Problem.pdf
4) Mathematical Models of Object-Based Distributed Systems [Електронний ресурс] // Режим доступу: https://www.chcduarte.com/distmath.pdf
5) Paxos vs. Raft: Have we reached consensus or distributed consensus? [Електронний ресурс] // Режим доступу: https://www.mendeley.com/catalogue/9cdbf9c9-c2d8-3f95-85e0-eaef2eb4fb1b/
6) Distributed.net project [Електронний ресурс] // Режим доступу: http://www.distributed.net/Main_Page
7) Альфред В. А. Структуры данных и алгоритмы : научно-популярная книга / Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман – 1-е изд., перераб. и доп. – М.: Вильямс, 2016.– 620 с.
8) Розроблення стартап-проєкту [Електронний ресурс]: Методичні рекомендації до виконання розділу магістерських дисертацій для студентів інженерних спеціальностей / За заг. ред. О.А. Гавриша. – Київ : НТУУ «КПІ», 2016. – 28 с.

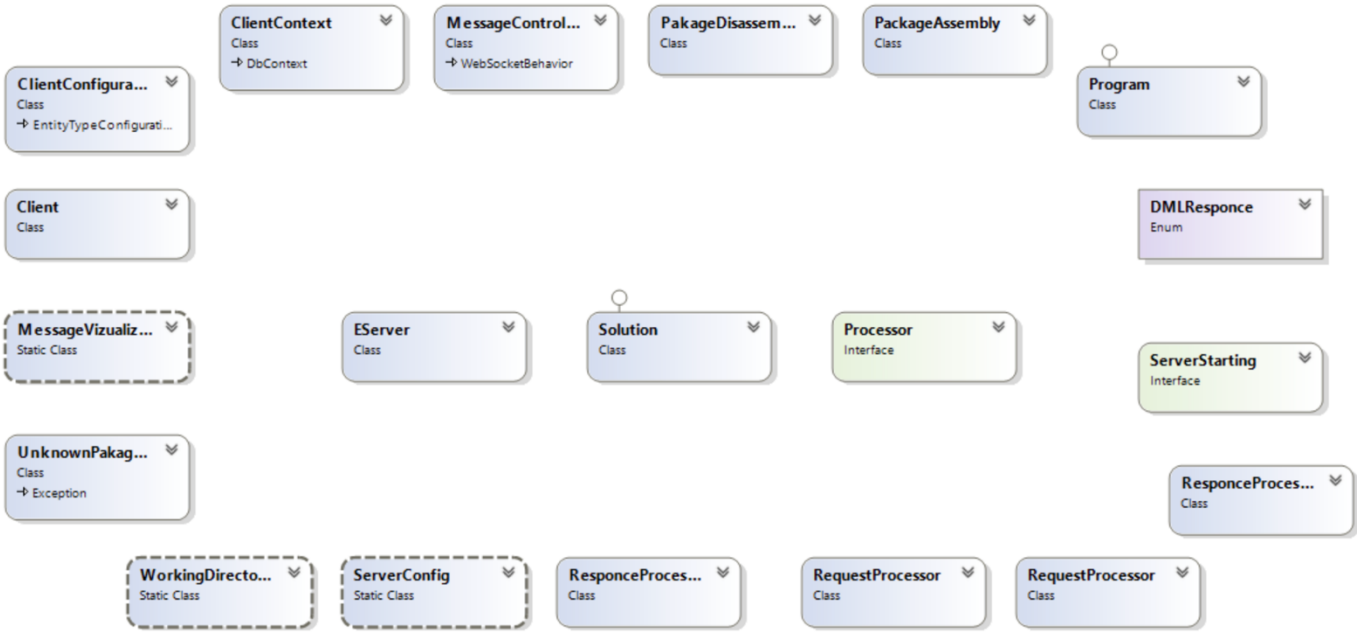
ДОДАТОК А

Графічний матеріал

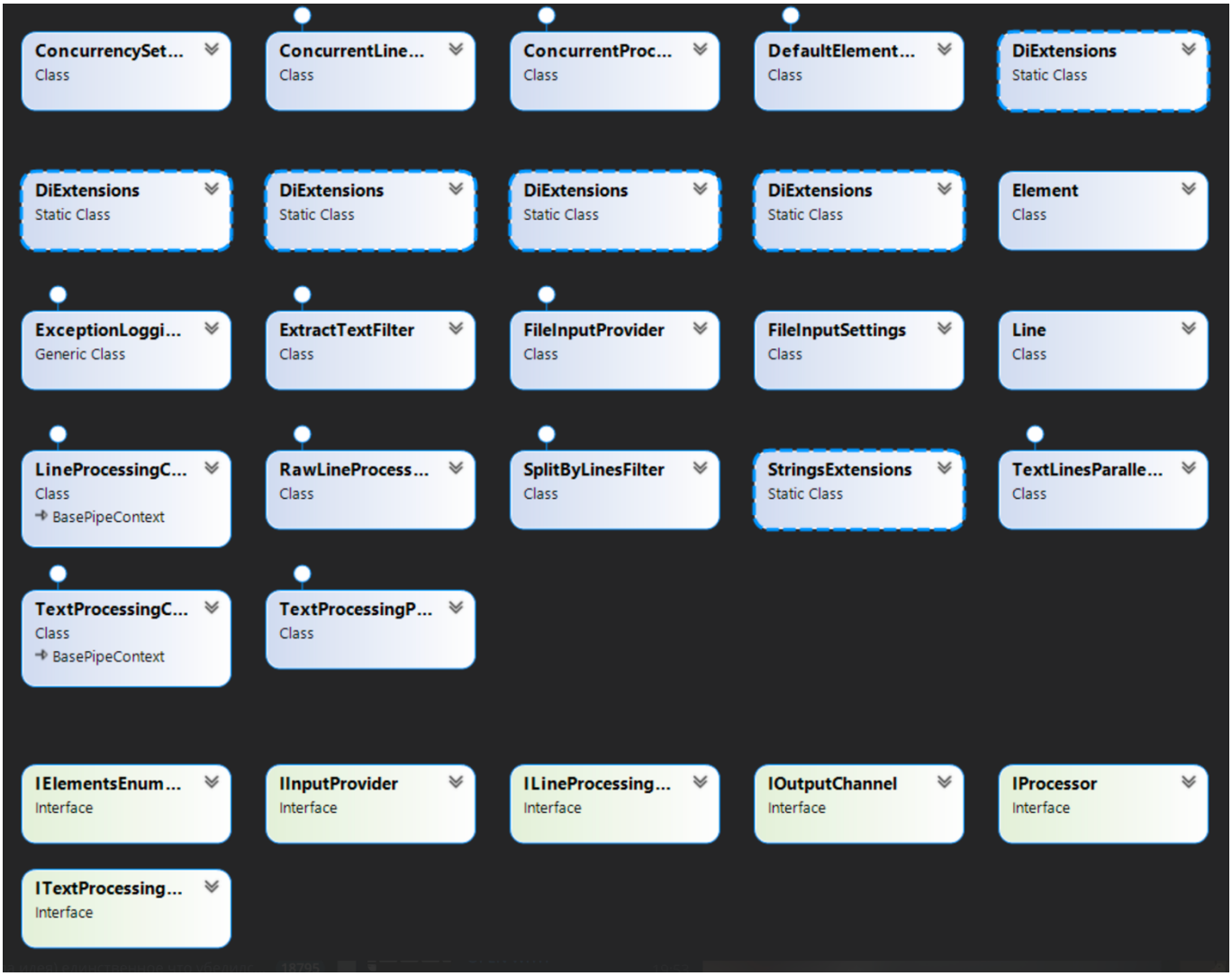
Плакат 1 - Діаграма класів бібліотеки



Плакат 2 - Діаграма класів сервісу Aggregator



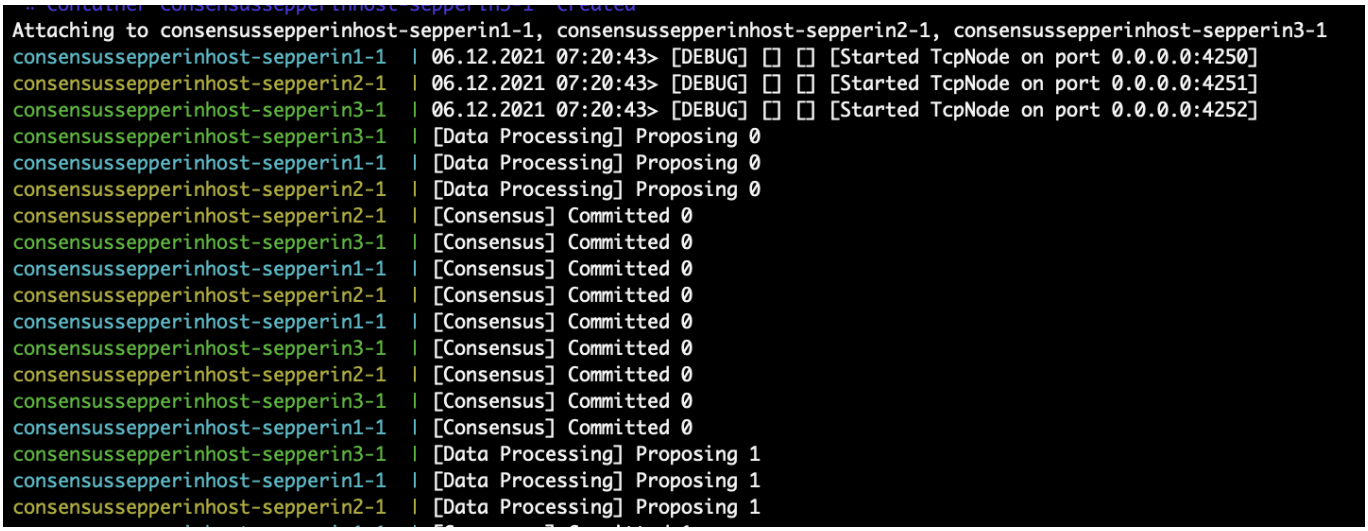
Плакат 3 - Діаграма класів сервісу Data Processor



Плакат 4 – Екранна форма прикладу конфігурації кластеру із 3х вузлів

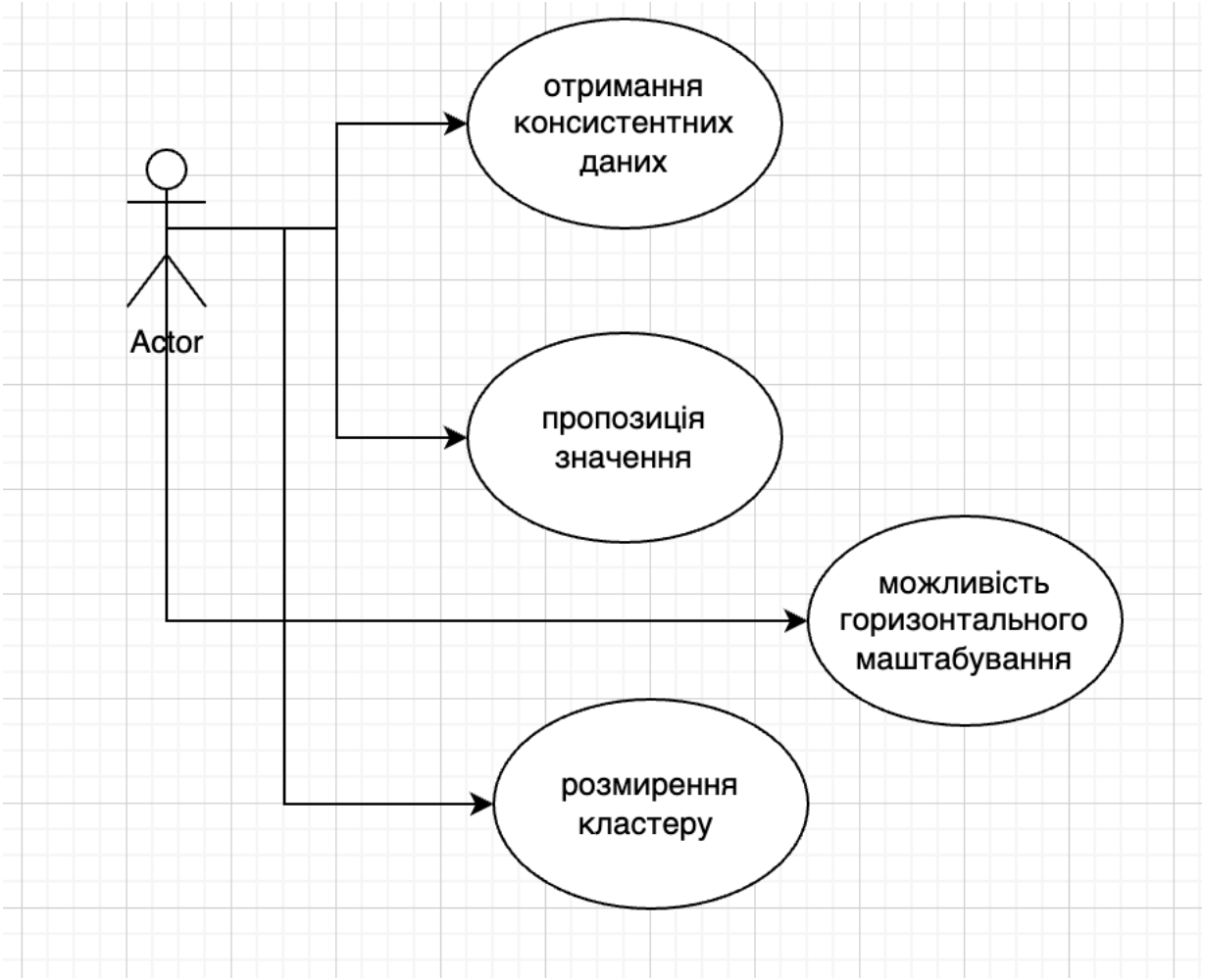
```
{
        "EntitiesSettings":
        [
                {
                        "EntityName":"default",
                        "DelayedPersistenceMs":10000,
                        "DelayedPersistenceIsActive":true,
                        "InMemoryEntity":false,
                        "InMemoryEntityStartSyncFromLatestEntity":false,
                        "Verbose":false,
                        "VerboseTransport":false
                }
        ],
        "ClusterEndPoints":
        [
                {
                        "Host":"127.0.0.1",
                        "Port":4250
                },
                {
                        "Host":"127.0.0.1",
                        "Port":4251
                },
                {
                        "Host":"127.0.0.1",
                        "Port":4252
                },
                {
                        "Host":"127.0.0.1",
                        "Port":4253
                },
                {
                        "Host":"127.0.0.1",
                        "Port":4254
                }
        ]
}
```

Плакат 5 – Екранна форма прикладу роботи кластеру із 3х вузлів



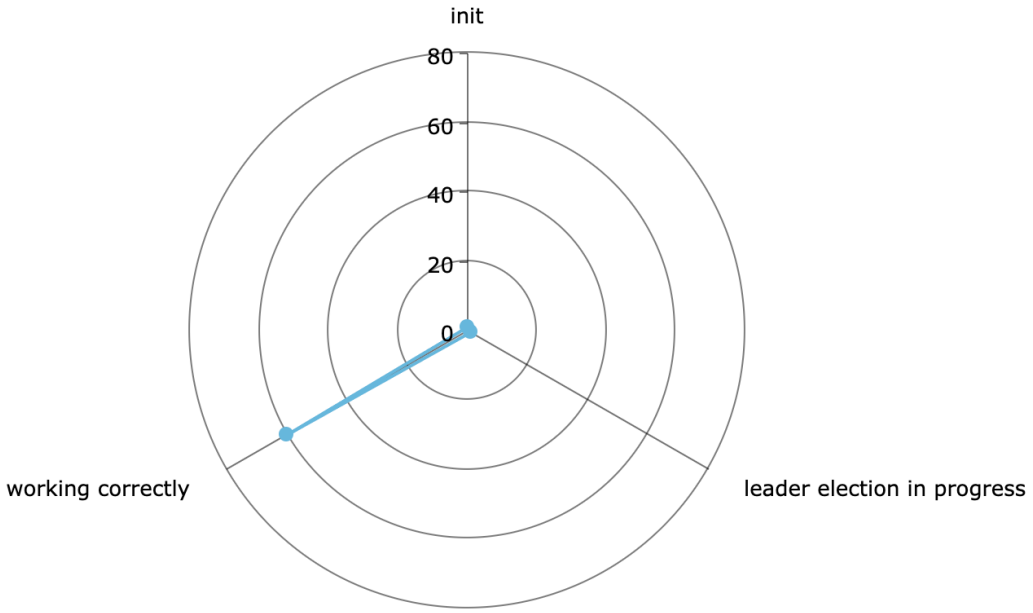
ДОДАТОК Б

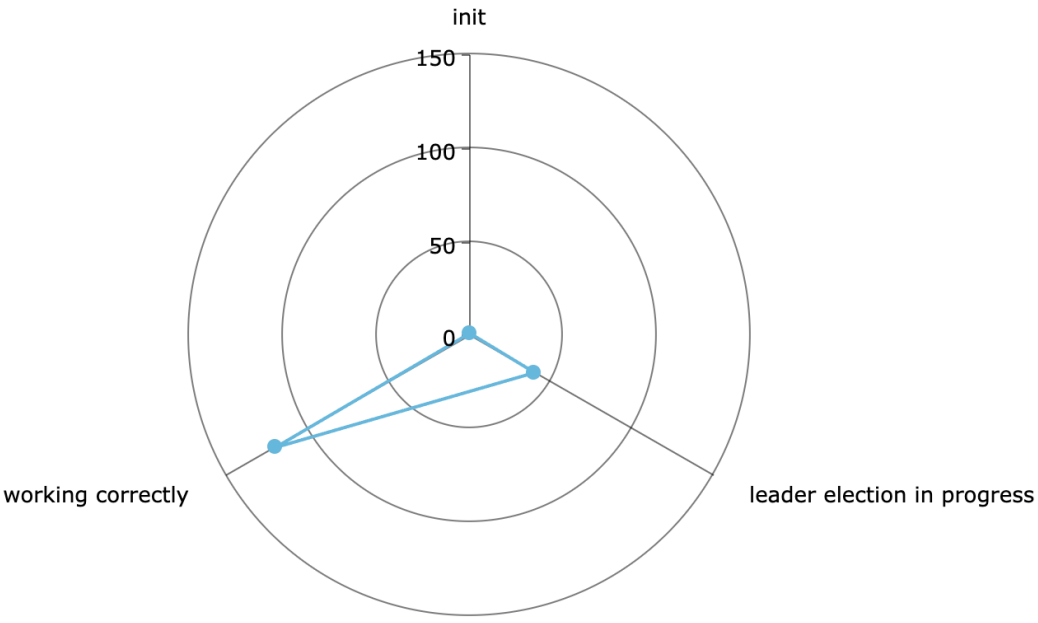
СХЕМА СТУКТУРНА ВАРІАНТІВ ВИКОРИСТАННЯ



ДОДАТОК В

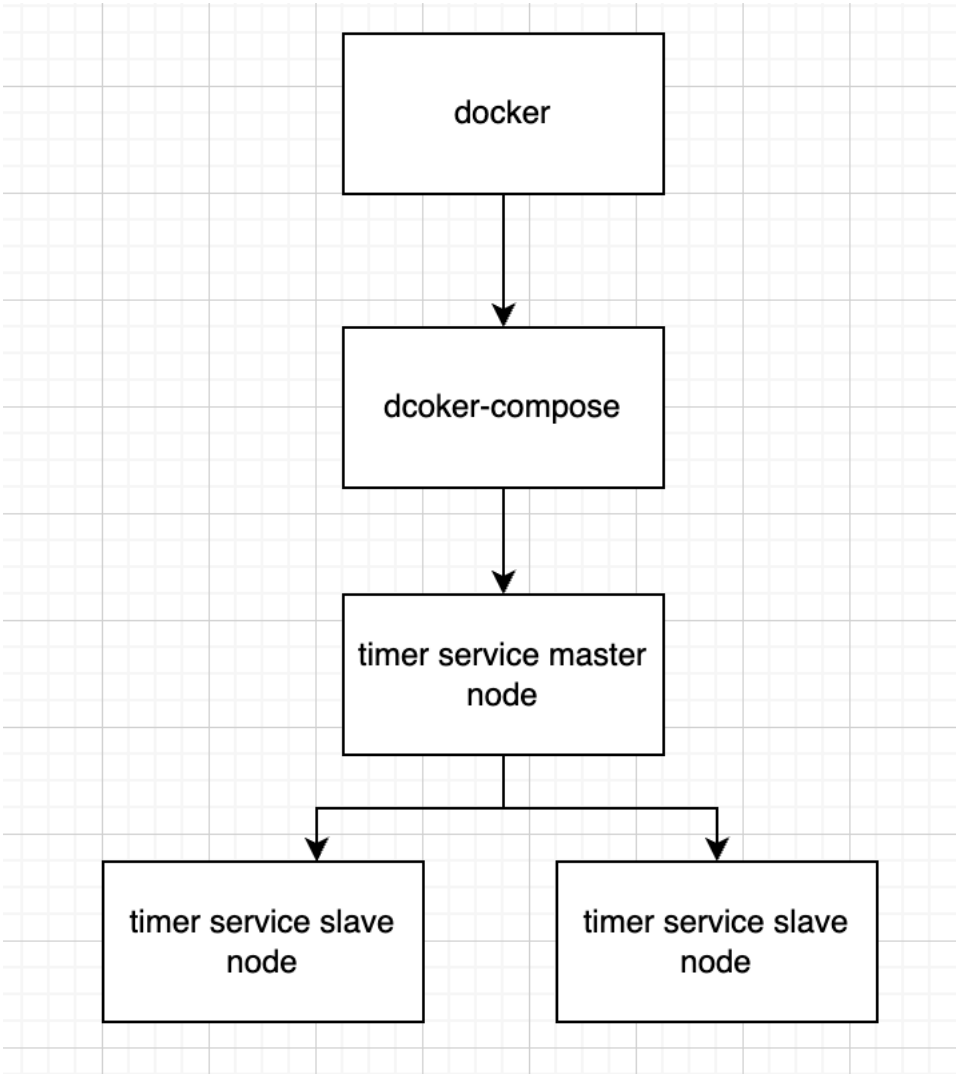
РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТІВ





ДОДАТОК Г

СХЕМА СТРУКТУРНА РОЗГОРТАННЯ КОМПОНЕНТІВ СИСТЕМИ



ДОДАТОК Д

ЛІСТИНГ ПРОГРАМИ

```
/*
 In this class all credits to https://github.com/StephenCleary/AsyncEx
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Raft
{
   internal class AsyncManualResetEvent
   {
      // can be used WaitHandleAsyncFactory.cs (From WaitHandle)

      private volatile TaskCompletionSource<bool> _tcs = new
TaskCompletionSource<bool>();
      private readonly object _mutex;

      public AsyncManualResetEvent()
      {
         _mutex = new object();
      }

      public Task<bool> WaitAsync()
      {
         lock (_mutex)
```

```
            {
                return _tcs.Task;
            }
        }


        public void Set()
        {
            lock (_mutex)
            {
                _tcs.TrySetResult(true);
            }
        }

        public void Reset()
        {
            lock (_mutex)
            {
                if (_tcs.Task.IsCompleted)
                    _tcs = new TaskCompletionSource<bool>();
            }
        }

    }
}
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using DBreeze.Utils;

namespace Raft
{
   internal static class AsyncResponseHandler
   {
      static object Sync = new object();
      static ulong MessageIdCnt = 0;

      public static ConcurrentDictionary<string, ResponseCrate> df = new ConcurrentDictionary<string, ResponseCrate>();

      public static byte[] GetMessageId()
      {
         lock (Sync)
         {
            MessageIdCnt++;
            return DateTime.UtcNow.To_8_bytes_array().Concat(MessageIdCnt.To_8_bytes_array_BigEndian());
         }
      }

      public static void ResponseCrateCleanUp(object userToken)
      {
         try
         {
            DateTime now = DateTime.UtcNow;
```

```
                foreach (var el in df.Where(r => now.Subtract(r.Value.created).TotalMilliseconds >= r.Value.TimeoutsMs).ToList())
                {
                    el.Value.IsRespOk = false;
                    el.Value.Set_MRE();
                }

            }
            catch (Exception ex)
            {

            }
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;

namespace Raft
{
    internal class ResponseCrate
    {
        /// <summary>
        /// Not SLIM version must be used (it works faster for longer delay which RPCs are)
        /// </summary>
```

```
public ManualResetEvent mre = null;
public byte[] res = null;
public Action<Tuple<bool, byte[]>> callBack = null;
public bool IsRespOk = false;

public AsyncManualResetEvent amre = null;

public DateTime created = DateTime.UtcNow;
public int TimeoutsMs = 30000;

public void Init_MRE()
{
   mre = new ManualResetEvent(false);
}

/// <summary>
/// Works faster with timer than WaitOneAsync
/// </summary>
public void Init_AMRE()
{
   amre = new AsyncManualResetEvent();
}

public void Set_MRE()
{

   if (mre != null)
   {
      mre.Set();
   }
```

```
            else if (amre != null)
            {
                amre.Set();
            }

        }

        public bool WaitOne_MRE(int timeouts)
        {
            //if (Interlocked.Read(ref IsDisposed) == 1 || mre == null)  //No sense
            //    return false;
            return mre.WaitOne(timeouts);
        }

        /// <summary>
        /// Works slower than amre (AsyncManualResetEvent) with the timer
        /// </summary>
        /// <param name="timeouts"></param>
        /// <returns></returns>
        async public Task WaitOneAsync(int timeouts)
        {

            //await resp.amre.WaitAsync();
            await                    WaitHandleAsyncFactory.FromWaitHandle(mre,
TimeSpan.FromMilliseconds(timeouts));
            //await mre.AsTask(TimeSpan.FromMilliseconds(timeouts));
        }

        //async public Task<bool> WaitOneAsync()
        //{
```

```
            //    //if (Interlocked.Read(ref IsDisposed) == 1 || amre == null)
            //    //    return false;

            //    return await amre.WaitAsync();

            //}


            long IsDisposed = 0;
            public void Dispose_MRE()
            {
                if (System.Threading.Interlocked.CompareExchange(ref IsDisposed, 1, 0) !=
0)
                    return;

                if (mre != null)
                {
                    mre.Set();
                    mre.Dispose();
                    mre = null;
                }
                else if (amre != null)
                {
                    amre.Set();
                    amre = null;
                }
            }

        }
    }
```

```
/*
 In this class all credits to https://github.com/StephenCleary/AsyncEx
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace Raft
{
    /// <summary>
    /// Provides interop utilities for <see cref="WaitHandle"/> types.
    /// </summary>
    internal static class WaitHandleAsyncFactory
    {
        /// <summary>
        /// Wraps a <see cref="WaitHandle"/> with a <see cref="Task"/>. When the <see cref="WaitHandle"/> is signalled, the returned <see cref="Task"/> is completed. If the handle is already signalled, this method acts synchronously.
        /// </summary>
        /// <param name="handle">The <see cref="WaitHandle"/> to observe.</param>
        public static Task FromWaitHandle(WaitHandle handle)
        {
            return FromWaitHandle(handle, Timeout.InfiniteTimeSpan, CancellationToken.None);
        }

        /// <summary>
```

/// Wraps a <see cref="WaitHandle"/> with a <see cref="Task{Boolean}"/>. If the <see cref="WaitHandle"/> is signalled, the returned task is completed with a <c>true</c> result. If the observation times out, the returned task is completed with a <c>false</c> result. If the handle is already signalled or the timeout is zero, this method acts synchronously.

/// </summary>

/// <param name="handle">The <see cref="WaitHandle"/> to observe.</param>

/// <param name="timeout">The timeout after which the <see cref="WaitHandle"/> is no longer observed.</param>

public static Task<bool> FromWaitHandle(WaitHandle handle, TimeSpan timeout)

{

return FromWaitHandle(handle, timeout, CancellationToken.None);

}

/// <summary>

/// Wraps a <see cref="WaitHandle"/> with a <see cref="Task{Boolean}"/>. If the <see cref="WaitHandle"/> is signalled, the returned task is (successfully) completed. If the observation is cancelled, the returned task is cancelled. If the handle is already signalled or the cancellation token is already cancelled, this method acts synchronously.

/// </summary>

/// <param name="handle">The <see cref="WaitHandle"/> to observe.</param>

/// <param name="token">The cancellation token that cancels observing the <see cref="WaitHandle"/>.</param>

public static Task FromWaitHandle(WaitHandle handle, CancellationToken token)

{

return FromWaitHandle(handle, Timeout.InfiniteTimeSpan, token);

}

```
public static Task<bool> FromWaitHandle(WaitHandle handle, TimeSpan timeout, CancellationToken token)
{
   // Handle synchronous cases.
   var alreadySignalled = handle.WaitOne(0);
   if (alreadySignalled)
      return TaskConstants.BooleanTrue;
   if (timeout == TimeSpan.Zero)
      return TaskConstants.BooleanFalse;
   if (token.IsCancellationRequested)
      return TaskConstants<bool>.Canceled;

   // Register all asynchronous cases.
   return DoFromWaitHandle(handle, timeout, token);
}

private static async Task<bool> DoFromWaitHandle(WaitHandle handle, TimeSpan timeout, CancellationToken token)
{
   var tcs = new TaskCompletionSource<bool>();
   using (new ThreadPoolRegistration(handle, timeout, tcs))
   using (token.Register(state => ((TaskCompletionSource<bool>)state).TrySetCanceled(), tcs, useSynchronizationContext: false))
      return await tcs.Task.ConfigureAwait(false);
}

/// <summary>
/// Provides completed task constants.
/// </summary>
```

```
public static class TaskConstants
{
    private static readonly Task<bool> booleanTrue = Task.FromResult(true);
    private static readonly Task<int> intNegativeOne = Task.FromResult(-1);

    /// <summary>
    /// A task that has been completed with the value <c>true</c>.
    /// </summary>
    public static Task<bool> BooleanTrue
    {
        get
        {
            return booleanTrue;
        }
    }

    public static Task Completed
    {
        get
        {
            return booleanTrue;
        }
    }

    /// <summary>
    /// A task that has been completed with the value <c>false</c>.
    /// </summary>
    public static Task<bool> BooleanFalse
    {
        get
```

```
    {
        return TaskConstants<bool>.Default;
    }
}

/// <summary>
/// A task that has been completed with the value <c>0</c>.
/// </summary>
public static Task<int> Int32Zero
{
    get
    {
        return TaskConstants<int>.Default;
    }
}

/// <summary>
/// A task that has been completed with the value <c>-1</c>.
/// </summary>
public static Task<int> Int32NegativeOne
{
    get
    {
        return intNegativeOne;
    }
}

/// <summary>
/// A task that has been canceled.
/// </summary>
```

```
        public static Task Canceled
        {
            get
            {
                return TaskConstants<object>.Canceled;
            }
        }
    }

    /// <summary>
    /// Provides completed task constants.
    /// </summary>
    /// <typeparam name="T">The type of the task result.</typeparam>
    public static class TaskConstants<T>
    {
        private static readonly Task<T> defaultValue = Task.FromResult(default(T));
        private static Task<T> canceled = null; //Task.FromCanceled<T>(new CancellationToken(true));

        /// <summary>
        /// A task that has been completed with the default value of <typeparamref name="T"/>.
        /// </summary>
        public static Task<T> Default
        {
            get
            {
                return defaultValue;
            }
        }
```

```
        public static Task<T> Canceled
        {
            get
            {

                if (canceled == null)
                {
                    var tcs = new TaskCompletionSource<T>();
                    tcs.SetCanceled();
                    canceled = tcs.Task;
                }

                return canceled;
            }

        }


    }


    private sealed class ThreadPoolRegistration : IDisposable
    {
        private readonly RegisteredWaitHandle _registeredWaitHandle;

        public ThreadPoolRegistration(WaitHandle handle, TimeSpan timeout,
TaskCompletionSource<bool> tcs)
        {
```

```
                _registeredWaitHandle =
ThreadPool.RegisterWaitForSingleObject(handle,
                    (state, timedOut) =>
((TaskCompletionSource<bool>)state).TrySetResult(!timedOut), tcs,
                    timeout, executeOnlyOnce: true);
            }

            void IDisposable.Dispose() => _registeredWaitHandle.Unregister(null);
        }

    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using DBreeze.Utils;

namespace Raft
{
    public static class BiserExtension
    {
        public static byte[] SerializeBiser(this Biser.IEncoder obj)
        {
            return new Biser.Encoder().Add(obj).Encode();
        }
```

```
        public static byte[] SerializeBiser(this IEnumerable<Biser.IEncoder> objs)
        {
            var en = new Biser.Encoder();
            en.Add(objs, r => { en.Add(r); });
            return en.Encode();



        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Raft
{
    internal static class ProtobufSerializer
    {
        /// <summary>
        /// Deserializes protobuf object from byte[]
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="data"></param>
        /// <returns></returns>
        public static T DeserializeProtobuf<T>(this byte[] data)
        {
            T ret = default(T);
```

```
            using (System.IO.MemoryStream ms = new System.IO.MemoryStream(data))
            {

                ret = ProtoBuf.Serializer.Deserialize<T>(ms);
                ms.Close();
            }

            return ret;
        }

        public static object DeserializeProtobuf(byte[] data, Type T)
        {
            object ret = null;
            using (System.IO.MemoryStream ms = new System.IO.MemoryStream(data))
            {

                ret = ProtoBuf.Serializer.NonGeneric.Deserialize(T, ms);
                ms.Close();
            }

            return ret;
        }

        /// <summary>
        /// Serialize object using protobuf serializer
        /// </summary>
        /// <param name="data"></param>
        /// <returns></returns>
```

```
        public static byte[] SerializeProtobuf(this object data)
        {
            byte[] bt = null;
            using (System.IO.MemoryStream ms = new System.IO.MemoryStream())
            {
                ProtoBuf.Serializer.NonGeneric.Serialize(ms, data);
                bt = ms.ToArray();
                ms.Close();
            }

            return bt;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Raft.Utils
{
    internal class SDictAutoIdentityLong<TValue> : SDictionary<long, TValue>
    {

        long _safeLongIncreaser = 0;

        DateTime lastCheckedRemove = DateTime.Now;

        /// <summary>
```

```
        /// Remove helper. Leaves only last inserted values in quantity of topX.
        /// Second param helps to balance check interval in seconds.
        /// </summary>
        /// <param name="topX"></param>
        /// <param name="secondsCheckInterval"></param>
        public void DeleteKeysLessThenTop(int topX, long secondsCheckInterval)
        {

            if (lastCheckedRemove.AddSeconds(secondsCheckInterval) < DateTime.Now)
            {
                _lock.EnterWriteLock();
                try
                {
                    foreach (var k in _dict.Keys.Where(r => r < (_safeLongIncreaser - topX)).ToList())
                    {
                        _dict.Remove(k);
                    }
                }
                catch
                {
                }
                finally
                {
                    _lock.ExitWriteLock();
                }

                lastCheckedRemove = DateTime.Now;
            }
```

```
}

public override bool AddSafe(KeyValuePair<long, TValue> pair)
{
    bool r = false;
    long id = -1;

    _lock.EnterWriteLock();
    try
    {
        _safeLongIncreaser++;
        id = _safeLongIncreaser;

        _dict.Add(id, pair.Value);
        r = true;
    }
    catch
    {
        r = false;
    }
    finally
    {
        _lock.ExitWriteLock();
    }

    return r;
}
```

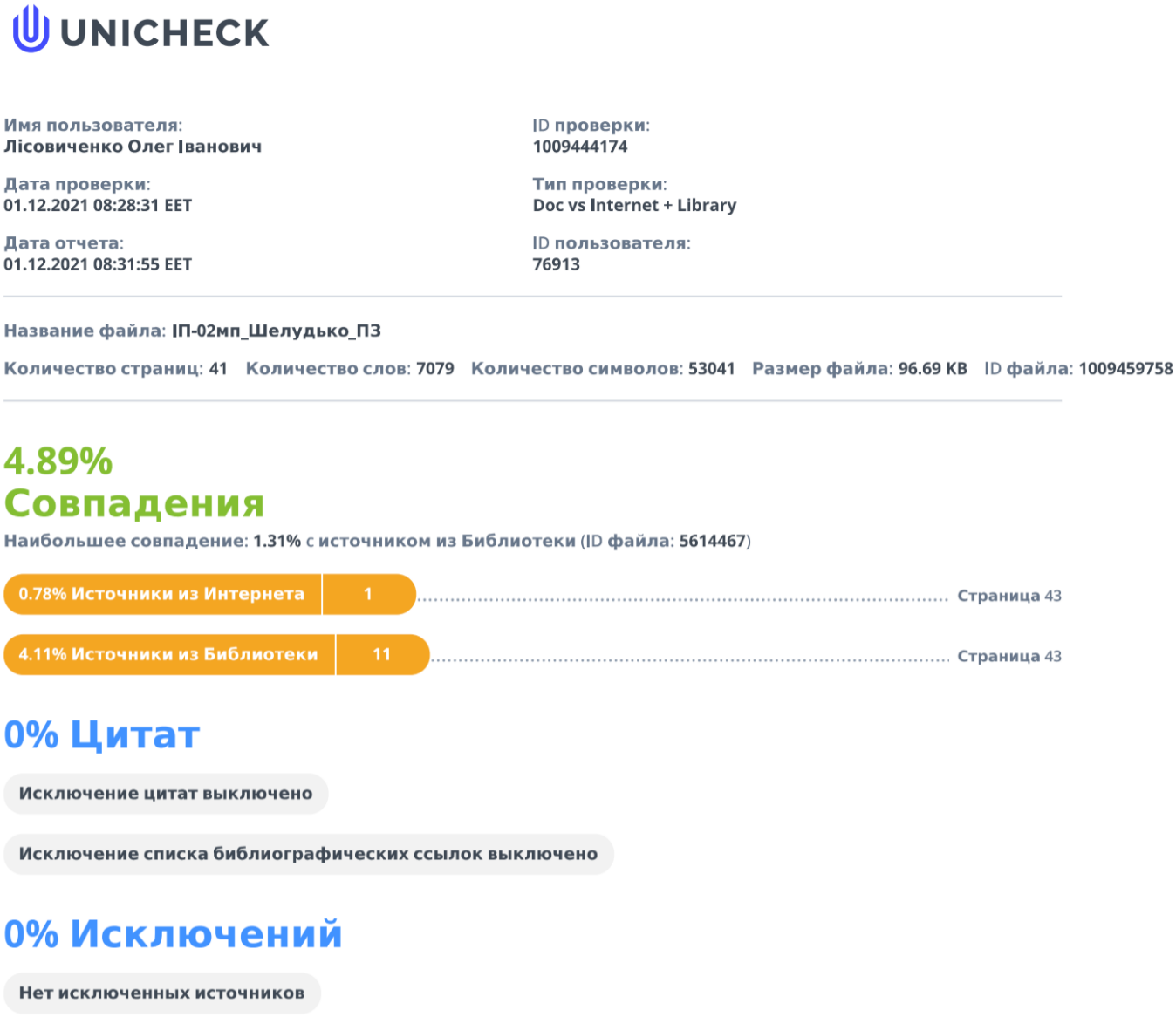
```
        /// <summary>
        /// Best Method to use here. Like Other Add Methods automatically creates key identity and returns it
        /// </summary>
        /// <param name="value"></param>
        /// <returns></returns>
        public long Add(TValue value)
        {
            long id = -1;
            _lock.EnterWriteLock();
            try
            {
                _safeLongIncreaser++;
                id = _safeLongIncreaser;

                _dict.Add(id, value);
            }
            catch
            {
                id = -1;
            }
            finally
            {
                _lock.ExitWriteLock();
            }

            return id;
        }
    }
```

```
}
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Consensus.Sepperin.Host;
using Raft.Transport;


class Program
{
   public static void Main(string[] args)
   {
      int port = int.Parse(Environment.GetEnvironmentVariable("SEPPERIN_PORT"));

      TcpRaftNode rn1 = null;
      rn1 = TcpRaftNode.GetFromConfig(System.IO.File.ReadAllText(
          "./cluster.config.json"),
      @"./DBreeze", port, new ConsoleLogger(),
      (entityName, index, data) =>
      {
         Console.WriteLine($"[Consensus] Committed {data.FirstOrDefault()}");
         return true;
      });

      Task.Run(async () =>
      {
         await Task.Delay(5000);
         byte value = 0;
```

```
            while (true)
            {
                Console.WriteLine($"[Data Processing] Proposing {value}");
                await rn1.AddLogEntryAsync(new[] {value});
                await Task.Delay(3000);
                value++;
            }
        });

        rn1.Start();

        Thread.Sleep(10000000);
    }
}
using System;
using Raft;

namespace Consensus.Sepperin.Host
{
    public class ConsoleLogger : IWarningLog
    {
        public void Log(WarningLogEntry logEntry)
        {
            Console.WriteLine(logEntry);
        }
    }
}
```

ДОДАТОК Е

Результати перевірки роботи на співпадіння



UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1009444174

Дата проверки:
01.12.2021 08:28:31 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
01.12.2021 08:31:55 EET

ID пользователя:
76913

Название файла: ІП-02мп_Шелудько_ПЗ

Количество страниц: 41 Количество слов: 7079 Количество символов: 53041 Размер файла: 96.69 KB ID файла: 1009459758

4.89%
Совпадения

Наибольшее совпадение: 1.31% с источником из Библиотеки (ID файла: 5614467)

0.78% Источники из Интернета 1 ..... Страница 43

4.11% Источники из Библиотеки 11 ..... Страница 43

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников