Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії

Назва теми

Рівень вищої освіти *другий (магістерський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр 240237.24.02.26 ПЗ

Виконав здобувач IV курсу, група КІ2м–22–1 _____ Владислав КРЕЩУК
Підпис Ініціали, прізвище

Керівник доктор філософії _____ Богдан САВЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер канд.ф–м. наук, доцент _____ Тетяна КИСІЛЬ
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КІІС _____ Ольга ПАВЛОВА
«_1_» _травня_ 2026 р. підпис Ініціали, прізвище

дата

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти ДРУГИЙ (МАГІСТЕРСЬКИЙ)

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
Ольга ПАВЛОВА
" 12 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Владиславу КРЕЩУКУ
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії

Керівник проекту (роботи) Богдан САВЕНКО, доктор філософії
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 12.01.2026 р. № 6

2. Термін подання здобувачем роботи на кафедру 01.05.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз відомих методів та засобів синтезу кіберфізична система на основі децентралізованого прийняття рішень

Архітектура та принципи синтезу кіберфізичних систем

Організація прийняття рішень в кфс згідно принципу децентралізації

Метод синтезу та реалізація кіберфізичних систем на основі децентралізованого прийняття рішень

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Тетяна КИСІЛЬ, доцент кафедри КІІС		
Антиплагіат	Андрій НІЧЕПОРУК, доцент кафедри КІІС		

7. Дата видачі завдання «_12_»_____01_____2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проєкту (роботи)	Термін виконання етапів проєкту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	12.01.2026	виконано
2	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	25.02.2026	виконано
3	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	15.03.2026	виконано
4	Робота над науковою статтею	01.04.2026	виконано
5	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	10.04.2026	виконано
6	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	25.04.2026	виконано
7	Оформлення пояснювальної записки згідно вимог	25.04.2026	виконано
8	Попередній захист ВКР	30.04.2025	виконано
9	Захист ВКР на засіданні ЕК	травень 2026 року	

Здобувач _____ Владислав КРЕЩУК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Богдан САВЕНКО
Підпис Ім'я, ПРІЗВИЩЕ

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: «Метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії»

Автор роботи: Крещук Владислав Олександрович

Керівник роботи: Савенко Богдан Олегович

Пояснювальна записка: 81 с., 9 рис., 10 табл., 4 дод., 81 джерело.

ВЗАЄМОБЛОКУВАННЯ, ЗАВДАННЯ, КОМП'ЮТЕРНА СИСТЕМА, ПРОЦЕСИ, ПРОТОКОЛ, РЕАКТОР, РОЗПОДІЛЕНА СИСТЕМА.

Об'єктом дослідження є процес уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.

Предметом дослідження є методи та засоби уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.

Метою роботи є забезпечення узгодженої координації процесів, мінімізація ризику виникнення циклічних залежностей та підвищення загальної стабільності функціонування системи шляхом розроблення методу уникнення взаємоблокувань завдань у розподілених системах на основі протоколу міжпроцесної взаємодії.

Для розв'язання поставлених задач використовувалися методи планування завдань в комп'ютерних системах, методи синтезу розподілених систем, теорія множин.

Наукова новизна отриманих результатів:

– розроблено новий метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії, особливістю якого є узгоджена координація процесів, мінімізація ризику виникнення циклічних залежностей, що дало змогу підвищити загальну стабільність функціонування розподілених системах.

На основі проведених досліджень розроблено метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.

Практична значимість отриманих результатів полягає у підвищенні точності виявлення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії для їх уникнення.

У вступі подано об’єкт та предмет дослідження, мету, наукову новизну та практичну цінність роботи, а також характеристику структури роботи.

У першому розділі проведено аналіз відомих рішень щодо уникнення взаємоблокувань завдань та взаємодії процесів в розподілених системах на основі протоколу міжпроцесної взаємодії.

У другому розділі здійснено дослідження предметної області та запропоновано архітектуру планувальника для процесів в розподілених системах на основі протоколу міжпроцесної взаємодії.

У третьому розділі розроблено цілісну методологічну основу керування розподіленими обчисленнями, у якій реактор виступає ключовим елементом поєднання структурної моделі залежностей і механізмів міжпроцесної взаємодії. Інтеграція стратегії розподілу реакторів із формалізованим протоколом обміну повідомленнями забезпечує не лише теоретично обґрунтоване уникнення взаємоблокувань, а й практичну гарантованість просування виконання критичних шляхів у складних гетерогенних середовищах. Також розроблено метод уникнення взаєблокувань завдань в розподілених системах представляє собою замкнену, узгоджену систему керування, що поєднує формальну модель, аналітичний інструментарій і механізми практичної реалізації, формуючи підґрунтя для побудови масштабованих, ефективних і стійких до взаємоблокувань розподілених обчислювальних середовищ.

У четвертому розділі здійснено розроблення архітектури системи, яка забезпечує повний цикл управління завданнями, тобто від надходження запиту до виділення ресурсів, синхронізації подій, контролю блокувань, обробки пріоритетів і динамічної міграції задач, що дозволяє моделювати реальні розподілені системи та експериментувати з різними стратегіями управління ресурсами.

У висновках підведено підсумки досягнення результатів з розв’язання завдань дослідження.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ІТ – інформаційні технології

КФС – кіберфізична система

ОС – операційна система

ПЗ – програмне забезпечення

IoT – Інтернет речей

TCP/IP – Transmission Control Protocol/Internet Protocol

ВСТУП

З кожним роком стає дедалі складніше підвищувати обчислювальну потужність окремої машини виключно за рахунок збільшення тактової частоти процесора чи кількості ядер, то сучасні інформаційні технології активно переходять до використання паралельних і розподілених систем [1, 2]. Такий підхід дозволяє досягти суттєвого приросту продуктивності шляхом декомпозиції складної задачі на менші підзадачі, які можуть виконуватися одночасно на кількох обчислювальних вузлах або процесах. Розподілені обчислення відкривають широкі можливості для масштабування сервісів, підвищення відмовостійкості та ефективного використання ресурсів, що є особливо актуальним в умовах стрімкого розвитку хмарних технологій, великих даних та високонавантажених систем [3, 4].

Водночас перехід від послідовної моделі виконання до паралельної та розподіленої суттєво ускладнює процес проєктування, реалізації та супроводу програмного забезпечення. Якщо в послідовних системах основна увага приділяється алгоритмічній оптимізації та ефективному використанню пам’яті, то в паралельних середовищах додаються проблеми синхронізації, координації доступу до спільних ресурсів, забезпечення узгодженості даних і коректної міжпроцесної взаємодії. Неправильна організація обміну повідомленнями або керування ресурсами може призвести до критичних збоїв у роботі системи, які складно виявити та усунути.

Однією з найбільш небезпечних і водночас поширених проблем у паралельних і розподілених системах є взаємоблокування. Воно визначається як стан, у якому виконання програми повністю або частково зупиняється через те, що система не може надати необхідні ресурси для продовження роботи процесів, або ж виникає циклічна залежність між ними. У такій ситуації кожен із процесів очікує на ресурс, який утримується іншим процесом, що, своєю чергою, також перебуває в стані очікування. У результаті жоден із процесів не може продовжити виконання, а система переходить у стан блокування.

Особливу складність становить те, що в розподілених системах взаємоблокування можуть виникати не лише через спільне використання апаратних ресурсів, а й через некоректну логіку протоколів обміну повідомленнями, затримки в мережі або порушення порядку виконання операцій. Виявлення таких ситуацій ускладнюється відсутністю централізованого контролю та необхідністю аналізу глобального стану системи. Наслідками взаємоблокувань можуть бути зниження продуктивності, втрата доступності сервісів, порушення цілісності даних та фінансові збитки.

У зв’язку з цим особливої актуальності набуває розробка ефективних методів запобігання виникненню взаємоблокувань ще на етапі проєктування системи. На відміну від підходів, що передбачають лише виявлення та усунення взаємоблокувань після їх виникнення, методи уникнення дозволяють організувати взаємодію процесів таким чином, щоб виключити саму можливість формування циклічних залежностей. Одним із перспективних напрямів у цьому контексті є використання формалізованих протоколів міжпроцесної взаємодії, які регламентують порядок обміну повідомленнями, виділення та звільнення ресурсів, а також узгодження станів процесів.

Таким чином, актуальність роботи зумовлена необхідністю підвищення надійності та ефективності розподілених систем шляхом запобігання взаємоблокуванням.

Актуальність роботи полягає в розробці методу уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.

Метою роботи є забезпечення узгодженої координації процесів, мінімізація ризику виникнення циклічних залежностей та підвищення загальної стабільності функціонування системи шляхом розроблення методу уникнення взаємоблокувань завдань у розподілених системах на основі протоколу міжпроцесної взаємодії.

Поставлена мета досягається розв’язанням таких основних завдань:

– проаналізувати відомі методи уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії;

– розробити метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії;

– розробити архітектуру та компоненти розподілених систем, включно з реалізованим методом уникнення взаємоблокувань;

– реалізувати метод уникнення взаємоблокувань в розподілених системах та провести з ним експериментальні дослідження.

Об’єктом дослідження є процес уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.

Предметом дослідження є методи та засоби уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.

Наукова новизна отриманих результатів:

– розроблено новий метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії, особливістю якого є узгоджена координація процесів, мінімізація ризику виникнення циклічних залежностей, що дало змогу підвищити загальну стабільність функціонування розподілених системах.

На основі проведених досліджень розроблено метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.

Практична значимість отриманих результатів полягає у підвищенні точності виявлення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії для їх уникнення.

Для розв’язання поставлених задач використовувалися методи планування завдань в комп’ютерних системах, методи синтезу розподілених систем, теорія множин.

За темою кваліфікаційної роботи опубліковано одну публікацію [81] у фаховому науковому журналі «*ВИМІРЮВАЛЬНА ТА ОБЧИСЛЮВАЛЬНА ТЕХНІКА В ТЕХНОЛОГІЧНИХ ПРОЦЕСАХ*» (м. Хмельницький, 2026. № 1. С. 294–307. DOI: https://doi.org/10.31891/2219–9365–2026–85–37).

1 АНАЛІЗ ВІДОМИХ МЕТОДІВ ТА ЗАСОБІВ УНИКНЕННЯ ВЗАЄМОБЛОКУВАНЬ ЗАВДАНЬ В РОЗПОДІЛЕНИХ СИСТЕМАХ

1.1 Огляд та поняття взаємоблокування завдань в розподілених системах

Розподілені системи стають дедалі поширенішими в сучасному інформаційному просторі, оскільки обсяги даних та інтенсивність обчислювальних навантажень постійно зростають, а підвищення продуктивності однієї окремої машини стає економічно та технічно менш доцільним. Масштабування «вгору» поступово поступається масштабуванню «в ширину», коли обчислювальні ресурси нарощуються шляхом об’єднання багатьох вузлів у єдину систему. Такий підхід дозволяє досягати високої продуктивності, гнучкості та відмовостійкості, однак водночас породжує низку нових викликів, які відсутні або менш виражені у традиційних послідовних виконаннях [5, 6].

У розподіленому середовищі виконання завдань відбувається паралельно на кількох вузлах, які взаємодіють між собою через мережу. Це створює додаткові складнощі, пов’язані з координацією доступу до ресурсів, синхронізацією станів, затримками передавання повідомлень і частковою відмовою окремих компонентів. Однією з критичних проблем є ситуація, коли система більше не може просуватися вперед через те, що завдання були розподілені таким чином, що вони взаємно блокують одне одного [7, 8]. У такому випадку процеси перебувають у стані очікування ресурсів або повідомлень, які ніколи не будуть отримані, що призводить до повної або часткової зупинки системи.

Зі збільшенням масштабу та складності розподілених систем зростає й складність виявлення та аналізу таких станів. Відсутність централізованого контролю, асинхронність обміну даними та динамічний характер розподілу ресурсів ускладнюють своєчасне врахування можливості виникнення глухого кута ще на етапі проєктування. У результаті навіть незначні помилки в логіці планування або розподілу ресурсів можуть мати суттєві наслідки для всієї системи.

У статті [9] було запропоновано схему диспетчеризації, що ґрунтується на принципах уникнення взаємоблокувань та забезпечує гарантії живучості системи.

Запропонований підхід спрямований на те, щоб система ніколи не переходила в стан взаємоблокування, навіть за умов інтенсивного розподілу задач і ресурсів. Вона поєднує механізми планування та контролю залежності між процесами, забезпечуючи формальні гарантії коректності виконання. Ця робота розпочинається з детального аналізу підходу та його теоретичних засад. Зокрема, розглядається певний вимір проблеми від розподілу сайтів між вузлами до формально визначеного з прямими наслідками для здійсненності та ефективності системи. Уточнення цього аспекту дозволяє глибше зрозуміти обмеження моделі та визначити умови, за яких гарантії уникнення взаємоблокувань зберігаються. В роботі пропонується конкретна стратегія розподілу вузлів на сайти в однорідному середовищі, де всі обчислювальні ресурси мають подібні характеристики. На основі цієї моделі здійснюється узагальнення для гетерогенного середовища, в якому вузли можуть відрізнятися за продуктивністю, обсягом пам’яті чи мережними параметрами. Окрему увагу приділено можливостям оптимізації ефективності в межах запропонованих розподілів без порушення гарантій живості системи. У роботі також представлено відповідні евристики планування, які дозволяють зменшити накладні витрати на координацію процесів та покращити балансування навантаження. Завершальним етапом є розробка моделі динамічної підграфової доцільності, що базується на реалізації цієї моделі і враховує зміну стану системи в реальному часі. Поєднання здійсненності, яку забезпечує стратегія розподілу реакторів, із гарантіями уникнення взаємоблокування та максимальної живучості системи формує цілісне й оптимальне середовище для ефективного розподіленого виконання складних обчислювальних задач.

У складних системах розподілу ресурсів [10, 11], зокрема в автоматизованих виробничих системах, гнучких виробничих модулях, логістичних комплексах, а також у високонавантажених програмно–апаратних середовищах, ефективне управління потоками завдань є критично важливим для забезпечення безперервності технологічних процесів. Використання багатопотокових і паралельних програм дозволяє значно підвищити продуктивність системи, скоротити час виконання операцій, оптимізувати використання обладнання та

зменшити енергоспоживання. Завдяки одночасному виконанню декількох процесів досягається краща утилізація ресурсів, знижується час простою машин і підвищується загальна пропускна здатність системи [12].

Проте впровадження паралелізму неминуче супроводжується появою додаткових ризиків і технічних ускладнень. До найбільш поширених проблем належать умови конкуренції, некоректна синхронізація доступу до спільних ресурсів, взаємоблокування, а також ситуації голодування [13, 14]. Такі явища можуть суттєво знижувати продуктивність системи, призводити до втрати узгодженості даних, збільшення часу простою обладнання та навіть створювати загрозу безпеці виробничих процесів. Особливо критичними ці наслідки є для автоматизованих виробничих систем, де порушення логіки керування може спричинити не лише економічні збитки, а й аварійні ситуації. Саме тому питання діагностики несправностей, виявлення причин блокувань, розробки методів їх усунення та запобігання взаємоблокуванням у паралельних програмах викликають є актуальними [15, 16].

З метою уникнення тупікових ситуацій у складних динамічних системах активно застосовується теорія дискретного керування [17, 18], яка забезпечує формальні засоби моделювання, аналізу та синтезу керуючих стратегій. Одним із найпотужніших інструментів у цій галузі є мережі Петрі, що завдяки своїй наочності та високій виразній здатності широко використовуються для моделювання автоматизованих виробничих систем [19], багатопоточних програмних систем [20], систем обслуговування, транспортних мереж та інших прикладних задач [21]. Мережі Петрі дозволяють формалізувати паралельність, конкуренцію, синхронізацію та розподіл ресурсів у межах єдиної математичної моделі, що значно полегшує аналіз поведінки системи.

Однак аналіз і керування загальними мережами Петрі є надзвичайно складними задачами [22, 23]. У деяких випадках вони належать до класу обчислювально важких або навіть нерозв’язних проблем. З огляду на це більшість досліджень зосереджена на вивченні керованих підкласів мереж Петрі, для яких можна сформулювати ефективні умови безблоковості та живучості [24].

Дослідження взаємоблокувань у таких моделях традиційно ґрунтується на структурному аналізі та аналізі множини досяжності. Проте метод побудови повної множини досяжних станів стикається з так званою проблемою вибуху стану, коли кількість можливих маркувань зростає експоненційно зі збільшенням розміру мережі. Унаслідок цього підхід на основі досяжності є придатним переважно для систем невеликого масштабу.

Наприклад, у роботі [25] запропоновано нову стратегію запобігання взаємоблокуванням, яка зменшує кількість обмежень і змінних у задачі цілочисельного лінійного програмування, однак вимагає генерації повного графа досяжності моделі мережі Петрі. Для великих мереж це призводить до значних обчислювальних витрат і практичних труднощів застосування. У зв’язку з цим значна частина сучасних стратегій контролю базується на структурному аналізі, зокрема на дослідженні ресурсів. Ключовою ідеєю є визначення умов керованості, за яких гарантується відсутність тупикових блокувань. На основі цього підходу було розроблено численні стратегії контролю, орієнтовані на керування завданнями і розподілом ресурсів [26]. Для забезпечення безблокової роботи необхідно, щоб кожен ресурс у мережі був керованим. Найпоширенішим способом досягнення цього є додавання спеціальних контролерів до моделі мережі Петрі. Проте якщо певні ресурси є керованими за своєю природою, то додаткові контролери стають надлишковими, що збільшує структурну складність системи та призводить до зайвих витрат ресурсів. Тому дослідники прагнуть знайти оптимальні методи керування, які дозволяють досягти повної керованості мережі з мінімальною кількістю додаткових елементів.

У звичайних мережах Петрі ресурс вважається керованим, якщо він залишається позначеним за всіх досяжних маркувань [27, 28]. Для таких мереж необхідною і достатньою умовою безблоковості є керованість усіх ресурсів. Для спеціалізованих класів мереж забезпечення відсутності взаємоблокувань еквівалентне гарантії живучості системи. Більшість існуючих стратегій реалізують запобігання тупіковим блокуванням шляхом додавання контролерів, які забезпечують виконання умови керованості ресурса. Якщо після цього мережа

зберігає властивості звичайної мережі Петрі, то живучість можна гарантувати на основі відомих структурних критеріїв. Проте на практиці додавання контролерів часто призводить до появи зважених дуг, унаслідок чого модель перетворюється на узагальнену зважену мережу Петрі. У такому випадку класичні умови керованості для звичайних мереж уже не можуть бути безпосередньо застосовані, що потребує розробки нових підходів.

Ідеальний супервізор, який забезпечує живу та безблокову роботу системи, повинен мати просту структуру, бути оптимальним з точки зору дозволеної поведінки та не вимагати надмірних обчислювальних ресурсів для синтезу й реалізації. Однак для великомасштабних систем такого універсального рішення фактично не існує. Більшість наявних підходів стикаються принаймні з однією з таких проблем: структурна складність моделі, надмірна вседозволеність поведінки або висока обчислювальна складність процедур синтезу [29].

У роботі [30] запропоновано політику запобігання взаємоблокуванням, засновану на аналізі графа досяжності моделі мережі Петрі із використанням теорії графів. Автор синтезував нові мережні елементи, які додаються до початкової моделі для забезпечення безблоковості. Проте застосування цього підходу до великих мереж обмежується проблемою вибуху стану через надмірні розміри графа досяжності. У подальшій роботі [31] було запропоновано вдосконалений метод із використанням редукції мережі Петрі, що зменшує складність аналізу. Хоча цей підхід є зручним для систем із невеликим простором станів, але він не гарантує оптимальності отриманого супервізора.

У дослідженні [32] використано теорію ризиків для побудови ефективного підходу, здатного знаходити оптимального супервізора за умови його існування. Водночас цей метод також характеризується високою обчислювальною та структурною складністю. Вибіркова політика керування ресурсами, запропонована в [33], дозволяє отримати супервізора невеликого розміру з досить дозволеною поведінкою, однак не містить формального доведення його максимальної дозволеності. Іншими словами, такий підхід зменшує складність наглядових структур, але не гарантує їх мінімальності та оптимальності.

У галузі планування для великих розподілених систем [34] упродовж останніх десятиліть було виконано значний обсяг наукових і прикладних досліджень. Особливу увагу приділено проблемі уникнення взаємоблокування процесів, яка є класичною та дослідженою у сфері проєктування операційних систем [35]. Такі системи функціонують в умовах одночасного виконання багатьох паралельних процесів і потребують ефективних механізмів координації доступу до спільних ресурсів. Проблема взаємоблокувань виникає тоді, коли декілька процесів взаємно очікують звільнення ресурсів, що призводить до повної зупинки їх виконання. Аналогічні виклики постають і під час розробки масштабних багатопотокових програмних комплексів, зокрема брокерів об’єктних запитів, які забезпечують взаємодію між клієнтськими та серверними компонентами розподілених систем [36]. У таких середовищах забезпечення узгодженості, продуктивності та відсутності блокувань є критично важливим завданням.

Семантика систем уникнення взаємоблокувань орієнтована на виявлення та запобігання взаємоблокуванням активно застосовується в різних дослідженнях і практичних реалізаціях. Особливо актуальною вона є в умовах обмежених ресурсів, наприклад у системах низьковольтних вимірювань [36], де доступні обчислювальні та енергетичні ресурси суттєво обмежені. Подібні підходи також впроваджуються у програмних середовищах з обмеженим обсягом пам’яті [37], де необхідно досягати балансу між ефективністю використання ресурсів і гарантіями коректного виконання. Зокрема, вони застосовуються для мінімізації явища інверсії пріоритетів [38], коли низькопріоритетне завдання блокує виконання більш пріоритетного, а також у системах із гетерогенними політиками уникнення взаємоблокувань [39], де різні компоненти можуть використовувати відмінні стратегії керування ресурсами.

Попри те, що більшість досліджень у сфері планування зосереджувалася переважно на забезпеченні дотримання термінів виконання та оптимізації часових характеристик системи, а не безпосередньо на оптимізації ресурсів чи уникненні взаємоблокувань, між цими аспектами існує певна концептуальна подвійність. Іншими словами, ефективність управління ресурсами безпосередньо впливає на

здатність системи вкладатися у визначені часові обмеження. У роботі [40] зазначена ідея отримує подальший розвиток через аналіз взаємозв'язку між розміром пулу потоків і пропускною здатністю (швидкістю) сайту. Передбачається, що зміна конфігурації пулу потоків може суттєво впливати як на продуктивність системи, так і на її стійкість до перевантажень.

У статті [41] здійснено ґрунтовний аналіз моделі напівфедеративного планування та досліджено час відгуку системи з урахуванням дисперсії швидкостей реакторів. Отримані результати демонструють, що варіативність характеристик обчислювальних компонентів може істотно впливати на загальну здійсненність і стабільність системи.

Таким чином, проблема синтезу ефективного, структурно простого й обчислювально прийнятного супервізора для великомасштабних розподілених систем залишається не вирішеною. Це зумовлює необхідність подальших досліджень, спрямованих на розробку нових методів керування, які поєднуватимуть гарантії живучості, відсутності взаємоблокувань та мінімальної структурної складності. Також, необхідно провести аналіз щодо дисперсії розмірів пулів потоків та її вплив на максимальну доцільність і здійсненність системи.

1.2 Процеси, розподіл ресурсів та взаємоблокування завдань в розподілених системах

Високопродуктивна мережа з'єднання є ключовим компонентом високопродуктивних обчислювальних систем, оскільки саме вона визначає ефективність міжвузлової взаємодії та масштабування паралельних застосунків [42]. Із зростанням кількості ядер і продуктивності обчислювальних вузлів масштаби мереж також збільшуються, що робить їхню масштабованість критичним чинником загальної продуктивності систем. Основні тенденції розвитку пов'язані зі збільшенням пропускної здатності, радіксу комутаторів і переходом до екзамасштабної архітектури [43]. Умови пост–Мура епохи потребують нових підходів до проєктування мережних рішень.

Аналіз систем рейтингу Top500 демонструє, що найважливішими характеристиками залишаються пропускна здатність, затримка, радіус комутатора та топологія. Подальше зростання продуктивності таких систем залежить від балансу між обчислювальною потужністю вузлів і пропускною здатністю мережі. Для екзамасштабних систем необхідні канали 400 Гбіт/с і вище, що створює виклики щодо енергоспоживання, щільності інтеграції та надійності. Уповільнення закону Мура та масштабування призводить до зростання енергетичних витрат на біт передачі, попри вдосконалення технологій [44].

Взаємоблокування процесів і завдань у розподілених системах є однією з найскладніших і найменш передбачуваних проблем сучасних обчислювальних середовищ. На відміну від централізованих систем, де всі ресурси та механізми керування зосереджені в одному місці, розподілена архітектура передбачає наявність кількох незалежних вузлів, що взаємодіють через мережу та спільно використовують ресурси. У таких умовах взаємоблокування виникає тоді, коли кілька процесів або завдань утворюють циклічну залежність у запитах на ресурси: кожен із них утримує певний ресурс і водночас очікує на інший, який уже зайнятий іншим учасником цього циклу. У результаті система переходить у стан, коли жоден із залучених процесів не може продовжити виконання, а загальна продуктивність різко знижується або повністю зупиняється [45, 46].

Особливість розподілених систем полягає в тому, що інформація про стан ресурсів є фрагментованою та розосередженою між вузлами [47, 48]. Кожен сервер або обчислювальний модуль володіє лише частковими відомостями про глобальний стан системи, що значно ускладнює виявлення циклів очікування. Якщо в централізованій системі можна побудувати повний граф розподілу ресурсів і проаналізувати його на наявність циклів, то в розподіленому середовищі для цього потрібна координація між вузлами, синхронізація даних і обмін службовими повідомленнями. Це створює додаткове навантаження на мережу та збільшує затримки, що особливо критично для систем реального часу або високонавантажених сервісів [49, 50].

Ще однією проблемою є непередбачуваність мережних затримок і можливі збої зв’язку. У розподілених системах взаємодія між компонентами відбувається через мережні протоколи, а тому час доставки повідомлень не є гарантовано сталим. Це може призводити до ситуацій, коли процес помилково вважає ресурс вільним або, навпаки, надто довго очікує підтвердження. Крім того, тимчасове відключення вузла або втрата пакета даних можуть бути інтерпретовані як блокування, хоча насправді йдеться лише про затримку. Таким чином, відмежування реального взаємоблокування від мережних аномалій стає окремим складним завданням [51, 52].

Важливою складовою проблеми є також різнорідність середовища. У сучасних розподілених системах можуть одночасно функціонувати вузли з різними обчислювальними потужностями, обсягами пам’яті та політиками планування. Це призводить до нерівномірного розподілу навантаження та створює передумови для локальних «вузьких місць», де ресурси вичерпуються швидше, ніж на інших вузлах. Якщо процеси активно мігрують між серверами або використовують віддалені ресурси, ризик утворення складних ланцюгів очікування зростає. Особливо небезпечними є ситуації, коли блокування охоплює критично важливі сервіси, наприклад системи автентифікації чи бази даних, оскільки це може паралізувати роботу всієї інфраструктури [53, 54].

Проблема взаємоблокування тісно пов’язана з питанням масштабованості. Зі зростанням кількості вузлів і процесів кількість потенційних комбінацій запитів на ресурси експоненційно збільшується. Навіть якщо ймовірність виникнення циклу очікування для окремої пари процесів є низькою, у великій системі вона стає статистично значущою. Тому механізми запобігання або виявлення взаємоблокувань повинні бути не лише коректними, а й ефективними з погляду витрат на їх реалізацію. Надмірно складні алгоритми контролю можуть самі по собі знижувати продуктивність системи, що створює дилему між безпекою виконання та швидкодією [55, 56].

Окремою проблемою є вибір стратегії реагування на виявлене взаємоблокування. У розподіленому середовищі немає єдиного центру прийняття

рішень, тому потрібно визначити, який процес або вузол має бути «жертвою» і примусово звільнити ресурси. Це може вимагати відкату транзакцій, повторного виконання обчислень або навіть втрати частини даних. У системах із високими вимогами до цілісності інформації такі дії повинні бути ретельно контрольованими, інакше ризик пошкодження даних може перевищити шкоду від самого блокування [57, 58].

Крім технічних аспектів, існує й проблема проєктування програмного забезпечення. Розробники часто не мають повної картини розподіленої взаємодії компонентів, особливо якщо система створюється кількома командами. Невдалий порядок захоплення ресурсів, відсутність чітких протоколів синхронізації або надмірна взаємозалежність сервісів можуть закладати потенційні сценарії взаємоблокування ще на етапі архітектурного проєктування. Усунення таких помилок після розгортання системи є складним і дорогим процесом [59, 60].

Таким чином, взаємоблокування процесів і завдань у розподілених системах становить багатовимірну проблему, що охоплює аспекти синхронізації, мережної взаємодії, масштабованості, відмовостійкості та архітектурного проєктування. Її складність зумовлена відсутністю глобального контролю, наявністю затримок і збоїв у комунікації, а також постійною динамікою середовища виконання. Ефективне розв’язання цієї проблеми потребує поєднання формальних методів аналізу, продуманих протоколів координації та відповідального підходу до розробки розподіленого програмного забезпечення [51–60].

Проблема взаємоблокування у розподілених системах породила значну кількість протоколів і алгоритмічних підходів, які відрізняються за моделлю системи, припущеннями щодо синхронності, способом збирання інформації та стратегією реагування. У загальному вигляді всі відомі рішення можна звести до трьох концептуальних стратегій: запобігання (prevention); уникнення (avoidance); виявлення з подальшим усуненням (detection and recovery). Однак у розподіленому середовищі кожна з цих стратегій набуває специфічних форм, оскільки відсутній глобальний спостерігач, а стан системи розосереджений між вузлами [61, 62].

Одним із найперших напрямів стали протоколи запобігання, що базуються на порушенні однієї з чотирьох умов виникнення взаємоблокування, зокрема: взаємного виключення; утримання і очікування; невід'ємності ресурсів; циклічного очікування. У розподілених системах найчастіше намагаються усунути саме циклічне очікування шляхом введення глобального порядку на ресурси. Відповідно до цього підходу кожному ресурсу або класу ресурсів присвоюється унікальний пріоритет чи ранг, і процеси зобов'язані запитувати ресурси лише у зростаючому порядку рангів. Така стратегія гарантує відсутність циклів у графі очікування, оскільки неможливо утворити замкнений ланцюг із монотонно зростаючих значень. У розподіленому середовищі це потребує погодження глобальної нумерації ресурсів між усіма вузлами, що саме по собі є нетривіальним завданням. Крім того, жорстка впорядкованість може знижувати паралелізм і призводити до зростання часу очікування, оскільки процеси змушені запитувати ресурси заздалегідь або в нефункціональному порядку [63, 64].

Іншим варіантом запобігання є протоколи, засновані на обмеженні утримання ресурсів. Наприклад, процес повинен запитати всі необхідні ресурси одночасно, і якщо хоча б один недоступний, запит відхиляється повністю. У розподіленій системі це реалізується через координовані запити до кількох вузлів із механізмом двофазного підтвердження. Проте такий підхід суттєво зменшує гнучкість і може викликати часті відмови у виділенні ресурсів навіть тоді, коли частина з них доступна. У великих системах це призводить до надмірного повторного надсилання запитів і додаткового мережного навантаження [65, 66].

Уникнення взаємоблокувань ґрунтується на ідеї перевірки безпечності стану системи перед виділенням ресурсу. Класичним прикладом є алгоритм банкіра, який оцінює стан системи щодо її перебування у безпечному стані після задоволення запиту. У розподіленому середовищі пряме застосування цього алгоритму ускладнюється тим, що необхідна повна інформація про максимальні потреби всіх процесів і поточний розподіл ресурсів. Для цього розроблялися модифіковані розподілені варіанти, в яких кожен вузол зберігає локальні дані та періодично обмінюється агрегованими станами з іншими. Такі протоколи можуть працювати

лише за умов відносно стабільного складу процесів і передбачуваних ресурсних вимог, що не завжди відповідає реаліям сучасних систем із динамічним масштабуванням [67, 68].

Найбільш поширеними в розподілених системах стали протоколи виявлення взаємоблокування з подальшим усуненням. Вони не намагаються повністю запобігти виникненню циклів, а зосереджуються на їх своєчасному виявленні. Одним із базових підходів є побудова розподіленого графа очікування. Кожен вузол підтримує локальний фрагмент графа, що відображає залежності між процесами на цьому вузлі, а також інформацію про віддалені залежності. Для виявлення циклів використовуються алгоритми поширення маркерів або зондувальні повідомлення. Відомим прикладом є алгоритм, у якому ініціатор підозри на взаємоблокування надсилає зондувальне повідомлення по ланцюгу залежностей. Якщо повідомлення повертається до ініціатора, то фіксується цикл. Цей підхід не потребує централізованого контролера і добре масштабується, однак може генерувати значну кількість службових повідомлень у системах із високою динамікою [69, 70].

Інший клас становлять централізовані або ієрархічні протоколи виявлення. У централізованій схемі всі вузли періодично надсилають інформацію про свої локальні залежності до координатора, який будує глобальний граф і перевіряє його на наявність циклів. Перевагою є простота аналізу, але недоліком це наявність єдиної точки відмови та потенційного вузького місця. Ієрархічні протоколи намагаються зменшити це навантаження, організовуючи вузли у кластери з локальними координаторами, які агрегують інформацію та передають її на вищий рівень. Такий підхід краще масштабується, проте ускладнює процедуру синхронізації й може страждати від затримок під час передачі агрегованих станів [71].

Окремий напрям [72] пов’язаний із використанням алгоритмів розподіленого виявлення циклів на основі знімків стану системи. Протокол глобального знімка, подібний до попереднього алгоритму, дозволяє зафіксувати узгоджений стан усіх вузлів без зупинки системи. На основі такого знімка можна побудувати повний

граф залежностей і перевірити його на циклічність. Перевага полягає в отриманні логічно узгодженого глобального стану навіть в асинхронному середовищі. Проте процедура створення знімка потребує додаткового обміну повідомленнями й може бути витратною у високонавантажених мережах.

У транзакційних розподілених базах даних [73] широко застосовуються протоколи, інтегровані з механізмами блокування. Наприклад, при використанні двофазного блокування система може вести таблиці очікування й періодично виконувати перевірку на цикли серед транзакцій. Деякі реалізації використовують тайм–аути. Якщо транзакція занадто довго очікує на ресурс, то вона примусово переривається. Хоча тайм–аути прості в реалізації, вони не гарантують точного виявлення циклу й можуть призводити до необґрунтованих відкатів.

Існують також протоколи [74], які засновані на часових мітках і впорядкуванні транзакцій. Вони застосовуються переважно у розподілених СУБД і базуються на принципі, що старші транзакції мають пріоритет над молодшими або навпаки. Якщо виникає конфлікт, одна з транзакцій примусово завершується відповідно до правила, що запобігає утворенню циклів. Ці протоколи фактично поєднують запобігання та виявлення, оскільки структура часових міток не дозволяє сформувати циклічну залежність. Водночас вони можуть спричиняти часті перезапуски транзакцій у середовищах із високим рівнем конкуренції.

У сучасних хмарних і мікросервісних архітектурах дедалі частіше використовуються координаційні сервіси [75], які реалізують механізми розподілених блокувань через узгоджені журнали або консенсусні алгоритми. У таких системах взаємоблокування на рівні логічних ресурсів зменшується завдяки використанню атомарних операцій і впорядкованих журналів подій. Однак це не усуває проблему повністю, оскільки можливі складні сценарії, де блокування виникає між кількома незалежними сервісами з різними механізмами синхронізації.

Окремим класом протоколів [76] є такі, що застосовують графи залежностей із мітками версій або поколінь. Вони дозволяють відрізняти застарілі повідомлення від актуальних і таким чином зменшують кількість хибних спрацьовувань у разі

мережних затримок. Такі підходи особливо важливі в асинхронних середовищах, де повідомлення можуть приходити в довільному порядку.

У системах реального часу [77] розроблялися спеціалізовані протоколи, що враховують пріоритети завдань. Вони інтегрують механізми уникнення взаємоблокування з протоколами успадкування або підвищення пріоритету, щоб запобігти інверсії пріоритетів і мінімізувати час блокування критичних задач. У розподіленому контексті це потребує передачі інформації про пріоритети між вузлами та узгодження політик планування.

Таким чином, відомі протоколи [62–77] вирішення проблеми взаємоблокування в розподілених системах охоплюють широкий спектр підходів. Він включає жорстке запобігання через впорядкування ресурсів і все включно з адаптивними алгоритмами виявлення з використанням зондувальних повідомлень, глобальних знімків і часових міток. Кожен із них має власні переваги та обмеження, що залежать від моделі системи, рівня синхронності, вимог до продуктивності та допустимих витрат на координацію. Саме різноманіття архітектури систем і сценаріїв застосування зумовлює відсутність універсального рішення.

Потреба у розробленні нового протоколу для запобігання та виявлення взаємоблокувань у розподілених системах зумовлена сукупністю технічних, архітектурних і експлуатаційних обмежень, які не повною мірою враховуються існуючими підходами. Класичні алгоритми боротьби із взаємоблокуваннями були створені переважно для централізованих або слабо розподілених середовищ, де існує можливість побудови глобального графа очікування ресурсів і централізованого аналізу циклів. У сучасних же розподілених інфраструктурах, тобто хмарних платформах, мікросервісних архітектурах, edge–обчисленнях, така централізація або неможлива, або економічно невигідна через масштаб, динамічність і географічну розосередженість компонентів.

По–перше, зростання масштабів систем істотно підвищує складність координації. Кількість вузлів, контейнерів, сервісів і паралельних запитів може змінюватися в режимі реального часу завдяки автоматичному масштабуванню. Традиційні протоколи, які передбачають періодичний глобальний обмін станами

або централізований моніторинг, створюють значне мережне навантаження та стають вузьким місцем продуктивності. Новий протокол повинен враховувати вимоги горизонтальної масштабованості, мінімізувати обсяг службового трафіку та працювати ефективно навіть за тисяч вузлів. По–друге, сучасні розподілені системи функціонують в умовах непередбачуваних затримок і часткових відмов. У мережах можливі тимчасові розриви зв’язку, асинхронність повідомлень і різна швидкість обробки на різних вузлах. Багато існуючих алгоритмів припускають відносно стабільну модель взаємодії або потребують узгодженого глобального часу. На практиці це призводить до хибних спрацьовувань, коли затримка інтерпретується як блокування, або, навпаки, до запізнілого виявлення реального взаємоблокування. Новий протокол має бути стійким до асинхронності, підтримувати часткову узгодженість даних і коректно працювати навіть за наявності тимчасових мережних аномалій. По–третє, значно зросли вимоги до безперервності сервісів. У критичних системах, тобто фінансових платформах, телекомунікаційних мережах, системах електронної комерції, навіть короткочасне блокування може призвести до фінансових втрат або втрати довіри користувачів. Існуючі механізми реагування часто передбачають примусовий відкат транзакцій або завершення процесів, що негативно впливає на якість обслуговування. Новий протокол повинен забезпечувати більш гнучкі механізми розв’язання конфліктів, наприклад адаптивний вибір «жертви», пріоритетність критичних задач або часткове розблокування без повного перезапуску. По–четверте, сучасні системи дедалі частіше використовують гетерогенні ресурси: багатоядерні процесори, графічні прискорювачі, віддалені сховища, різні типи пам’яті. Різнорідність політик планування й керування ресурсами створює складні сценарії залежностей, які не описуються простою моделлю «процес–ресурс». Традиційні протоколи зазвичай не враховують вагу або критичність ресурсу, його тип чи часові обмеження доступу. Новий підхід повинен інтегрувати інформацію про характеристики ресурсів і задач, щоб приймати більш обґрунтовані рішення щодо запобігання або усунення блокувань.

Крім того, зростає потреба у формальній верифікації та передбачуваності поведінки систем [78, 79]. У розподілених архітектурах із динамічною конфігурацією складно гарантувати відсутність взаємоблокувань лише на етапі проєктування. Тому новий протокол має бути формально описаним, придатним до математичного аналізу та моделювання, щоб можна було довести його коректність і оцінити межі здійсненності системи за різних навантажень.

Важливим аргументом є також інтеграція з сучасними підходами до керування навантаженням і пулом потоків. Розмір пулів, політики черг, балансування запитів безпосередньо впливають на ймовірність утворення циклів очікування. Існуючі протоколи зазвичай розглядають блокування як ізольовану проблему, не пов’язуючи її з параметрами продуктивності. Новий протокол повинен поєднувати контроль ресурсів із механізмами адаптивного планування [80], забезпечуючи баланс між максимальною пропускною здатністю та гарантією відсутності критичних блокувань.

Отже, розроблення нового протоколу є необхідним кроком для забезпечення надійності, масштабованості та передбачуваності сучасних розподілених систем. Він має поєднати децентралізований характер прийняття рішень, мінімізацію накладних витрат, стійкість до асинхронності, адаптивність до навантаження та формальну обґрунтованість. Лише комплексний підхід, що враховує сучасні технологічні реалії, дозволить ефективно зменшити ризик взаємоблокувань без істотного зниження продуктивності системи.

1.3 Постановка задачі

Поставлена мета досягається розв’язанням таких основних завдань:

– проаналізувати відомі методи уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії;

– розробити метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії;

– розробити архітектуру та компоненти розподілених систем, включно з реалізованим методом уникнення взаємоблокувань;

– реалізувати метод уникнення взаємоблокувань в розподілених системах та провести з ним експериментальні дослідження.

1.4 Висновки до першого розділу

Проаналізовано відомі методи та засоби уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії, а також визначено стратегію для покращення ефективності цього процесу.

2 АРХІТЕКТУРА ТИПОВИХ КЛАСІВ РОЗПОДІЛЕНИХ СИСТЕМ З ПРОТОКОЛОМ МІЖПРОЦЕСНОЇ ВЗАЄМОДІЇ

2.1 Архітектура та компоненти типових класів розподілених систем з протоколом міжпроцесної взаємодії

Зростання масштабів обчислювальної інфраструктури, перехід до сервісно орієнтованих архітектур, інтеграція хмарних і периферійних обчислень призвели до суттєвого ускладнення механізмів координації процесів. У таких умовах проблема взаємоблокувань перестає бути суто локальною помилкою синхронізації та перетворюється на системний ризик, що впливає на надійність, продуктивність і відмовостійкість всієї розподіленої екосистеми.

Одним із базових класів є клієнт серверні розподілені системи. У межах такого класу обчислювальні ресурси централізуються на серверній стороні, тоді як клієнти ініціюють запити та очікують відповіді. Попри відносну простоту логічної моделі, сучасні клієнт серверні системи рідко обмежуються одним сервером. Зазвичай це група взаємопов’язаних сервісів, які обмінюються запитами в процесі обробки одного користувацького звернення. За умов обмежених пулів потоків, обмеженої кількості з’єднань до бази даних або використання механізмів транзакційного блокування виникає ризик ситуацій, коли один сервер утримує ресурс і водночас очікує результату від іншого сервера, що перебуває в аналогічному стані. Актуальність проблеми зростає в умовах високого навантаження та горизонтального масштабування, коли кількість конкурентних запитів значно перевищує проєктні припущення. У дослідницькому аспекті важливим є аналіз впливу конфігурації пулів ресурсів і політик блокування на ймовірність формування циклічних залежностей.

Другим значущим класом є багаторівневі та мікросервісні архітектури. У таких системах функціональність декомпонується на незалежні сервіси, що взаємодіють через мережні протоколи або брокери повідомлень. Платформи контейнерної оркестрації, зокрема Kubernetes, стимулюють побудову великої кількості дрібних сервісів із власними життєвими циклами. З одного боку це

підвищує гнучкість, з іншого створює складну мережу залежностей. Якщо сервіси використовують синхронні виклики та утримують локальні ресурси до отримання віддаленої відповіді, то виникають умови для формування розподілених циклів очікування. Додатковий ризик з'являється під час використання брокерів повідомлень, наприклад RabbitMQ, коли механізми підтвердження отримання або транзакційні черги налаштовані некоректно. У дослідницькому плані актуальним є моделювання графів залежностей між сервісами та оцінка їхньої стійкості до часткових відмов і перевантажень.

Окремий клас формують кластерні та високопродуктивні обчислювальні системи. У таких системах велика кількість процесів виконує паралельні обчислення з періодичною синхронізацією. Бар'єрні механізми та колективні операції передбачають, що всі процеси повинні досягти певної точки виконання. Якщо один або кілька процесів затримуються через помилки або очікування повідомлень, то решта переходить у стан блокування. Проблема ускладнюється масштабом обчислень і нерівномірністю навантаження. Тому важливим є аналіз впливу топології мережі, латентності та алгоритмів розподілу задач на ймовірність глобального блокування.

Значну групу становлять розподілені транзакційні системи та розподілені бази даних. У таких системах механізми узгодженості даних реалізуються через протоколи блокування та підтвердження. У разі двофазного блокування транзакції можуть утримувати частину ресурсів і очікувати інші, що створює класичні умови для взаємоблокування, але вже в міжвузловому масштабі. У розподілених базах даних, наприклад Apache Cassandra, параметри узгодженості та політики реплікації впливають на структуру доступу до даних. Актуальною проблемою є баланс між високою узгодженістю та мінімізацією ризику блокувань, особливо в системах із високою інтенсивністю записів.

Системи «peer to peer» утворюють окремий клас, у якому відсутній централізований координатор. Вузли одночасно виконують ролі клієнтів і серверів, що створює динамічні та часто непередбачувані графи взаємодії. У таких системах блокування можуть виникати через взаємні запити ресурсів або через очікування

підтверджень у розподілених алгоритмах узгодження. Тому розроблення децентралізованих алгоритмів виявлення циклів очікування без глобального знання стану системи є актуальним завданням.

Системи реального часу та кіберфізичні комплекси також належать до класів із підвищеним ризиком блокувань. Тут важливу роль відіграють часові обмеження та пріоритети задач. Пріоритетна інверсія, коли задача з високим пріоритетом очікує ресурс, зайнятий менш критичною задачею, може призвести до каскадних затримок. У таких системах блокування мають не лише функціональні, а й безпекові наслідки.

Системи потокової обробки даних і розподілені черги повідомлень формують клас, у якому блокування часто пов’язані з механізмами керування потоком і зворотного тиску. Якщо виробники та споживачі повідомлень залежать від взаємних підтверджень і працюють із обмеженими буферами, виникають ситуації взаємного очікування. Актуальність проблеми посилюється в умовах інтеграції потокових платформ із транзакційними сховищами та аналітичними сервісами.

Отже, типові класи розподілених систем, у яких можливі блокування завдань, охоплюють широкий спектр архітектур від централізованих клієнт серверних до повністю децентралізованих «peer to peer» структур. У кожному класі механізми виникнення блокувань зумовлені специфікою координації ресурсів, синхронізації та протоколів міжпроцесної взаємодії. Актуалізація цієї проблематики пов’язана зі зростанням масштабів систем, підвищенням вимог до безперервності сервісів і ускладненням залежностей між компонентами. Саме тому систематичний аналіз класів архітектур є необхідною передумовою для подальшого моделювання сценаріїв взаємоблокувань і розроблення методів їх запобігання та виявлення.

Архітектура типового класу розподілених систем із протоколом міжпроцесної взаємодії (MPI/IPC) формується навколо сукупності обчислювальних вузлів, які з’єднані мережею передачі даних, поверх якої функціонують транспортні та прикладні протоколи обміну повідомленнями. У класичному випадку йдеться про системи, побудовані за моделлю клієнт–сервер, багаторівневою (n–tier) моделлю або кластерною архітектурою з розподіленими

службами. Прикладами практичної реалізації можуть бути платформи на кшталт Apache Hadoop або Apache Kafka, однак незалежно від конкретної технології базова структура включає вузли виконання, координуючі вузли, підсистему зберігання, комунікаційну підсистему, механізми синхронізації та протокол міжпроцесної взаємодії.

Протокол міжпроцесної взаємодії визначає формат повідомлень, порядок їх обміну, гарантії доставки, правила синхронізації та механізми обробки помилок. У розподіленому середовищі відсутній глобальний такт часу, тому синхронізація ґрунтується на логічних годинниках, кворумних алгоритмах або механізмах блокування ресурсів. Саме на рівні координації доступу до ресурсів (файлів, записів БД, черг, семафорів, мережних сокетів) і виникають передумови взаємоблокувань.

Типову конфігурацію архітектури задамо множиною вузлів $N = \{n_1, n_2, \ldots, nk\}$, множиною ресурсів $R = \{r_1, r_2, \ldots, rm\}$ та множиною процесів $P = \{p_1, p_2, \ldots, ps\}$, де процеси виконуються на вузлах і взаємодіють через обмін повідомленнями. Кожен процес може перебувати в одному зі станів: виконання; очікування ресурсу; блокування; завершення. Модель взаємодії формалізуємо у вигляді орієнтованого графа очікування так:

$$G = (P \cup R, E), \tag{2.1}$$

де ребро $(p_i \rightarrow r_j)$ означає запит ресурсу;

ребро $(r_j \rightarrow p_k)$ – виділення ресурсу процесу;

P та R – множини вершин, що відповідають процесам та ресурсам відповідно;

E – множина ребер, що означають запити та виділення ресурсу.

Наявність циклу в цьому графі є достатньою умовою виникнення взаємоблокування. Така модель відповідає класичній схемі взаємоблокування.

У розподілених системах типові архітектурні конфігурації, в яких з’являються сценарії блокування завдань, включають:

1) багаторівневу сервісну архітектуру (мікросервіси), де сервіс A викликає сервіс B, а той у відповідь звертається до A;

2) транзакційні системи з розподіленим блокуванням (two–phase locking);

3) системи з централізованим координатором, що здійснює серіалізацію доступу до спільних ресурсів;

4) peer–to–peer архітектури з взаємними запитами ресурсів;

5) системи–кластери з бар'єрною синхронізацією задач.

Сценарій взаємоблокування у мікросервісній архітектурі визначимо так: процес p_1 на вузлі n_1 утримує ресурс r_1 (локальна БД) і очікує відповіді від сервісу на вузлі n_2, який, у свою чергу, утримує ресурс r_2 і запитує дані в n_1, що потребують r_1. Виникає циклічне очікування $p_1 \rightarrow r_2 \rightarrow p_2 \rightarrow r_1 \rightarrow p_1$. У транзакційній моделі розподіленої СУБД дві транзакції можуть заблокувати різні записи й намагатися отримати доступ до вже зайнятих, тобто це класичне взаємоблокування на рівні двофазного блокування. У середовищі можлива ситуація, коли частина процесів досягає бар'єру синхронізації, а інші затримуються через очікування повідомлення, яке ніколи не буде доставлене через логічну помилку, тобто формується глобальне блокування всієї групи.

Моделі виникнення взаємоблокувань спираються на чотири необхідні умови: взаємне виключення, утримання й очікування, відсутність примусового вилучення, циклічне очікування. У розподілених системах ці умови посилюються через відсутність спільної пам'яті та затримки мережі. Додається ще один фактор такий як часткові відмови, що можуть імітувати взаємоблокування, наприклад, через втрату повідомлення або розрив з'єднання. Класи взаємоблокувань у таких системах доцільно поділити за кількома ознаками. За рівнем локалізації: локальні (в межах одного вузла) та глобальні (між вузлами). За природою ресурсу: об'єктні (блокування файлів, записів БД), комунікаційні (взаємне очікування повідомлень), транзакційні (блокування в рамках розподілених транзакцій), синхронізаційні (бар'єри, семафори). За часовою характеристикою: постійні (стійкі циклічні залежності) та псевдовзаємоблокування (тимчасові затримки, що імітують deadlock). За способом виявлення: централізовано детектовані (координатор

аналізує глобальний граф очікування) та децентралізовані (алгоритми на кшталт probe–based detection).

Окремо виділяють комунікаційні взаємоблокування, характерні для систем з повідомленнями, де процеси виконують блокуючі операції. Якщо p_1 виконує blocking send до p_2, а p_2 одночасно виконує blocking send до p_1 без попереднього receive, то обидва процеси переходять у стан очікування. Такий клас притаманний системам із синхронною передачею повідомлень.

Таким чином, архітектура розподілених систем з протоколом міжпроцесної взаємодії повинна розглядатися через призму залежності між процесами й ресурсами. Формальна модель у вигляді графа очікування дозволяє аналізувати потенційні цикли, а класифікація взаємоблокувань дає змогу визначати механізми їх запобігання, зокрема: упорядкування доступу до ресурсів; тайм–аути; алгоритми виявлення; відкат транзакцій; використання неблокуючих протоколів. Саме архітектурні рішення на рівні протоколу взаємодії визначають стійкість систем до сценаріїв із взаємоблокуваням у масштабованому розподіленому середовищі.

Задамо формалізований опис загальної моделі розподіленої системи з протоколом міжпроцесної взаємодії та моделей типових класів розподілених систем, де відображені їхні специфічні особливості.

Загальну формалізовану модель розподіленої системи визначимо через її базові компоненти так:

$$M_O = (P, N, C, S, M, F, T), \tag{2.2}$$

де P – множина процесів;

N – множина вузлів (обчислювальних елементів);

C – канали зв’язку; S – множина станів системи;

M – множина повідомлень; F – функція переходів станів;

T – модель часу.

Формалізуємо окремо кожну компоненту в формулі (2.2).

Здійснимо формалізацію процесу в розподілених системах так:

$$p_i = \langle s_i, I_i, O_i, d_i \rangle, \tag{2.3}$$

де s_i – локальний стан;

I_i – множина вхідних повідомлень для i – того процесу;

O_i – множина вихідних повідомлень для i – того процесу;

d_i – функція переходу для i – того процесу; $s_i \in S_i$;

S_i – множина локальних станів системи для i – того процесу;

$S_i \subset S$; **S** – множина станів системи; $I_i \subset$ M;

$O_i \subset$ M; M – множина повідомлень;

$d_i: S_i \times I_i \to S_i \times O_i$.

Таким чином, процес в розподілених системах задано за формулою (2.3), в якій враховано повідомлення та стани системи в процесі його життєвого циклу, що дає змогу відокремити повідомлення за типами і встановити їх подальші потенційні блокування.

Задомо модель каналу зв'язку так:

$$c_i = \langle O_{ij}, f_{ij} \rangle, \tag{2.4}$$

де O_{ij} – черга повідомлень;

f_{ij} – властивості каналу такі як надійність, впорядкованість, затримка;

i – номер процесу;

j – номер властивості каналу.

Глобальний стан містить всі локальні стани процесів та стани всіх каналів і задамо його так:

$$S = D_{i=1}^{n_S} S_i \times D_{i=1}^{n_S} D_{j=1}^{m_S} O_{ij}\ , \tag{2.5}$$

де S_i – множина локальних станів системи для i – того процесу;

$S_i \subset S$;

S – множина станів системи;

O_{ij} – черга повідомлень;

f_{ij} – властивості каналу такі як надійність, впорядкованість, затримка;

i – номер процесу;

j – номер властивості каналу;

n_S – кількість процесів;

m_S – кількість властивостей.

Таким чином, глобальний стан задано з врахуванням властивостей та локальних станів системи. Результатом задання буде матриця розмірністю $n_S \times m_S$.

Протокол міжпроцесної взаємодії визначимо так:

$$P_p = \langle R_P, A_P, d_i \rangle, \tag{2.6}$$

де d_i – функція переходу для i – того процесу відповідно до правил протоколу;

R_P – ролі процесів;

A_P – алфавіт повідомлень.

Протокол задає допустимі послідовності повідомлень, умови коректності, гарантії (безпека, живучість).

Визначимо також модель часу так:

1) синхронна (обмежені затримки, відомий верхній час);

2) асинхронна (затримки необмежені, немає глобального годинника);

3) частково синхронна.

Ця модель часу може враховувати визначені варіанти та вони будуть використані для уникнення взаємоблокувань завдань в розподілених системах.

Введемо загальну модель щодо типів розподілених систем для яких будуть розглядатись сценарії появи взаємоблокувань завдань так:

$$M_Z = \langle M_{\text{Consistency}}, M_{\text{FaultModel}}, M_{\text{Replication}}, M_{\text{Coordination}}, M_O \rangle, \tag{2.7}$$

де $M_{\text{Consistency}}$ – модель узгодженості;

$M_{\text{FaultModel}}$ – модель відмов;

$M_{\text{Replication}}$ – стратегія реплікації;

$M_{\text{Coordination}}$ – механізми координації;

M_O – загальна формалізована модель розподіленої системи (формула (2.2)).

Ця модель M_Z дозволяє отримати конкретний клас системи через обмеження параметрів.

Введемо на основі моделі M_Z (формула (2.7)) моделі типових класів розподілених систем.

Модель класу, що визначається клієнт–серверною архітектурою, задамо так:

$$M_{KS}^{Z} = \langle P_o^{Z,KS}, P_s^{Z,KS}, R_1^{Z,KS}, R_2^{Z,KS} \rangle, \tag{2.8}$$

де $P_o^{Z,KS}$ – клієнти;

$P_s^{Z,KS}$ – сервери;

$R_1^{Z,KS}$ – запити;

$R_2^{Z,KS}$ – відповіді.

Особливості визначення моделі M_{KS}^{Z} в тому, що відображені асиметричні ролі, централізована логіка, прості протоколи (request–response) та наявна слабка горизонтальна масштабованість

Модель для класу Peer–to–Peer (P2P) задамо так:

$$M_{\text{P2P}}^{Z} = \langle P_p^{Z,\text{P2P}}, P_o^{Z,\text{P2P}}, P_r^{Z,\text{P2P}} \rangle, \tag{2.9}$$

де $P_p^{Z,\text{P2P}}$ – всі вузли рівноправні;

$P_o^{Z,\text{P2P}}$ – логічна мережа;

$P_r^{Z,\text{P2P}}$ – функція маршрутизації.

Особливості моделі M^Z_{P2P} в тому, що відсутній центральний вузол, можна здійснювати децентралізований пошук, наявна самоорганізація та висока масштабованість.

Модель класу розподілених систем з реплікацією даних задамо так:

$$M^Z_{RS} = \langle P_p^{Z,RS}, P_d^{Z,\text{RS}}, P_r^{Z,\text{RS}}, P_c^{Z,\text{RS}} \rangle, \tag{2.10}$$

де $P_p^{Z,RS}$ – множина сховищ даних;

$P_d^{Z,\text{RS}}$ – множина даних;

$P_r^{Z,\text{RS}}$ – множина копій;

$P_d^{Z,\text{RS}}$ – логічна мережа.

Особливості моделі M^Z_{RS} враховують проблему консенсусу та необхідність координації

Модель класу розподілених транзакційних систем з реплікацією даних задамо так:

$$M^Z_{RTS} = \langle P_p^{Z,RTS}, P_t^{Z,\text{RTS}}, P_a^{Z,\text{RTS}}, P_c^{Z,\text{RTS}} \rangle, \tag{2.11}$$

де $P_p^{Z,RTS}$ – множина сховищ даних;

$P_t^{Z,\text{RTS}}$ – множина транзакцій;

$P_r^{Z,\text{RS}}$ – множина об’єктів в розподіленому середовищі;

$P_d^{Z,\text{RS}}$ – множина протоколів.

Особливості моделі M^Z_{RTS} полягають в атомарності, блокуванні та чутливості до відмов координатора.

Модель для класу систем з консенсусом задамо так:

$$M^Z_{\text{CONS}} = \langle P_1^{Z,\text{CONS}}, P_2^{Z,\text{CONS}}, P_3^{Z,\text{CONS}}, P_4^{Z,\text{CONS}} \rangle, \tag{2.12}$$

де $P_1^{Z,\mathrm{CONS}}$ – забезпечення безпеки; $P_2^{Z,\mathrm{CONS}}$ – забезпечення живучості; $P_3^{Z,\mathrm{CONS}}$ – забезпечення вимог; $P_4^{Z,\mathrm{CONS}}$ – виконання згідно вставнолених термінів.

Модель для класу подієво–орієнтованих систем задамо так:

$$M_{PO}^{Z} = \langle P_1^{Z,PO}, P_2^{Z,PO}, P_3^{Z,PO}, P_4^{Z,PO} \rangle, \tag{2.13}$$

де $P_1^{Z,PO}$ – множина виробників;

$P_2^{Z,PO}$ – множина споживачів;

$P_3^{Z,PO}$ – множина подій;

$P_4^{Z,PO}$ – множина менеджерів.

Особливості моделі M_{PO}^{Z} у врахуванні асинхронності, слабкому зв'язуванні, масштабованості та у досягненні кінцевого стану дії.

Модель для класу розподілених обчислювальних систем задамо так:

$$M_{RO}^{Z} = \langle P_1^{Z,RO}, P_2^{Z,RO}, P_3^{Z,RO}, P_4^{Z,RO} \rangle, \tag{2.14}$$

де $P_1^{Z,RO}$ – множина вузлів;

$P_2^{Z,RO}$ – множина завдань;

$P_3^{Z,RO}$ – множина дій планувальника;

$P_4^{Z,RO}$ – менеджер ресурсів.

Особливості моделі M_{RO}^{Z} пов'язані із паралельними задачами, балансуванням навантаження та контролем ресурсів.

Таким чином, задані моделі типів розподілених систем формулами (2.7)–(2.14) і є основою для розроблення сценаріїв взаємоблокування в них. Відмінності між ними подано в табл. 2.1.

Загальна модель розподілених систем зводиться до множини процесів, моделі обміну повідомленнями, функції переходів станів, моделі часу, моделі відмов та моделі узгодженості. Будь–який конкретний клас системи отримується

шляхом фіксації ролей, обмеження топології, визначення моделі узгодженості, вибору протоколу координації та визначення моделі відмов.

Таблиця 2.1 – Узагальнююча таблиця відмінностей

Клас	Централізація	Узгодженість	Відмово–стійкість	Масшта–бованість
Client–Server	Висока	Зазвичай strong	Низька	Середня
P2P	Низька	Eventual	Висока	Висока
Репліковані	Часткова	Налаштовувана	Висока	Висока
Транзакційні	Координатор	Strong (ACID)	Середня	Низька
Консенсус	Децентралізована	Strong	Висока	Середня
Event–driven	Децентралізована	Eventual	Висока	Висока

2.2 Моделі взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії

Взаємоблокування у розподілених системах має специфіку, якої немає в централізованих системах, а саме: відсутність глобального стану; затримки мережі; часткові відмови вузлів; асинхронна міжпроцесна взаємодія. У розподілених системах взаємоблокування виникає внаслідок циклічної залежності процесів при захопленні ресурсів або під час міжпроцесної взаємодії через обмін повідомленнями. На відміну від централізованих систем, у розподілених системах відсутній глобальний стан, що ускладнює виявлення та усунення взаємоблокувань.

У розподілених системах проблема блокування завдань є однією з ключових загроз надійності та безперервності функціонування. На відміну від централізованих обчислювальних середовищ, де стан усіх процесів та ресурсів може контролюватися єдиним диспетчером, розподілена система складається з множини автономних вузлів, які взаємодіють через мережу, обмінюючись повідомленнями та координуючи доступ до спільних ресурсів. Відсутність глобального часу, асинхронність передачі даних, можливі затримки або втрати

пакетів, часткові відмови компонентів і реплікація даних створюють передумови для складних сценаріїв блокування завдань, які можуть проявлятися як повна зупинка групи процесів, тривале очікування або навіть активна, але безрезультатна взаємодія.

Модель блокування завдань у розподіленій системі зазвичай базується на формальному поданні множини процесів і ресурсів та відношень між ними. Нехай система містить множину процесів, які виконуються на різних вузлах, та множину ресурсів, що можуть бути локальними або глобально розподіленими. Процес може перебувати у стані виконання, очікування або блокування. Блокування виникає тоді, коли процес не може продовжити роботу без отримання певного ресурсу або повідомлення від іншого процесу. Якщо така залежність має лінійний характер, то система лише уповільнюється, однак якщо формується замкнений цикл очікування, то виникає взаємоблокування, тобто стан, у якому жоден із процесів циклу не здатний продовжити виконання.

Найбільш класичною моделлю є модель циклічного очікування ресурсів. У ній система подається у вигляді орієнтованого графа очікування, де вершинами є процеси, а дуга від одного процесу до іншого означає, що перший очікує ресурс, утримуваний другим. У централізованій системі побудова такого графа відносно проста, але в розподіленому середовищі кожен вузол володіє лише частковою інформацією. Потенційний сценарій виникнення взаємоблокування може виглядати так: транзакція на одному сервері блокує локальний ресурс і надсилає запит до іншого вузла, де паралельно виконується інша транзакція, що вже утримує потрібний ресурс і, у свою чергу, очікує на відповідь від першого сервера. Утворюється замкнений цикл, який не може бути розірваний без втручання зовнішнього механізму. Подібні ситуації часто виникають у розподілених банківських системах або системах електронної комерції, де одночасно виконуються тисячі транзакцій із різних регіонів.

Іншим поширеним варіантом є блокування в межах розподілених транзакцій, що використовують протокол двофазного блокування. Суть цього підходу полягає у тому, що транзакція спочатку накопичує всі необхідні блокування, а вже потім

переходить до фази їх звільнення. У розподіленій архітектурі, де транзакція може охоплювати декілька баз даних або сервісів, кожен вузол накладає власні обмеження доступу. Якщо порядок захоплення ресурсів різними транзакціями не узгоджений глобально, то виникає ризик взаємоблокування. Наприклад, мікросервіс замовлень резервує запис про покупку і очікує підтвердження від сервісу складу, тоді як сервіс складу вже заблокував інформацію про товар і звертається до сервісу платежів, який, у свою чергу, очікує завершення першої транзакції. Ланцюг очікування замикається, і система зупиняється у певній частині функціоналу.

Особливу категорію становлять сценарії блокування через міжпроцесну взаємодію повідомленнями. У сервісно–орієнтованих архітектурах запит одного сервісу до іншого може бути синхронним, тобто викликаючий процес очікує відповіді. Якщо архітектура має циклічні залежності, то можливий сценарій, коли сервіс A звертається до сервісу B, той — до сервісу C, а сервіс C — знову до сервісу A. Усі три компоненти переходять у стан очікування відповіді, що ніколи не буде отримана. На практиці подібні ситуації можуть виникати у великих кластерах контейнеризованих застосунків, де взаємодія між мікросервісами не була спроєктована з урахуванням уникнення циклів залежностей.

У розподілених системах з менеджерами блокувань можливі сценарії так званого «мережного блокування». Якщо вузол, який отримав розподілене блокування, втрачає зв’язок із мережею, інші вузли можуть вважати ресурс зайнятим протягом невизначеного часу. З іншого боку, у разі помилки механізму синхронізації можливе паралельне надання одного й того самого блокування кільком вузлам, що призводить до неконсистентності даних. Особливо критичними такі ситуації є у фінансових або виробничих системах, де подвійне виконання операції може спричинити матеріальні збитки.

Окрім класичного взаємоблокування, у розподілених системах спостерігаються інші форми блокування завдань. Однією з них є голодування, коли процес теоретично може отримати ресурс, але постійно витісняється іншими через політику пріоритетів або нерівномірний розподіл навантаження. У великих

обчислювальних кластерах задачі з низьким пріоритетом можуть залишатися в черзі невизначено довго, що знижує ефективність використання ресурсів. Іншою формою є livelock, тобто стан, у якому процеси активно змінюють свій стан, намагаючись уникнути конфлікту, але жоден із них не досягає прогресу. Наприклад, два вузли можуть по черзі відпускати ресурс, щоб уникнути потенційного конфлікту, і одразу ж знову намагатися його захопити, повторюючи цикл нескінченно.

Потенційні сценарії блокування часто пов’язані з відсутністю глобального порядку захоплення ресурсів. Якщо в системі не визначено єдиний алгоритм, який гарантує, що всі процеси запитують ресурси в однаковій послідовності, імовірність утворення циклу суттєво зростає. Наприклад, у хмарній інфраструктурі одна служба може спочатку резервувати обчислювальні ресурси, а потім мережні, тоді як інша – у зворотному порядку. За певного навантаження це призводить до взаємного блокування. Аналогічні проблеми виникають у міжкластерних транзакціях, коли операція охоплює кілька дата–центрів із різною політикою керування доступом.

Важливу роль відіграють мережні затримки та часткові відмови. У розподіленій системі відсутність відповіді не завжди означає відмову вузла. Можливо, повідомлення просто затримується. Якщо механізм тайм–аутів налаштований некоректно, то система може передчасно вважати ресурс недоступним і запускати процедури повторного захоплення, що призводить до лавиноподібного збільшення кількості запитів і нових блокувань. У протилежному випадку занадто довгі тайм–аути можуть спричинити тривале простоювання ресурсів.

Запобігання блокуванням передбачає впровадження формальних моделей та алгоритмів координації. Одним із підходів є використання часових міток, коли кожному процесу присвоюється унікальний ідентифікатор або логічний час, що визначає порядок доступу до ресурсів. Старші процеси можуть мати пріоритет у конфліктних ситуаціях, що усуває можливість утворення циклу. Інший підхід полягає у періодичному виявленні циклів у графі очікування шляхом обміну

спеціальними службовими повідомленнями між вузлами. У разі виявлення взаємоблокування один із процесів примусово переривається, а його ресурси звільняються.

У сучасних розподілених архітектурах дедалі частіше застосовуються асинхронні механізми взаємодії та ідемпотентні операції, які зменшують потребу в жорстких блокуваннях. Наприклад, замість синхронного очікування відповіді сервіс може передавати повідомлення через чергу подій і продовжувати виконання інших задач. Такий підхід знижує ризик циклічних залежностей, але потребує складнішої логіки узгодження станів.

Таким чином, моделі блокування завдань у розподілених системах охоплюють широкий спектр явищ. Наприклад, від класичного взаємоблокування ресурсів до голодування і livelock. Потенційні сценарії їх виникнення пов’язані з конкурентним доступом до розподілених ресурсів, циклічними залежностями сервісів, мережними затримками, відмовами вузлів та неузгодженістю політик керування. Аналіз таких сценаріїв дозволяє проєктувати системи з урахуванням можливих ризиків, впроваджувати механізми запобігання та забезпечувати стійкість до збоїв. Ефективне управління блокуваннями є необхідною умовою забезпечення масштабованості, продуктивності та надійності сучасних розподілених інформаційних систем.

У розподілених системах блокування завдань виникає тоді, коли процеси або транзакції не можуть продовжити виконання через очікування ресурсів, повідомлень або синхронізаційних подій. Особливістю таких систем є відсутність глобального стану, асинхронність взаємодії, затримки мережі, можливі відмови вузлів та конкурентний доступ до розподілених ресурсів. Блокування може бути частковим (затримка виконання), повним (взаємоблокування) та тривалим очікуванням.

Введемо моделі блокувань завдань з врахуванням типів розподілених систем.

Модель циклічного очікування ресурсів задамо так:

$$P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_k \rightarrow P_1, \tag{2.15}$$

де $P = \{P_1, P_2, \ldots, P_n\}$ – множина процесів;

$R = \{R_1, R_2, \ldots, R_m\}$ – множина ресурсів.

Блокування існує, якщо сформовано замкнений цикл очікування.

Наприклад, приклад потенційного сценарію у розподіленій банківській системі може бути такий, а саме:

1) транзакція T1 (вузол A) блокує рахунок X;

2) транзакція T2 (вузол B) блокує рахунок Y;

3) T1 запитує Y;

4) T2 запитує X.

Таким чином, утворюється цикл і система зупиняється.

Модель блокування через розподілені транзакції (2PL) характеризується використанням механізму двофазного блокування, в якому такі дії:

1) фаза зростання (захоплення блокувань);

2) фаза спадання (звільнення).

Потенційний сценарій, наприклад мікросервісна система, може бути такий, а саме:

1) сервіс замовлень блокує запис замовлення;

2) сервіс складу блокує товар;

3) обидва очікують підтвердження один від одного;

4) система входить у стан взаємоблокування.

Модель блокування через обмін повідомленнями грунтується га ситуації, коли процес блокується, очікуючи відповідь від іншого процесу.

Сценарій можна задати так для системи з RPC–запитами:

1) сервіс A викликає сервіс B;

2) сервіс B викликає сервіс C;

3) сервіс C викликає сервіс A.

Усі очікують відповіді і, тому цикл замкнувся.

Модель блокування через розподілені блокування характеризується використанням централізованого або кворумного менеджеру блокувань (наприклад, на базі ZooKeeper).

Сценарій задамо так:

1) вузол А отримав lock;

2) мережний розрив;

3) вузол В вважає, що lock вільний.

Таким чином, виникає неконсистентний стан або тривале блокування.

Модель голодування характеризується тим, що процес постійно витісняється іншими. Сценарій згідно цієї моделі задамо так з врахуванням розподіленої черги завдань:

1) пріоритетні задачі постійно додаються;

2) низькопріоритетна задача ніколи не отримує ресурс.

Тоді, виникає блокування низькопріоритетних завдань.

Суть моделі livelock в тому, що процеси активно працюють, але не просуваються вперед. Наприклад, такий сценарій, в якому наявні два вузли:

1) вузол А відпускає ресурс, щоб уникнути взаємоблокування;

2) вузол В робить те саме.

Обидва вузли повторюють дію нескінченно.

Загалом причинами виникнення блокувань є відсутність глобального контролера, асиметрія затримок мережі, часткові відмови вузлів, некоректна стратегія тайм–аутів та неправильне використання розподілених транзакцій. Блокування завдань у розподілених системах мають складну природу через розподілену пам’ять, відсутність глобального часу, асинхронність, мережні збої. Моделі блокування дозволяють формалізувати поведінку системи, а аналіз потенційних сценаріїв допомагає проєктувати надійні архітектури.

2.3 Висновки до другого розділу

Формалізовані моделі типів розподілених систем, задані формулами (2.7)–(2.14), утворюють методологічну основу для аналізу та побудови сценаріїв взаємоблокування. Представлення їх відмінностей дає змогу систематизувати ключові характеристики кожного типу систем та враховувати їх під час моделювання критичних ситуацій. Загальна модель розподілених систем охоплює множину процесів, механізми обміну повідомленнями, функцію переходів станів, модель часу, модель відмов і модель узгодженості.

Причини блокувань у розподілених пов’язані з відсутністю глобального контролера й єдиного часу, асинхронності обміну, мережних затримок та збоїв, часткових відмов вузлів, а також помилок у стратегіях тайм–аутів і використанні розподілених транзакцій. Саме тому формалізація моделей блокування та системний аналіз потенційних сценаріїв їх виникнення є необхідною умовою проєктування надійних, відмовостійких і масштабованих архітектур розподілених систем.

3 МЕТОД УНИКНЕННЯ БЛОКУВАНЬ ЗАВДАНЬ В РОЗПОДІЛЕНИХ СИСТЕМАХ НА ОСНОВІ ПРОТОКОЛУ МІЖПРОЦЕСНОЇ ВЗАЄМОДІЇ

3.1 Моделювання взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії

Моделювання взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії здійснимо з використанням теорії графів та реакторів, що дають змогу відокремити події у вузлах.

Для загальних орієнтованих ациклічних графів (ОАГ), що моделюють графи викликів у розподілених системах, питання конфлікту при призначенні вузлів реакторам набуває принципового значення. На відміну від структур типу дерева, ОАГ допускають існування вузлів із кількома вхідними шляхами, що породжує потенційну неоднозначність у визначенні реактора для такого вузла. Якщо в дереві кожна вершина має єдиний шлях від кореня, то в ОАГ одна й та сама вершина може належати кільком шляхам різної довжини. Відтак виникає потреба у формальному механізмі визначення пріоритету між шляхами з метою забезпечення узгодженості розподілу.

Пропонована стратегія ґрунтується на принципі пріоритету довшого шляху. Інтуїтивно це відповідає орієнтації на критичний шлях, тобто на ту послідовність вузлів, яка визначає мінімальний можливий час завершення всього графа. Якщо вузол належить двом або більше шляхам, то реактор для нього визначається тим шляхом, який має більшу довжину. У випадку рівності довжин застосовується довільний, але детермінований порядок, що не впливає на фінальне забарвлення. Такий підхід забезпечує, що конфлікти вирішуються не локально, а з урахуванням глобальної структури графа.

Ключовою метою є збереження інваріанта нерозподілу, який формулюється як умова $\lceil L / n \rceil \geq \max(R(p))$ для будь–якого шляху p, де L – довжина критичного шляху, n – кількість реакторів, а $\max(R(p))$ – максимальна кількість вузлів шляху p, призначених одному реактору. Цей інваріант гарантує, що якщо ресурси достатні

для обслуговування критичного шляху, то стратегія розподілу не створить штучного вузького місця через невдале призначення реакторів.

Реалізаційно стратегію базуватимемо на двофазному алгоритмі. На першій фазі для кожного джерельного вузла ОАГ виконаємо DFS–обхід. Уздовж кожного шляху використовується циклічний лічильник із періодом n, що забезпечує рівномірне чергування реакторів. Кожному шляху присвоюється унікальний ідентифікатор, а його довжина накопичується як рекурсивний параметр. Результатом цієї фази є хеш–таблиця, що відображає ідентифікатори шляхів у їхню довжину.

На другій фазі здійснюється повторний обхід графа, під час якого кожна вершина отримує реактор відповідно до того шляху, що має максимальну довжину серед усіх шляхів, які проходять через цю вершину. Завдяки доведеному факту, що шляхи однакової довжини можна впорядковувати довільно без зміни кінцевого забарвлення, алгоритм є детермінованим і не залежить від конкретного порядку обходу.

Асимптотична складність процедури у найгіршому випадку становить O(|V|(|V|+|E|)), оскільки алгоритм запускається з кожного джерела. Додатковий обхід та операції з хеш–таблицею не змінюють порядку складності. Важливо, що алгоритм може виконуватися офлайн, під час етапу завантаження або компіляції графа, що усуває його вплив на час виконання системи.

Доведення коректності базується на аналізі можливих спроб порушення інваріанта нерозподілу. Припустимо, що існує шлях p, для якого можна сконструювати набір шляхів pi, що перезаписують призначення його вершин так, щоб max(R(p)) перевищив допустиму межу. Для цього кожен шлях pi повинен бути довшим за p, оскільки коротший або рівний шлях не може змінити забарвлення через правило пріоритету. Більше того, щоб змінити колір вершини, довжина нового шляху має перевищувати попередню щонайменше на n (через циклічність мод n). Це означає, що кожна така ітерація збільшує L щонайменше на n. Відтак будь–яка спроба примусового перезапису неминуче призводить до зростання L, що

зберігає виконання інваріанта. Таким чином, не існує конструкції, яка могла б порушити інваріант без одночасного збільшення критичного шляху.

Для систем, близьких до межі здійсненності, випадковий розподіл дедалі частіше порушує інваріант нерозподілу, особливо зі збільшенням кількості реакторів. Натомість запропонована стратегія не демонструє жодного порушення в експериментальних серіях. Це свідчить про те, що стратегічний, структурно обґрунтований розподіл є не лише оптимізаційною перевагою, а необхідною умовою забезпечення коректності у масштабованих розподілених системах.

Отже, запропонований підхід забезпечує формально доведену узгодженість призначення реакторів у довільних ОАГ, зберігає інваріант нерозподілу та гарантує пріоритет критичного шляху. Це створює теоретичну основу для побудови відмовостійких і продуктивних механізмів планування в розподілених середовищах.

Реактор у контексті графа викликів або обчислювального графа походить із архітектурного шаблону Reactor pattern та є одним із базових підходів до побудови високопродуктивних подієво–орієнтованих систем. У класичному розумінні реактор це механізм організації обробки подій, що поєднує мультиплексування джерел подій, їх диспетчеризацію та виконання відповідних обробників у межах контрольованого середовища виконання. Однак у дослідницькому контексті розподілених і паралельних систем це поняття набуває розширеного змісту та трансформується в абстрактну модель керування виконанням залежних задач.

У моделі графа викликів реактор розглядається як логічна одиниця виконання, якій призначається підмножина вузлів графа. Вузли інтерпретуються як окремі обчислювальні операції або задачі, а ребра як відношення залежності чи причинно–наслідкового виклику. Таким чином, реактор виступає абстрактним ресурсом планування, що визначає середовище, у якому виконується відповідна група вузлів. У фізичній реалізації реактор може відповідати окремому потоку, пулу потоків або ізольованому циклу обробки подій, проте в теоретичній моделі він трактується як ресурс із власною чергою задач і обмеженою пропускною здатністю. Запровадження цієї абстракції є необхідним для формалізації розподілу

паралелізму та контролю конкуренції в розподілених системах. Відсутність глобального часу, асинхронність обміну повідомленнями, часткові відмови та варіативність затримок мережі створюють передумови для виникнення взаємоблокувань. Якщо кілька логічно залежних вузлів виконуються в одному середовищі без чітко визначеної дисципліни планування, можливе утворення циклічних очікувань. Призначення вузлів різним реакторам дозволяє структуровано розмежувати області виконання, мінімізувати локальну конкуренцію за ресурси та формалізувати умови, за яких неможливе виникнення циклічного очікування. Саме тому введення поняття реактора є методологічною передумовою розроблення методу уникнення взаємоблокувань у розподілених системах.

З формальної точки зору реактор задамо функцією R: $V \rightarrow \{r_1, r_2, \ldots, r_n\}$, де V – множина вузлів графа, а $\{r_1, \ldots, r_n\}$ – множина реакторів. Це відображення задає розбиття графа на підмножини, кожна з яких виконується в межах окремого середовища планування. На відміну від класичного розфарбовування графа, де кольори мають лише топологічне значення, реактори мають операційний сенс. Вони відповідають конкретним обчислювальним ресурсам, що впливають на часові характеристики виконання, довжину критичного шляху та можливість блокувань. Коректно вибрана стратегія відображення дозволяє обмежити максимальну кількість послідовних вузлів одного шляху, що виконуються в одному реакторі, тим самим знижуючи ризик локального накопичення залежностей. На рис. 3.1 зображено приклад з використанням реактора.

Поданий граф на рис. 3.1 є прикладом формалізованої моделі виконання задач у розподіленій системі з використанням реакторної архітектури. Він поєднує дві сутності, а саме: граф залежності (граф викликів); множину реакторів як ізольованих середовищ виконання, пов’язаних протоколом міжпроцесної взаємодії (IPC). Така структура дозволяє одночасно описувати логіку обчислень і механізм їх фізичного виконання.

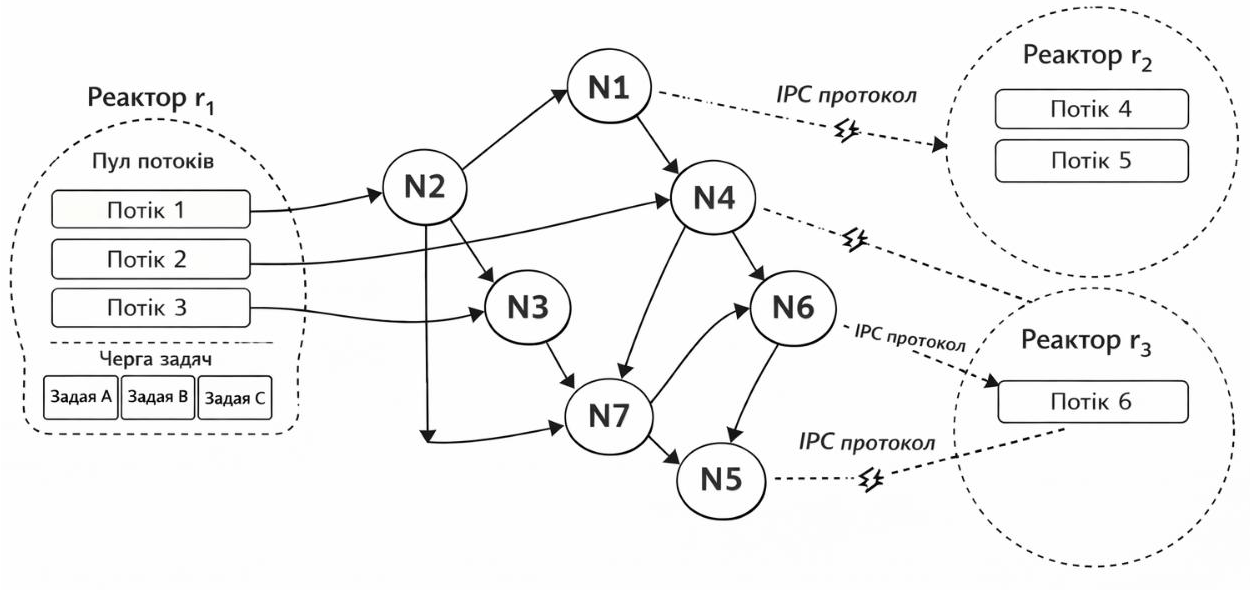


Рисунок 3.1 – Приклад графа з реактором

У центрі схеми на рис. 3.1 розташований граф викликів, що складається з вузлів N1–N7 та орієнтованих ребер між ними. Вузол інтерпретується як елементарна обчислювальна операція, сервісний виклик або транзакційний крок. Орієнтоване ребро відображає залежність типу «після виконання A може бути виконано B» або «B потребує результату A». Таким чином, граф задає частковий порядок виконання задач. Якщо розглядати його формально, то це орієнтований ациклічний граф (ОАГ), що визначає множину допустимих послідовностей виконання.

Ліва та права частини схеми відображають реактори r_1, r_2 та r_3. Кожен реактор представлений як ізольоване середовище з власним пулом потоків і чергою задач. Пул потоків визначає внутрішній рівень паралелізму, тоді як черга задач задає дисципліну планування. Реактор отримує задачі (вузли графа), обробляє їх відповідно до внутрішнього алгоритму планування та може ініціювати взаємодію з іншими реакторами через IPC–протокол.

Пунктирні стрілки між вузлами графа на рис. 3.1 та іншими реакторами позначають міжпроцесну взаємодію. Це означає, що виконання певного вузла вимагає відправлення повідомлення до іншого реактора та очікування відповіді.

Саме в цій точці логічний граф залежностей перетворюється на мережу ресурсних очікувань.

Граф може описувати блокування в розподілених системах через накладання двох структур, а саме: графа залежності; графа ресурсного очікування. Якщо вузол N4 у реакторі r_1 надсилає запит до реактора r_2 і блокується до отримання відповіді, а в той самий час у r_2 виконується вузол, який потребує результату з r_1, то виникає циклічне очікування між реакторами. Формально це означає, що в системі формується цикл у графі очікування ресурсів, навіть якщо первинний граф викликів є ациклічним.

Блокування можливе через кілька механізмів. По–перше, через обмеженість потоків у межах реактора. Якщо всі потоки зайняті задачами, що очікують відповіді від інших реакторів, то нова задача, яка могла б розірвати цикл залежностей, не буде запущена. По–друге, через синхронну модель IPC, коли реактор утримує ресурс під час очікування відповіді. По–третє, через конкуренцію за спільні логічні ресурси, які не відображені явно у графі викликів, але проявляються у виконанні.

У такій моделі виділимо два типи графів так:

1) граф причинно–наслідкових залежностей;

2) граф очікування ресурсів, який формується динамічно під час виконання.

Блокування виникає тоді, коли у графі очікування ресурсів з’являється цикл. Представлена схема демонструє, як такий цикл може виникнути між реакторами через IPC–виклики. Якщо призначення вузлів реакторам здійснюється за визначеною стратегією, наприклад, циклічним розподілом із пріоритетом критичного шляху, тоді можна формально довести, що цикл очікування не утворюється або що завжди існує хоча б один реактор, здатний продовжити виконання. Тобто реакторна декомпозиція графа дозволяє контролювати структуру можливих ресурсних конфліктів.

Отже, даний граф є формальною моделлю взаємодії обчислювальних задач і ресурсів у розподіленому середовищі. Він демонструє, як логічні залежності трансформуються у ресурсні очікування, і показує механізм виникнення взаємоблокувань через циклічні міжреакторні залежності. Саме тому така модель є

придатною для аналізу коректності протоколів міжпроцесної взаємодії та розроблення методів гарантування прогресу виконання.

Ключовим доповненням до цієї моделі є протокол міжпроцесної взаємодії, який визначає правила обміну повідомленнями між реакторами. Оскільки вузли графа, призначені різним реакторам, можуть мати залежності, то їх взаємодія відбувається через передачу подій або повідомлень. Протокол міжпроцесної взаємодії встановлює порядок ініціювання викликів, підтвердження завершення операцій, обробки помилок і тайм–аутів. У контексті уникнення взаємоблокувань важливо, щоб цей протокол виключав можливість двостороннього синхронного очікування між реакторами. Це досягається шляхом використання асинхронної моделі запит–відповідь, односторонніх повідомлень або дисципліни «неблокуючого виклику», за якої реактор не утримує ресурс під час очікування відповіді від іншого реактора.

Таким чином, реактор виступає не лише абстрактною одиницею обчислювального ресурсу, а й елементом формальної моделі, що поєднує структуру графа залежностей із протоколом міжпроцесної взаємодії. Саме інтеграція стратегії розподілу реакторів із коректно визначеним протоколом обміну повідомленнями створює основу для побудови методу уникнення взаємоблокувань. Вона дозволяє довести, що за певних умов призначення вузлів та правил взаємодії між реакторами система не утворює циклів очікування, а отже, забезпечує гарантовану можливість просування виконання критичного шляху навіть у складних розподілених середовищах.

3.2 Метрики для організації оцінювання виконання потоків в розподілених системах

Визначимо підходи до організації роботи систем із гетерогенними серверами швидкості. У дослідженні високопродуктивних обчислювальних систем особливу увагу приділяють задачі ефективного розподілу потоків між серверами з різними швидкісними характеристиками. Такі системи називають гетерогенними, оскільки

їхні обчислювальні вузли мають відмінні параметри продуктивності. Проблема оптимального розподілу ресурсів у цьому контексті може бути формалізована через два базові підходи, кожен з яких відповідає різним початковим умовам конфігурації системи. Визначимо два підходи в залежності від кількості потоків в розподілених системах.

Перший підхід до розподілу потоків визначимо з урахуванням того, що система функціонує з фіксованою кількістю потоків. Тоді, у межах першого підходу вважатимемо, що задано множину серверів із різними швидкостями обробки та загальну кількість потоків (S), які необхідно розподілити між серверами. І на основі такого початкового вибору задамо моделі. Нехай множину швидкостей серверів визначимо так:

$$M_H = \left\{m_{H,1}, m_{H,2}, \dots, m_{H,N_{M_H}}\right\}, \tag{3.1}$$

де $m_{H,i}$ – швидкість і–го сервера;

N_{M_H} – кількість серверів; $i = 1,2, \dots, N_{M_H}$.

З метою коректного порівняння продуктивності серверів проведемо нормалізацію відносно найбільшого спільного дільника для елементів множини M_H так:

$$q_H = f_H(m_{H,1}, m_{H,2}, \dots, m_{H,N_{M_H}}), \tag{3.2}$$

де f_H – функція знаходження найменшого за значенням елементу серед елементів множини M_H.

Тоді визначаємо коефіцієнти відносної швидкості серверів так:

$$b_i = \frac{m_{H,i}}{q_H}, \tag{3.3}$$

де $m_{H,i}$ – швидкість і–го сервера;

N_{M_H} – кількість серверів;

$i = 1,2, \dots, N_{M_H}$;

q_H – найменший за значенням елемент серед елементів множини M_H.

Сумарний коефіцієнт відносної швидкості серверів дає змогу оцінити частку кожного сервера в загальній продуктивності розподілених систем. Визначимо його так:

$$b = \sum_{i=1}^{N_{M_H}} b_i, \tag{3.4}$$

де b_i – коефіцієнти відносної швидкості і–го сервера;

N_{M_H} – кількість серверів;

$i = 1,2, \dots, N_{M_H}$.

Тоді частка $\frac{b_i}{b}$ буде відображати частку загальної продуктивності системи, що припадає на (i)–й сервер.

Розподіл потоків здійснимо за такою схемою:

1) первинний розподіл потоків виконуватимемо пропорційно відносним швидкостям:

$$V_i = V \cdot \frac{b_i}{b}; \tag{3.5}$$

2) оскільки використовується операція округлення, то виникає залишок потоків:

$$r = V - \sum_{i=1}^{N_{M_H}} V_i, \tag{3.6}$$

3) потрібно здійснити перевірку щодо коректності отриманого результату так:

$$r < N_{M_H}, \tag{3.7}$$

де b_i – коефіцієнти відносної швидкості і–го сервера;

N_{M_H} – кількість серверів;

$i = 1,2, \dots, N_{M_H}$.

Згідно формули (3.7) похибка округлення кожного доданка не може бути більшою за одиницю. Якщо нерівність не виконується, тоді в системі наявна проблема. Залишкові потоки розподіляються ітеративно, починаючи з серверів із найбільшою швидкістю, що мінімізує очікуваний час виконання.

Така схема дає змогу редукувати гетерогенну систему до еквівалентної однорідної моделі шляхом уявного "розщеплення" кожного сервера на b_i віртуальних підсерверів із однаковими характеристиками. Це створює аналітичну можливість застосування методів, які розроблені для однорідних систем, при збереженні максимальної заповненості та балансування навантаження.

За другим підходом розглядатимемо розподіл реакторів у системі з попередньо визначеними пулами потоків. Приймемо, що сервери мають різні швидкості і те, що кожен сервер уже оснащений пулом потоків однакового розміру. Завдання полягає у розподілі реакторів або обчислювальних вузлів між серверами таким чином, щоб зберігалася інваріантність здійсненності та враховувалися швидкісні відмінності серверів. Введемо метрики для фіксації цих дій в розподілених системах.

Аналіз заповненості системи

Нехай L – довжина критичного шляху, що вимірюється в потоках, S – загальна кількість потоків.

Якщо L= S, то система повністю заповнена і будь–яка спроба нерівномірного розподілу може порушити гарантії здійсненності.

Якщо ж L< S, то виникає резерв продуктивності, що дозволяє здійснити контрольований дисбаланс на користь швидших серверів.

Здійснимо формалізацію частот і періодів як метрик для процесів чи завдань в розподілених системах. Аналогічно до першого підходу визначимо так відносну частоту планування сервера:

$$f_i = \frac{b_i}{b}, \tag{3.8}$$

де b_i – коефіцієнти відносної швидкості і–го сервера;

N_{M_H} – кількість серверів;

$i = 1,2, \dots, N_{M_H}$;

b – загальна продуктивності розподілених систем.

Ці значення f_i відображають відносну частоту планування сервера.

Період обслуговування сервера:

$$T_i = \frac{1}{f_i}, \tag{3.9}$$

де f_i – коефіцієнти відносної швидкості і–го сервера;

N_{M_H} – кількість серверів;

$i = 1,2, \dots N_{M_H}$.

Таким чином, швидші сервери мають менші періоди та повинні плануватися частіше.

Для формування простору можливих рішень використаємо експоненціальний розподіл періодів, адаптований не до розмірів пулів, а до швидкісного розподілу. Визначимо його вектором так:

$$T = (T_1, T_2, \dots, T_{N_{M_H}}), \tag{3.10}$$

де N_{M_H} – кількість серверів;

$i = 1,2, \dots N_{M_H}$;

T_i – період обслуговування і–го сервера.

Проте виникає конфлікт оптимізаційних критеріїв в контексті максимізації продуктивності і забезпечення здійсненності.

Розглянемо забезпечення здійсненності в однорідному випадку. Оскільки всі сервери мають однаковий розмір пулу потоків, то перевірку здійсненності можна спростити. Для цього введемо мінімальний допустимий період так:

$$y = \frac{L}{\mathrm{S}} + 1, \tag{3.11}$$

де L – довжина критичного шляху, що вимірюється в потоках;

S – загальна кількість потоків.

Якщо всі періоди $T_i \geq y$, то гарантується відсутність конфліктів планування. Найменший період відповідає найшвидшому серверу. Обмеження мінімального періоду зменшує простір пошуку та дозволяє будувати лише гарантовано здійсненні розподіли.

Критерій оптимальності дає змогу перевіряти наближення поточного стану до ідеально можливого варіанту. Якість отриманого експоненціального розподілу оцінимо за ступенем наближення до ідеального швидкісного профілю $T = (T_1, T_2, \dots, T_{N_{M_H}})$ (формула (3.10)).

Чим ближчий фактичний розподіл до теоретично оптимального, за умови дотримання інваріантів здійсненності, тим вища сумарна продуктивність системи.

Обидва підходи демонструють різні стратегії адаптації до гетерогенності. Перший підхід орієнтований на пропорційний розподіл потоків за умови централізованого контролю. Другий підхід досліджує можливості контрольованого дисбалансу за умови збереження інваріантів здійсненності. Обидва підходи є основою формалізованої моделі, яка дозволяє одночасно враховувати неоднорідність обчислювальних ресурсів, мінімізувати час виконання, гарантувати коректність та стабільність функціонування системи.

Оцінювання виконання потоків у розподілених системах із гетерогенними серверами швидкості базується на системі кількісних метрик, що дозволяють формально описати продуктивність, баланс навантаження, ефективність використання ресурсів і якість обслуговування. У таких системах кожен сервер характеризується власною швидкістю обробки за параметром середньої кількості завдань або інструкцій, які сервер здатен виконати за одиницю часу. Якщо система складається з m серверів, то їхня сукупна теоретична продуктивність дорівнює сумі індивідуальних швидкостей, однак фактична продуктивність залежить від політики розподілу потоків і рівня завантаження.

Базовою метрикою є інтенсивність вхідного потоку завдань, що визначає середню кількість запитів або потоків, які надходять до системи за одиницю часу. Цей показник дозволяє визначити відповідність загальної обчислювальної потужності системи поточному навантаженню. Якщо інтенсивність надходження перевищує сумарну здатність серверів обробляти задачі, то у системі починають накопичуватися черги, що призводить до зростання затримок та зниження якості обслуговування. Тому першочерговим критерієм є забезпечення стабільності системи, за якої кожен сервер здатен обробляти закріплену за ним частину потоку без накопичення нескінченної черги.

Важливою метрикою є коефіцієнт завантаження окремого сервера. Він показує, яку частку свого потенціалу використовує вузол у конкретний момент часу. У гетерогенній системі одні сервери можуть працювати майже на межі своїх можливостей, тоді як інші залишаються частково недовантаженими. Такий дисбаланс негативно впливає на загальну ефективність. Тому при оцінюванні системи аналізують не лише середнє завантаження, а й рівномірність його розподілу між вузлами. Оптимальною вважається ситуація, коли навантаження розподіляється пропорційно швидкості серверів, тобто швидші вузли отримують більшу частку потоків.

Наступною суттєвою характеристикою є середній час перебування потоку в системі. Він охоплює як час очікування в черзі, так і безпосередній час обробки на сервері. У гетерогенних системах цей показник може істотно відрізнятися залежно

від того, на який вузол потрапляє завдання. Якщо алгоритм планування не враховує різницю швидкостей, повільні сервери можуть створювати «вузькі місця», що збільшує загальну затримку. Тому середній час обробки є одним із ключових критеріїв оцінювання ефективності політики розподілу потоків.

Пропускна здатність системи відображає кількість задач, які завершуються за певний проміжок часу. У стабільному режимі вона приблизно дорівнює інтенсивності надходження запитів. Проте у випадку перевантаження фактична пропускна здатність може знижуватися через накопичення черг і збільшення часу очікування. У гетерогенному середовищі досягнення максимальної пропускної здатності залежить від здатності балансувальника навантаження враховувати різну продуктивність серверів. Якщо потоки розподіляються без урахування цих відмінностей, частина ресурсів може використовуватися неефективно.

Для аналізу рівномірності використання ресурсів застосовують метрики дисбалансу навантаження. Вони показують, наскільки відрізняється завантаження окремих серверів від середнього по системі. Чим менша ця різниця з урахуванням продуктивності вузлів, тим ефективніше організовано розподіл потоків. У гетерогенних системах абсолютна рівність завантаження не є ціллю; натомість важливо забезпечити пропорційність між швидкістю сервера та обсягом оброблюваних ним задач.

Ще однією вагомою метрикою є ефективність використання ресурсів. Вона відображає, яку частину свого потенціалу система реально використовує для виконання корисної роботи. Якщо сервери простоюють або працюють нерівномірно, загальна ефективність знижується. У сучасних розподілених середовищах керування контейнерами та мікросервісами, таких як Kubernetes, подібні показники використовуються для автоматичного масштабування та оптимального розміщення навантаження. У системах потокової обробки даних, наприклад у Apache Kafka, метрики продуктивності застосовуються для балансування партицій між брокерами та контролю швидкості обробки повідомлень.

Окремо аналізуватимемо показник прискорення, який дозволяє порівняти продуктивність гетерогенної системи з базовою конфігурацією, наприклад із виконанням на одному сервері. Ця метрика демонструє, наскільки ефективно система використовує паралелізм. В ідеальному випадку збільшення кількості серверів приводить до пропорційного скорочення часу виконання задачі, проте в реальних умовах існують накладні витрати на координацію, обмін повідомленнями та синхронізацію, що знижує фактичний виграш.

Крім того, важливою характеристикою є масштабованість системи. Вона показує, як змінюється продуктивність при зростанні кількості потоків або при додаванні нових серверів. У добре спроєктованій системі продуктивність зростає майже лінійно до досягнення апаратних або мережевих обмежень. У гетерогенному середовищі масштабованість ускладнюється різницею в швидкості вузлів і потребує адаптивних алгоритмів балансування.

Розглядаючи підходи до організації роботи систем залежно від кількості потоків, можна виділити два принципово різні режими. У першому випадку кількість потоків не перевищує кількість серверів або є близькою до неї. Тут головною метою є мінімізація часу завершення кожного окремого завдання. Раціональним рішенням стає призначення більш складних або ресурсоємних задач на швидші сервери. У другому випадку, коли потоків значно більше, ніж серверів, формується чергова модель обслуговування. Основний акцент переноситься на підтримання стабільності системи, мінімізацію середнього часу очікування та забезпечення високої пропускної здатності. У цьому режимі критично важливим стає пропорційний розподіл навантаження відповідно до продуктивності вузлів.

Отже, система метрик оцінювання виконання потоків у гетерогенних розподілених середовищах охоплює показники навантаження, затримки, пропускної здатності, ефективності, масштабованості та рівномірності використання ресурсів. Їхній комплексний аналіз дозволяє обґрунтовано вибирати алгоритми планування, політики балансування та стратегії масштабування, що особливо важливо в умовах зростаючої складності сучасних обчислювальних інфраструктур.

3.3 Метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії

У розподілених системах міжпроцесна взаємодія завдань відбувається за відсутності спільної глобальної пам’яті та єдиного фізичного годинника, що ускладнює координацію доступу до ресурсів і створює передумови для виникнення взаємоблокувань. Взаємоблокування виникає тоді, коли множина процесів перебуває в стані циклічного очікування. Кожен процес утримує певні ресурси та одночасно очікує на ресурс, який утримує інший процес з тієї ж множини. Саме усунення можливості утворення такого циклу або гарантування його розриву лежить в основі більшості протоколів уникнення взаємоблокувань. Більшість протоколів у розподілених системах спрямовані саме на усунення умови циклічного очікування або на обмеження утримання ресурсів під час очікування інших.

Однією з найпростіших і водночас концептуально прозорих моделей уникнення взаємоблокувань є підхід глобального впорядкування ресурсів. Його сутність полягає в тому, що всім ресурсам у системі заздалегідь призначається унікальний порядок. Кожен процес, який потребує доступу до кількох ресурсів, зобов’язаний запитувати їх лише у визначеній послідовності, тобто від ресурсу з меншим порядковим номером до ресурсу з більшим. Отже, якщо процес уже захопив певний ресурс, то він може звертатися лише до тих, що мають вищий пріоритет у встановленому порядку, і не має права «повертатися назад». Така дисципліна формує односпрямовану структуру очікування. Залежності між процесами завжди орієнтовані в одному напрямку. У результаті в системі принципово неможливо утворити цикл очікування, оскільки жоден процес не може опинитися в ситуації, коли він прямо чи опосередковано очікує ресурс, що має нижчий порядок, ніж той, який уже утримує. Саме відсутність циклів у графі залежностей гарантує неможливість взаємоблокування. Інший підхід базується на впорядкуванні не ресурсів, а самих процесів за допомогою часових міток. Кожному процесу під час його створення призначається унікальна мітка, що відображає

момент його появи в системі. Відповідно, «старші» процеси мають менші часові значення, а «молодші» – більші. У протоколі молодший процес не має права очікувати ресурс, який утримує старший. У такій ситуації він примусово перезапускається. Натомість старший процес може чекати на завершення молодшого. Завдяки цьому залежності завжди узгоджені з часовим порядком. Очікування можливе лише в одному напрямку, тобто від старших до молодших або навпаки, залежно від конкретного варіанта протоколу. Аналогічну ідею реалізує підхід, де старший процес має право «витіснити» молодший, змушуючи його звільнити ресурс. В обох випадках забезпечується строгий частковий порядок між процесами, що унеможливлює формування циклічного ланцюга очікувань.

Більш формалізованим і теоретично обґрунтованим є алгоритм банкіра. Його ключовою ідеєю є поняття безпечного стану системи. Перед виділенням ресурсу система аналізує спроможність в майбутньому завершити всі процеси без входження в стан взаємоблокування. Для цього враховуються максимальні потреби кожного процесу, вже виділені ресурси та наявні вільні ресурси. Якщо існує така послідовність завершення процесів, за якої кожен із них зможе отримати необхідні ресурси й коректно завершити виконання, стан вважається безпечним. Якщо ж такої послідовності не існує, система розцінює стан як небезпечний і відмовляє у виділенні ресурсу. Таким чином, взаємоблокування запобігається на етапі планування. Водночас практичне застосування цього алгоритму в масштабних розподілених середовищах є складним через потребу в глобальній інформації про стан усіх вузлів і процесів.

У системах із передачею повідомлень значну роль відіграє характер комунікаційних операцій. Якщо взаємодія між процесами здійснюється через неблокуючі операції надсилання та отримання повідомлень, ризик взаємного блокування істотно зменшується. У такій моделі процес не зупиняє своє виконання під час передавання даних, а може продовжувати роботу, не очікуючи негайної відповіді. Концепції передбачають відкладене отримання результату. Обчислення відбувається асинхронно, а процес періодично перевіряє готовність результату без

повного блокування. Це усуває одну з ключових умов виникнення взаємоблокування: ситуацію утримання ресурсу під час очікування іншого.

У розподілених алгоритмах взаємного виключення, що використовують логічні годинники Лампорта, формується глобальний порядок подій. Кожна подія отримує логічну часову позначку, яка узгоджується між процесами відповідно до причинно–наслідкових залежностей. Запити до критичної секції впорядковуються за цими мітками, а в разі однакових значень враховується ідентифікатор процесу. Така лексикографічна впорядкованість створює єдину послідовність доступу до спільного ресурсу. Процес може увійти до критичної секції лише після отримання підтвердження від інших учасників, що гарантує відсутність циклічного очікування.

У токен–орієнтованих протоколах застосовується спеціальний маркер, який циркулює між процесами. У будь–який момент часу в системі існує лише один токен, і лише його власник має право входити до критичної секції. Передача токена відбувається за наперед визначеним алгоритмом, наприклад у межах кільцевої топології. Оскільки доступ до ресурсу повністю контролюється унікальним маркером, одночасне захоплення ресурсу кількома процесами неможливе, а циклічне очікування не виникає за умови збереження інваріанту єдиного токена.

У розподілених системах важливу роль відіграють протоколи консенсусу, які забезпечують узгоджене прийняття рішень щодо змін стану. У системах потокової обробки або в оркестраційних платформах використовується модель лідера. Усі операції фіксуються лише після схвалення вузлом–координатором, що створює централізовану точку впорядкування. Такий механізм зменшує ризик конфліктів і запобігає виникненню суперечливих станів, які потенційно могли б призвести до взаємоблокування.

Окрему групу становлять підходи, орієнтовані не на запобігання, а на виявлення взаємоблокувань. У таких системах періодично аналізується глобальний граф очікування процесів. Якщо виявляється цикл, один із процесів примусово переривається з метою розриву залежності. Подібна стратегія характерна для

транзакційних систем керування базами даних, де допустимим є відкат окремих операцій задля забезпечення загальної коректності.

Всі моделі уникнення взаємоблокувань у розподілених системах зводяться до запровадження впорядкування, тобто ресурсного, часового або централізованого, контролю безпечних станів, мінімізації блокуючих операцій або застосування механізмів глобальної координації. Фундаментальним критерієм коректності таких протоколів є відсутність циклу в структурі очікувань між процесами, що забезпечує гарантовану можливість завершення виконання кожного з них. На основі таких вимог сформулюємо суть та кроки методу уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.

Розробимо концепцію комплексного методу уникнення взаємоблокувань завдань у розподілених системах, що поєднує подієво–орієнтовану модель, метрики оцінювання, алгоритми виявлення блокувань і спеціалізований протокол міжпроцесної взаємодії.

Метод буде базуватись на інтеграції подієво–орієнтованої моделі розподіленої системи з адаптивним протоколом міжпроцесної взаємодії, який повинен забезпечити динамічне впорядкування доступу до ресурсів на основі аналізу стану подій і процесів у вузлах. Основною ідеєю є перехід від реактивного підходу, коли система лише виявляє взаємоблокування, до проактивного механізму, що прогнозує потенційно небезпечні конфігурації ще до їх фактичного виникнення.

У межах подієво–орієнтованої моделі кожен вузол розглядається як множина процесів і ресурсів, а глобальний стан системи описується через множину подій. Події поділяються на локальні, комунікаційні та ресурсні. Локальні події відображають зміну внутрішнього стану процесу, комунікаційні, тобто обмін повідомленнями, ресурсні – запити, виділення та звільнення ресурсів. Для впорядкування подій використовується механізм логічного часу, що дозволяє формувати частковий порядок причинно–наслідкових залежностей. Глобальний граф подій є основою для аналізу потенційних циклів очікування.

Кожен процес описується вектором стану, який включає ідентифікатор, набір утримуваних ресурсів, множину запитуваних ресурсів, часову мітку створення та пріоритет. Кожен вузол підтримує локальний журнал подій, що періодично узгоджується з іншими вузлами через протокол міжпроцесної взаємодії. Таким чином формується розподілений журнал причинно впорядкованих подій.

Метод передбачає систему метрик оцінювання, які використовуються для прийняття рішень щодо допуску запитів до ресурсів. До основних метрик належать інтенсивність подій у вузлі, середній час очікування ресурсу, коефіцієнт утримання ресурсів, глибина ланцюга залежностей процесу, коефіцієнт блокувального ризику та щільність графа очікування. На глобальному рівні обчислюється індекс циклічної загрози, який визначається як функція кількості взаємних очікувань, середньої довжини ланцюгів залежностей і швидкості накопичення незавершених запитів. Якщо значення цього індексу перевищує встановлений поріг, система переходить у режим превентивної координації.

Протокол міжпроцесної взаємодії складається з трьох основних фаз. Перша фаза це реєстрація наміру доступу до ресурсу. Процес перед запитом ресурсу надсилає повідомлення до координуючої групи або всіх вузлів, залежно від топології. Друга фаза це узгодження порядку. На основі логічного часу, пріоритету та поточних метрик формується глобально узгоджений порядок обслуговування. Третя фаза це підтвердження або відхилення запиту. Якщо аналіз показує ризик утворення циклу, запит відкладається або процес переводиться у стан контрольованого очікування.

Ключовим елементом методу є гібридний алгоритм виявлення блокувань. На локальному рівні кожен вузол підтримує частковий граф очікування та застосовує алгоритм пошуку циклів у межах власних процесів. На глобальному рівні використовується розподілений алгоритм агрегації графів очікування. Періодично або за тригером перевищення порогового значення метрики блокувального ризику вузли обмінюються стислими описами своїх залежностей. З отриманих фрагментів формується узагальнений граф очікування, в якому перевіряється наявність циклів.

Для зменшення накладних витрат використовується інкрементний підхід, який полягає в тому, що аналізуються лише нові або змінені залежності.

Якщо виявлено потенційний цикл, активується механізм розв’язання конфлікту. Він базується на комбінованому критерії, що враховує часову мітку процесу, його пріоритет, вартість перезапуску та внесок у системне навантаження. Один із процесів визначається як жертва та переводиться в стан контрольованого відкату. При цьому протокол гарантує узгоджене інформування всіх вузлів про зміну стану, що запобігає повторному формуванню того самого циклу.

Для мінімізації ризику блокувань в методі інтегровано принцип обмеженого утримання ресурсів. Якщо процес перевищує граничний час утримання ресурсу без прогресу, система ініціює процедуру перевірки його стану. Додатково застосовується адаптивне регулювання пріоритетів: процеси з довгим часом очікування отримують підвищений пріоритет, що зменшує ймовірність голодування.

Особливістю методу є підтримка різних моделей міжпроцесної взаємодії. У системах із передачею повідомлень використовується неблокуючий обмін із підтвердженням доставки. У системах спільної пам’яті застосовується розподілений алгоритм взаємного виключення з глобальним порядком подій. У кластерних архітектурах із лідером застосовується централізована координація рішень щодо доступу до ресурсів. Протокол є модульним і може адаптуватися до топології зірки, кільця або повнозв’язної мережі.

Формально коректність методу забезпечується інваріантом ациклічності глобального графа очікування. На відміну від класичних статичних підходів, запропонований метод є динамічним і метрик–орієнтованим, що дозволяє враховувати реальний стан навантаження та структуру подій у системі. Поєднання прогнозування, локального та глобального аналізу, а також узгодженого протоколу міжпроцесної взаємодії забезпечує зниження ймовірності взаємоблокування без істотного зростання комунікаційних накладних витрат.

Послідовність основних кроків розробленого методу уникнення взаємоблокувань завдань у розподілених системах та логічні зв’язки між ними.

Метод побудований як циклічний керуючий процес із вбудованими механізмами зворотного зв’язку.

Крок 1. Формування подієво–орієнтованої моделі системи.

На початковому етапі кожен вузол формує локальну модель процесів, ресурсів і подій. Визначаються типи подій: створення процесу; запит ресурсу; виділення ресурсу; звільнення ресурсу; передача повідомлення; завершення процесу. Для кожної події фіксується логічна часова мітка та ідентифікатор процесу. Сформована подієва модель створює інформаційну основу для обчислення метрик і побудови локального графа очікування.

Крок 2. Обчислення локальних і глобальних метрик.

На основі потоку подій у вузлі розраховуються такі метрики: середній час очікування ресурсу; інтенсивність надходження запитів; коефіцієнт утримання ресурсів; довжина ланцюга залежностей процесу; локальний індекс ризику блокування. Періодично вузли обмінюються агрегованими показниками для формування глобального індексу циклічної загрози. Метрики є похідними від подієвої моделі. Результати оцінювання використовуються для прийняття рішення щодо дозволу або відкладення запиту ресурсу в межах протоколу міжпроцесної взаємодії.

Крок 3. Реєстрація наміру доступу до ресурсу.

Перед фактичним захопленням ресурсу процес ініціює подію «запит наміру» і надсилає відповідне повідомлення іншим вузлам або координатору. У повідомленні містяться дані про ідентифікатор процесу, перелік утримуваних ресурсів, перелік запитуваних ресурсів, логічна мітка часу та локальні метрики. Передача наміру активується на основі поточного рівня ризику, визначеного метриками. Отримані повідомлення формують основу для узгодження порядку доступу.

Крок 4. Узгодження глобального порядку доступу.

Використовуючи часові мітки, пріоритети та показники навантаження, формується частковий або повний порядок обслуговування запитів. У системах із координатором рішення приймає центральний вузол. У децентралізованих

системах застосовується алгоритм розподіленого узгодження. Порядок визначається на основі отриманих запитів наміру. Після визначення порядку виконується перевірка на можливість утворення циклу очікування.

Крок 5. Локальне та глобальне виявлення потенційних циклів.

Кожен вузол підтримує локальний граф очікування. За потреби виконується розподілена агрегація часткових графів для формування глобального графа залежностей. Використовується алгоритм пошуку циклів. Якщо цикл прогнозується або вже існує, формується сигнал ризику. Граф будується з урахуванням узгодженого порядку доступу. Результат аналізу визначає, чи буде запит дозволено, відкладено або ініційовано процедуру розв’язання конфлікту.

Крок 6. Прийняття рішення щодо доступу до ресурсу.

Можливі такі варіанти: підтвердження доступу; відкладення запиту; примусовий відкат одного з процесів; коригування пріоритетів. Рішення базується на мінімізації глобального індексу ризику та збереженні ациклічності графа очікування. Вибір дії визначається результатом перевірки на циклічність. Після виконання рішення оновлюється подієвий журнал і метрики.

Крок 7. Оновлення стану та адаптація параметрів.

Після кожної завершеної операції або зміни стану перераховуються метрики, оновлюється граф очікування, коригуються порогові значення ризику. Система переходить до наступного циклу аналізу. Оновлення відображає наслідки прийнятого рішення. Цикл замикається так: нові події знову потрапляють у подієву модель, ініціюючи наступний цикл оцінювання.

Метод має циклічну адаптивну структуру:

Події →

Метрики →

Запит наміру →

Узгодження порядку →

Перевірка циклів →

Рішення →

Оновлення стану → повернення до 1.

Між кроками існують прямі функціональні зв’язки, тобто результат попереднього кроку є вхідними даними наступного, та зворотні зв’язки, тобто метрики й пороги коригуються залежно від результатів розв’язання конфліктів.

Таким чином, метод представляє собою замкнену систему керування доступом до ресурсів у розподіленому середовищі, де подієва модель забезпечує інформаційну повноту, метрики, тобто кількісну оцінку ризику, алгоритми виявлення забезпечують структурний контроль циклів, а протокол міжпроцесної взаємодії забезпечує механізм узгодженої реалізації рішень.

Таким чином, розроблений метод інтегрує подієво–орієнтовану модель розподілених систем, систему кількісних метрик, алгоритми розподіленого виявлення циклів та адаптивний протокол координації процесів. Це створює основу для побудови масштабованих і стійких до взаємоблокувань розподілених обчислювальних середовищ.

3.4 Висновки до третього розділу

Отже, запропонований підхід формує цілісну методологічну основу керування розподіленими обчисленнями, у якій реактор виступає ключовим елементом поєднання структурної моделі залежностей і механізмів міжпроцесної взаємодії. Інтеграція стратегії розподілу реакторів із формалізованим протоколом обміну повідомленнями забезпечує не лише теоретично обґрунтоване уникнення взаємоблокувань, а й практичну гарантованість просування виконання критичних шляхів у складних гетерогенних середовищах.

Комплексна система метрик створює кількісну основу для оцінювання стану системи, вибору алгоритмів планування, балансування та масштабування, а алгоритми виявлення циклів разом із адаптивним протоколом координації забезпечують структурний і процедурний контроль доступу до ресурсів.

Розроблений метод уникнення взаєблокувань завдань в розподілених системах представляє собою замкнену, узгоджену систему керування, що поєднує формальну модель, аналітичний інструментарій і механізми практичної реалізації, формуючи підґрунтя для побудови масштабованих, ефективних і стійких до взаємоблокувань розподілених обчислювальних середовищ.

4 ЕФЕКТИВНІСТЬ ТА РЕАЛІЗАЦІЯ СТІЙКИХ РОЗПОДІЛЕНИХ СИСТЕМ ДО ВЗАЄБЛОКУВАНЬ ЗАВДАНЬ НА ОСНОВІ ПРОТОКОЛУ МІЖПРОЦЕСНОЇ ВЗАЄМОДІЇ

4.1 Ефективність методу уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії

Методика оцінювання ефективності розробленого методу уникнення взаємоблокувань завдань у розподілених системах на основі протоколу міжпроцесної взаємодії повинна бути спрямованою не лише на емпіричну перевірку працездатності, а й на підтвердження теоретичних положень, покладених в основу запропонованого підходу. Її призначення полягає у встановленні причинно–наслідкових зв’язків між структурою графа залежностей, правилами розподілу реакторів, особливостями міжпроцесної взаємодії та кінцевими характеристиками функціонування системи. Такий підхід дозволяє розглядати оцінювання не як суто прикладне тестування, а як інструмент верифікації коректності моделі та перевірки її узгодженості з реальними умовами функціонування гетерогенних розподілених середовищ.

В основі методики лежить принцип комплексності, відповідно до якого дослідження охоплює одночасно функціональні, структурні та продуктивні аспекти системи. Функціональна складова спрямована на підтвердження відсутності циклів очікування за умов дотримання визначених правил взаємодії між реакторами. Це передбачає формування контрольованих сценаріїв із різним рівнем конкуренції за ресурси, моделювання потенційно конфліктних ситуацій та аналіз поведінки системи в умовах інтенсивного обміну повідомленнями. Важливо не лише зафіксувати відсутність взаємоблокувань, а й довести, що це є наслідком властивостей протоколу, а не випадковим результатом конкретної конфігурації експерименту. Саме тому дослідження повинно включати варіювання топології, кількості вузлів, інтенсивності запитів і динаміки зміни станів системи.

Структурний аспект оцінювання пов’язаний із аналізом еволюції графа залежностей під впливом розподіленого протоколу координації. Методика

передбачає фіксацію моментів виникнення потенційних конфліктів, дослідження механізмів їхнього локального врегулювання та оцінку стабільності структури взаємодії у часі. Особлива увага приділяється перевірці того, що алгоритми виявлення циклів і правила маршрутизації повідомлень забезпечують глобальну узгодженість без централізованого контролю. Таким чином, підтверджується гіпотеза про можливість формування замкненої системи керування доступом до ресурсів, у якій структурна інформація та протокол взаємодії взаємно доповнюють одне одного.

Продуктивний вимір методики орієнтований на оцінювання впливу запропонованого методу на часові характеристики виконання завдань, пропускну здатність і рівномірність використання ресурсів. У цьому контексті важливо встановити появу надмірних накладних витрат, які може створювати механізм уникнення взаємоблокувань, що нівелюють його переваги. Для цього проводимо порівняльний аналіз із альтернативними підходами, зокрема з системами, що використовують стратегії виявлення та відновлення, часові механізми або централізоване керування доступом. Порівняння здійснюватимемо в однакових умовах навантаження, що дозволить оцінити внесок протоколу міжпроцесної взаємодії у підвищення стабільності та масштабованості.

Важливою складовою методики є дослідження масштабованості, яке передбачає поступове збільшення кількості вузлів і задач із фіксацією змін у затримках, інтенсивності службового трафіку та ефективності використання обчислювальних ресурсів. Обґрунтування цього етапу полягає у перевірці узгодженості емпіричних результатів із теоретично передбачуваною складністю алгоритмів. Якщо зі зростанням розмірності системи не спостерігається критичної деградації продуктивності або лавиноподібного зростання службових повідомлень, то це підтверджує коректність закладених у модель припущень щодо розподіленої координації.

Методика також враховує динамічні фактори, характерні для сучасних обчислювальних інфраструктур, зокрема зміну навантаження, вибуття або додавання вузлів, варіативність затримок передачі даних. Дослідження в таких

умовах дозволяє оцінити адаптивність протоколу та його здатність підтримувати узгоджений стан без переходу в нестабільні режими. Це має принципове значення для підтвердження практичної придатності методу у реальних гетерогенних середовищах.

Розглянемо методику оцінювання ефективності розробленого методу уникнення взаємоблокувань завдань у розподілених системах на основі протоколу міжпроцесної взаємодії. Вона спрямована на формальне підтвердження коректності запропонованого підходу, кількісне вимірювання його впливу на продуктивність системи та дослідження масштабованості в умовах гетерогенного розподіленого середовища.

Оцінювання функціональної коректності здійснимо через визначення ймовірності виникнення взаємоблокування так:

$$P_{deadlock} = \frac{N_{deadlock}}{N_{runs}}, \tag{4.1}$$

де $P_{deadlock}$ – ймовірність виникнення взаємоблокування;

$N_{deadlock}$ – кількість експериментальних запусків, у яких було зафіксовано хоча б один цикл очікування ресурсів;

N_{runs} – загальна кількість незалежних експериментальних запусків системи.

Змінна $N_{deadlock}$ є критично важливою, оскільки відображає реальні випадки порушення прогресу виконання задач. Змінна N_{runs} забезпечує статистичну репрезентативність результатів, дозволяючи оцінити стійкість методу до варіацій навантаження та конфігурацій. Якщо занчення $P_{deadlock}$ прямує до нуля при зростанні кількості запусків, то це підтверджує коректність протоколу міжпроцесної взаємодії.

Додатково оцінимо середній час виявлення потенційного циклу очікування, який визначається як середній інтервал між моментом формування циклу залежностей та моментом його виявлення алгоритмом. Цей показник є необхідним для аналізу оперативності механізму структурного контролю.

Продуктивність системи аналізуємо через середній час виконання задач:

$$T_{exec} = \frac{\sum_{i=1}^{N_{completed}} T_i}{N_{completed}}. \quad (4.2)$$

де T_{exec} – середній час виконання задач;

T_i – фактичний час виконання і–ї задачі від моменту ініціації до завершення;

$N_{completed}$ – кількість успішно завершених задач у межах експерименту.

Змінна T_i дозволяє врахувати індивідуальні особливості виконання задач різної складності, а $N_{completed}$ характеризує фактичну результативність системи. Середній час виконання відображає вплив протоколу координації на затримки доступу до ресурсів.

Пропускну здатність визначимо так:

$$T_k = \frac{N_{\text{completed}}}{T_{total}}, \quad (4.3)$$

де T_k – кількість завершених задач за одиницю часу;

T_{total} – загальна тривалість експериментального інтервалу;

$N_{\text{completed}}$ – відображає реальний рівень корисної роботи системи в кількості задач.

Змінна T_{total} задає часову нормалізацію вимірювання, а $N_{\text{completed}}$ відображає реальний рівень корисної роботи системи.

Для оцінювання ефективності використання ресурсів вводиться коефіцієнт завантаження:

$$U = \frac{T_{base}}{T_{total}}, \quad (4.4)$$

де U – коефіцієнт використання ресурсу;

T_{base} – сумарний час, протягом якого ресурс перебував у стані активного

виконання задач;

T_{total} – загальний час спостереження.

Змінна T_{base} демонструє фактичну корисну зайнятість ресурсу, а співвідношення з T_{total} дозволяє оцінити ступінь простоїв або перевантаження.

Окремо визначимо додаткові витрати протоколу міжпроцесної взаємодії так:

$$R_v = \frac{M_{\text{protocol}}}{M_{total}}, \tag{4.5}$$

де R_v – частка службового трафіку;

M_{protocol} – кількість або обсяг службових повідомлень, що генеруються протоколом координації;

M_{total} – загальний обсяг переданих повідомлень у системі.

Змінна M_{protocol} відображає витрати на підтримання узгодженості та контроль циклів, тоді як M_{total} характеризує повне комунікаційне навантаження. Аналіз цього співвідношення дозволяє оцінити наявність потенційного перевищення службових витрат допустимого рівня.

Для комплексної інтегральної оцінки застосовується узагальнений показник ефективності:

$$E_{total} = w_1 \cdot (1 - P_{deadlock}) + w_2 \cdot \frac{T_k}{T_{k,max}} + w_3 U - w_4 R_v, \tag{4.6}$$

де E_{total} – інтегральний показник ефективності;

R_v – частка службового трафіку;

U – коефіцієнт використання ресурсу;

w_1, w_2, w_3, w_4 вагові коефіцієнти, що відображають пріоритетність відповідних критеріїв;

T_k – кількість завершених задач за одиницю часу;

$T_{k,max}$ – максимальне зафіксоване значення пропускної здатності в

контрольному експерименті.

Вагові коефіцієнти дозволяють адаптувати оцінювання до специфіки застосування системи, наприклад, надаючи більшої ваги гарантованій відсутності взаємоблокувань або мінімізації накладних витрат.

Таким чином, розширена методика забезпечує всебічний аналіз розробленого методу, дозволяє кількісно підтвердити його коректність, оцінити вплив на продуктивність і встановити межі масштабованості. Деталізація змінних у формулах гарантує прозорість інтерпретації результатів та відтворюваність експериментів, що є необхідною умовою наукової обґрунтованості дослідження.

На рис. 4.1 – рис. 4.4 зображені графіки результатів експериментального дослідження ефективності розробленого методу уникнення взаємоблокувань у розподілених системах на основі протоколу міжпроцесної взаємодії. Візуалізація охоплює чотири ключові групи показників: ймовірність виникнення взаємоблокувань; середній час виконання задач; частку протокольних додаткових витрат; пропускну здатність системи залежно від масштабування. Сукупний аналіз цих залежностей дозволяє комплексно оцінити функціональну коректність, продуктивність та масштабованість запропонованого підходу.

Графік на рис. 4.1 відображає залежність ймовірності виникнення взаємоблокування $P_{deadlock}$ від кількості незалежних запусків системи N_{runs}. Крива для запропонованого методу демонструє стрімке зниження показника на початкових етапах експерименту з подальшою стабілізацією на рівні, близькому до нуля. Така динаміка має принципове значення. Вона свідчить про те, що зі зростанням статистичної вибірки підтверджується системна властивість протоколу, яка полягає у відсутності формування циклів очікування за умов дотримання визначених правил взаємодії між реакторами. Початкові ненульові значення пояснюються перехідними процесами, що виникають у фазі ініціалізації або при моделюванні граничних сценаріїв конкуренції ресурсів. Однак подальша стабілізація кривої на мінімальному рівні підтверджує, що метод не просто зменшує частоту взаємоблокувань, а фактично запобігає їх системному виникненню.

Для порівняння, крива класичного підходу виявлення взаємоблокувань демонструє суттєво вищий рівень $P_{deadlock}$. Це пояснюється тим, що механізм виявлення та відновлення реагує вже на сформований цикл, а не запобігає його появі. Тому в певній частці експериментів система переходить у стан блокування, який потім усувається процедурою відновлення. Ще більш виражений негативний результат спостерігається для системи без механізму уникнення, де ймовірність залишається стабільно високою та майже не залежить від кількості запусків, що вказує на структурну схильність до формування циклів очікування.

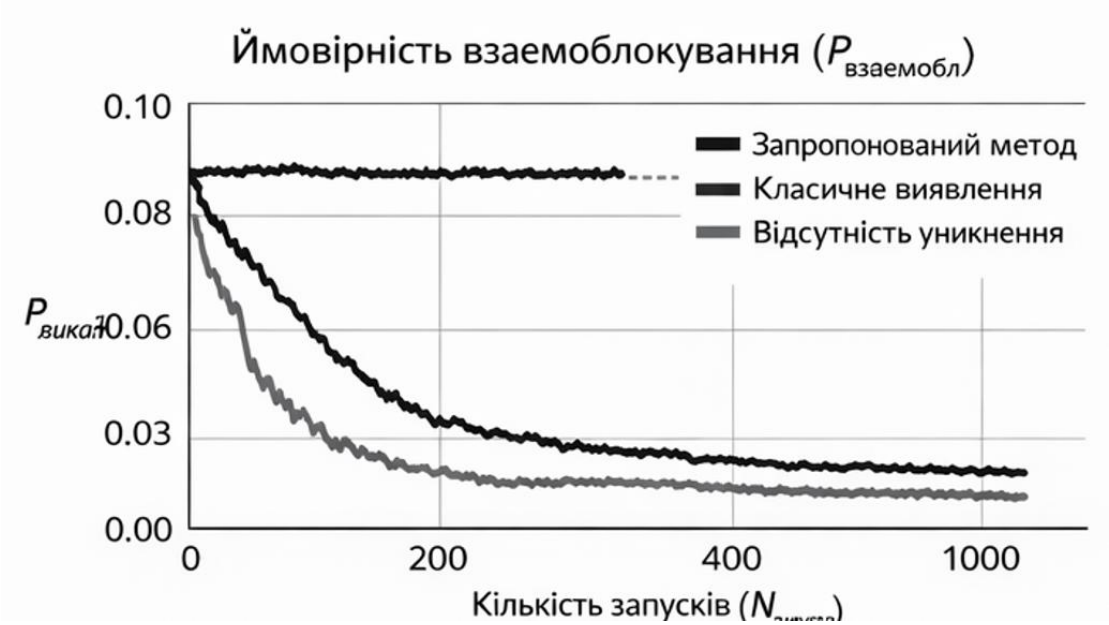


Рисунок 4.1 – Порівняльні графіки щодо ймовірності виникнення взаємоблокувань

Графік на рис. 4.2 демонструє середній час виконання задач T_{exec} за умов низького, середнього та високого навантаження. Для розробленого методу спостерігається помірне зростання часу виконання зі збільшенням навантаження, що є природним наслідком підвищеної конкуренції за ресурси. Водночас навіть за високого навантаження значення T_{exec} залишається нижчим порівняно з альтернативними підходами. Це означає, що протокол міжпроцесної взаємодії не створює надмірних синхронізаційних затримок і не призводить до каскадного накопичення очікувань.

У класичному методі виявлення взаємоблокувань середній час виконання зростає більш інтенсивно. Причиною цього є додаткові витрати на аналіз глобального стану та процедури відновлення після виявлення циклу. Система без механізму уникнення демонструє ще гірші результати, оскільки періодичні блокування спричиняють значні простої. Таким чином, другий графік підтверджує, що запропонований метод забезпечує не лише коректність, а й підвищену часову ефективність.

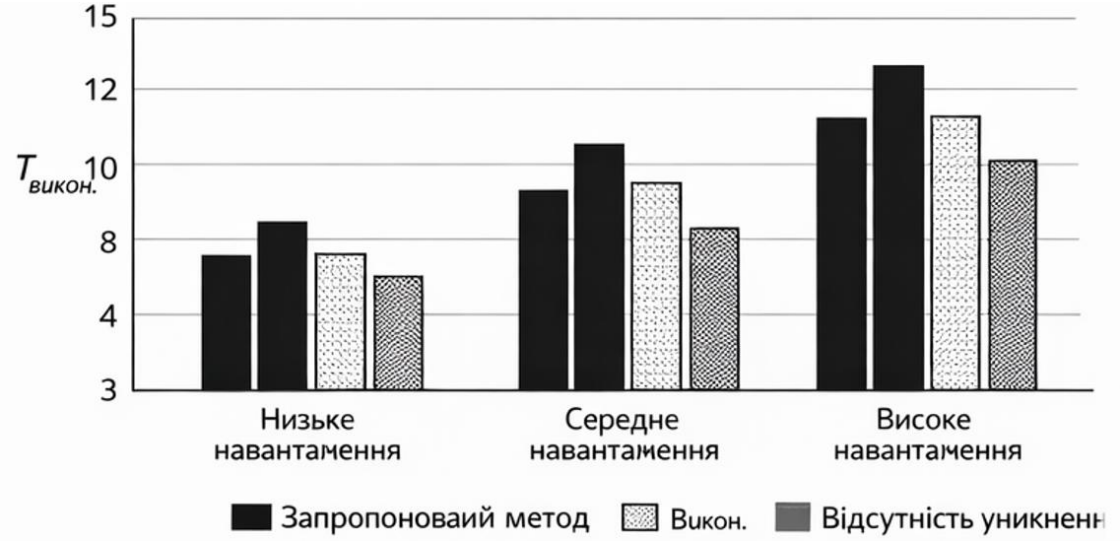


Рисунок 4.2 – Порівняльні графіки щодо середнього часу виконання завдань

Графік на рис. 4.3 зображає залежність частки протокольних додаткових витрат R_v від кількості вузлів N. Для запропонованого методу спостерігається плавне зростання додаткових витрат із подальшою стабілізацією. Це свідчить про те, що механізм координації масштабуються передбачувано та не призводить до експоненційного зростання службового трафіку. Фактично протокол забезпечує локалізацію взаємодії та мінімізує глобальні синхронізації.

Натомість централізований контроль демонструє стрімке зростання додаткових витрат із масштабуванням системи. Це пов’язано з концентрацією запитів у центральному вузлі, що створює комунікаційне перевантаження. Часова стратегія має менші додаткові витрати на початкових етапах, проте зі збільшенням кількості вузлів частота повторних запитів і перезапусків зростає, що також

призводить до накопичення службового трафіку. Отже, графік на рис. 4.3 підтверджує структурну перевагу розподіленого протоколу координації.

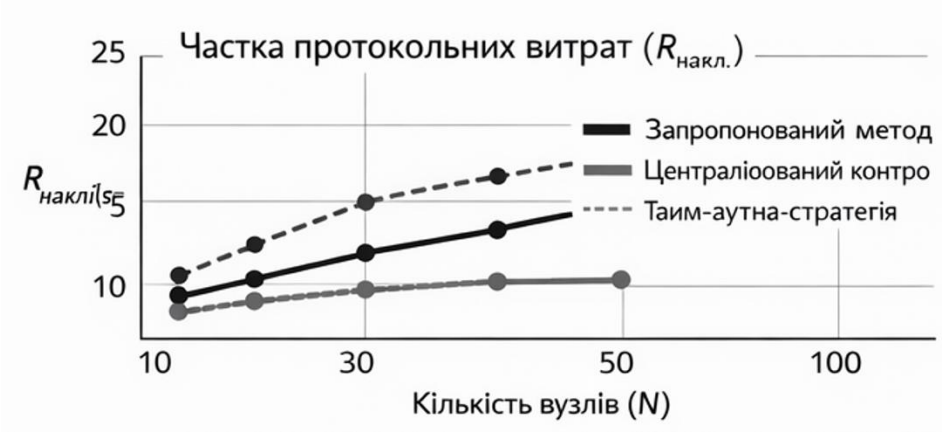


Рисунок 4.3 – Порівняльні графіки щодо частки протокольних витрат

\

Графік на рис. 4.4 відображає залежність пропускної здатності системи від кількості вузлів. Для запропонованого методу спостерігається майже лінійне зростання продуктивності зі збільшенням числа вузлів, що свідчить про ефективну масштабованість. Кожне додавання нового вузла приводить до пропорційного зростання кількості виконаних задач за одиницю часу. Це означає, що координаційні механізми не обмежують паралелізм і не створюють вузьких місць.

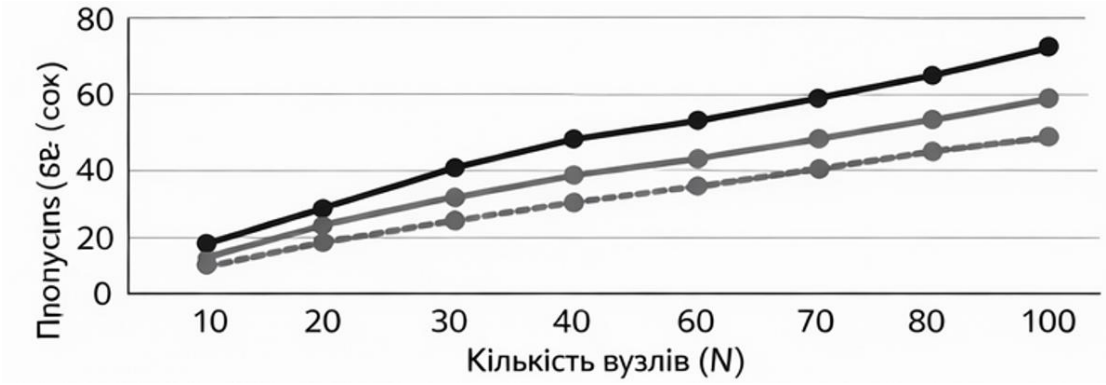


Рисунок 4.4 – Порівняльні графіки щодо пропускної здатності системи

У випадку централізованого або реактивного підходу масштабування є менш ефективним. Пропускна здатність зростає повільніше, що свідчить про

накопичення координаційних витрат або періодичні втрати продуктивності через взаємоблокування.

Підсумкова табл. 4.1 узагальнює кількісні значення основних показників. Значення $P_{deadlock}$, близьке до нуля для запропонованого методу, підтверджує функціональну коректність. Найменший середній час виконання задач та найбільша пропускна здатність демонструють ефективність з точки зору продуктивності. Водночас частка накладних витрат залишається контрольованою і значно нижчою, ніж у централізованих підходах.

Таблиця 4.1 – Загальні кількісні показники

Показник	Розроблений	Класичне виявлення	Відсутність уникнення
$P_{deadlock}$	0,002	0,056	0,081
Середній час виконання	5,2 с	8,4 с	11,5 с
Пропускна здатність	45,6	38,2	29,5
Частка витрат	6,7 %	15,4 %	20,3 %

У сукупності результати графічного та табличного аналізу підтверджують гіпотезу про те, що інтеграція подієво–орієнтованої моделі розподілу задач із формалізованим протоколом міжпроцесної взаємодії дозволяє забезпечити структурну відсутність циклів очікування без втрати масштабованості. Запропонований метод демонструє стабільність у широкому діапазоні навантажень, передбачуване зростання накладних витрат і лінійну динаміку продуктивності при розширенні системи. Це свідчить про його придатність до застосування в сучасних гетерогенних розподілених обчислювальних середовищах, де поєднуються вимоги високої надійності, масштабованості та ефективності використання ресурсів.

4.2 Концептуальна реалізація методу уникнення взаємоблокувань завдань в розподіленій системі

У сучасних розподілених обчислювальних системах проблема управління ресурсами та уникнення взаємоблокувань є однією з найважливіших задач для забезпечення надійності та ефективності виконання завдань. Глобальний планувальник у такій системі відповідає за координацію доступу до спільних ресурсів, забезпечуючи, щоб одночасні запити від кількох вузлів системи не призводили до взаємоблокувань або стану, коли система не може виконати жодне завдання через конфлікти ресурсів. Реалізація такого планувальника потребує використання алгоритмів уникнення взаємоблокувань, протоколів управління блокуванням, механізмів пріоритетного планування, а також інтеграції міжпроцесної взаємодії, щоб вузли могли повідомляти про свої запити та отримувати дозволи на використання ресурсів у реальному часі. Глобальний планувальник поєднує алгоритм банкіра для перевірки безпечного стану системи, протокол двофазного блокування (2PL) для запобігання паралельному захопленню ресурсів, пріоритетне планування завдань для визначення порядку їх виконання, Lamport Clock для синхронізації подій у розподіленій системі, розподілений алгоритм Chandy–Misra–Haas для виявлення потенційних взаємоблокувань, а також механізми динамічної міграції задач для балансування навантаження між вузлами. Усі ці компоненти інтегровані через TCP IPC, що дозволяє WorkerNode надсилати запити на виділення ресурсів до глобального планувальника та отримувати відповіді у форматі GRANT або DENY, забезпечуючи реальний приклад розподіленої взаємодії. На рис. 4.5 зображено схемою процес та розподілене середовище опрацювання завдань.

Основні елементи схеми програмного проєкту глобального планувальника з рис. 4.5:

1. WorkerNode – вузол виконавця, який надсилає запити на виділення ресурсів до GlobalScheduler через TCP. Містить інформацію про завдання, пріоритет та потреби в ресурсах.

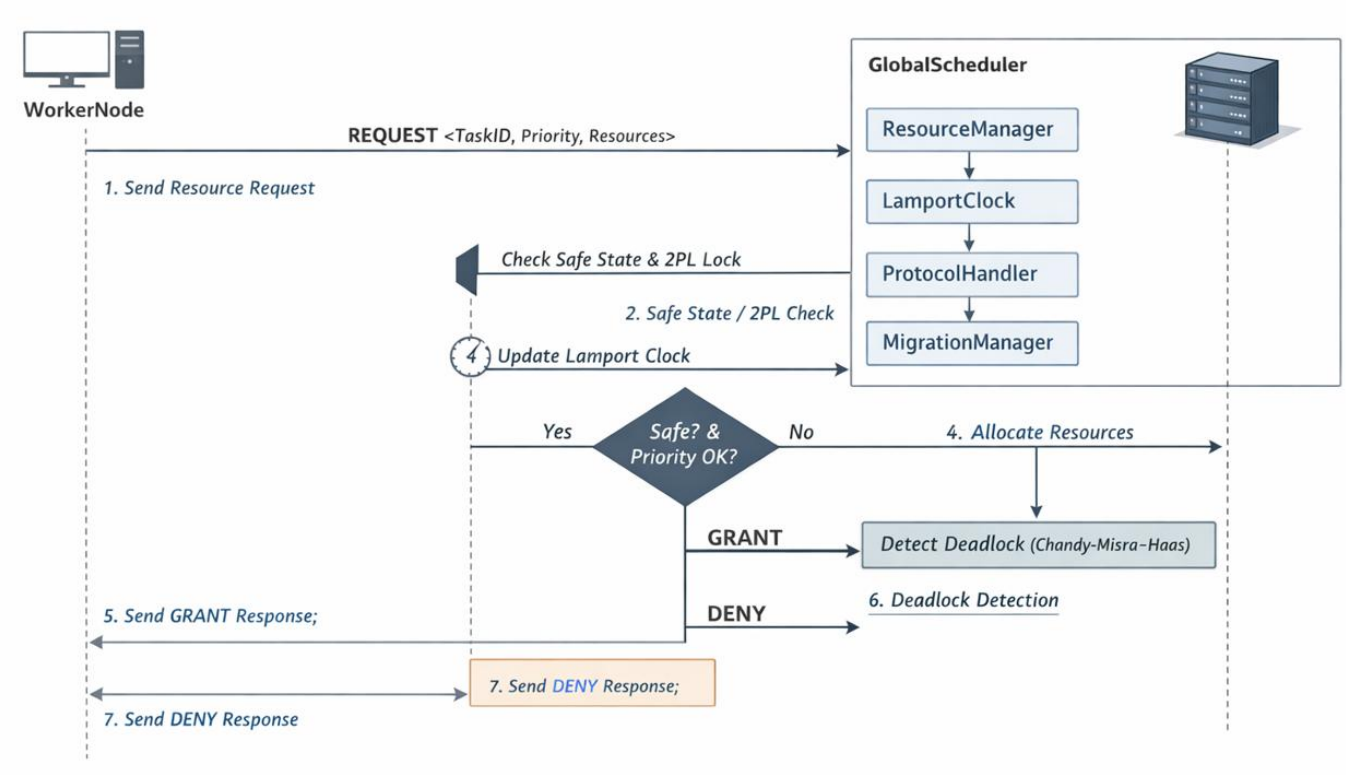


Рисунок 4.5 – Схема розподіленого середовища виконання завдань

2. GlobalScheduler – центральний планувальник, що приймає запити від WorkerNode, перевіряє їх безпечність та приймає рішення про виділення ресурсів.

Центральний планувальник містить такі підкомпоненти:

1. ResourceManager – відповідає за перевірку безпечного стану системи, виділення ресурсів за алгоритмом банкіра, реалізацію двофазного блокування (2PL) та облік наявних і зайнятих ресурсів.

2. LamportClock – векторний годинник для синхронізації подій у розподіленій системі, допомагає визначити порядок обробки запитів.

3. ProtocolHandler – реалізує розподілений алгоритм Chandy–Misra–Haas для виявлення потенційних deadlock у системі, відслідковує цикли очікування ресурсів.

4. MigrationManager – забезпечує динамічну міграцію завдань між вузлами для балансування навантаження та уникнення блокувань.

Потік запитів RESOURCE_REQUEST це дані від WorkerNode до GlobalScheduler з інформацією про завдання, ресурси та пріоритет, який перевіряється на безпечність та пріоритетність.

Відповіді GRANT / DENY це рішення GlobalScheduler щодо виділення або відмови у ресурсах, що надсилається назад WorkerNode через TCP.

Інтеракції між компонентами це запити від WorkerNode, які обробляються ResourceManager, синхронізуються через LamportClock, перевіряються пріоритети. Перевіряється можливе взаємоблокування через ProtocolHandler і, при необхідності, відбувається міграція завдань через MigrationManager.

Ці елементи разом утворюють цілісну розподілену систему управління ресурсами з гарантією безпечного стану та ефективного виконання завдань, програмний код якої подано в Додатку Г.

Архітектура системи включає кілька основних компонентів. Перший компонент ResourceManager управляє наявними ресурсами та обліком потреб кожного завдання. Він відповідає за перевірку безпечного стану системи перед виділенням ресурсів і забезпечує реалізацію алгоритму банкіра Дейкстри. ResourceManager зберігає інформацію про наявні ресурси, максимальні потреби завдань, ресурси, що вже виділені, а також залишкові потреби кожного завдання. При надходженні запиту на виділення ресурсів ResourceManager спочатку перевіряє, чи не перевищує запит залишкову потребу завдання та наявні ресурси. Після цього він тимчасово резервує ресурси та перевіряє, чи не призведе це до небезпечного стану, використовуючи алгоритм перевірки safe state. Якщо стан є безпечним, ресурси виділяються, а якщо ні, запит відхиляється, щоб уникнути deadlock. Одночасно ResourceManager реалізує двофазне блокування (2PL), яке гарантує, що ресурси спочатку блокуються перед виконанням завдання і звільняються тільки після завершення або відхилення запиту, що дозволяє уникнути конфліктів між паралельними завданнями.

Другий важливий компонент Lamport Clock використовується для відстеження часу подій у розподіленій системі. Кожен запит від WorkerNode супроводжується часовою міткою Lamport, яка дозволяє глобальному планувальнику впорядковувати запити та визначати послідовність подій у системі незалежно від фізичного часу. Lamport Clock забезпечує синхронізацію подій у

різних вузлах і допомагає уникати станів, коли одне завдання неправомірно отримує ресурси через затримку комунікації між вузлами.

Для забезпечення пріоритетного планування завдань, кожне завдання містить інформацію про свій пріоритет. Перед виділенням ресурсів глобальний планувальник вибирає з черги запитів завдання з найвищим пріоритетом. Це дозволяє системі віддавати перевагу більш критичним завданням і гарантує, що менш важливі задачі не блокуватимуть виконання ключових процесів. Пріоритетне планування інтегрується у перевірку safe state: перевірка безпечного стану проводиться спочатку для завдань з високим пріоритетом, а потім для інших.

Компонент ProtocolHandler відповідає за інтеграцію розподіленого алгоритму Chandy–Misra–Haas для виявлення потенційних deadlock у системі. Якщо запит на виділення ресурсів відхиляється, глобальний планувальник може надіслати probe message іншим вузлам системи, щоб перевірити, чи є цикл очікування, який може призвести до deadlock. Probe message містить інформацію про ініціатора запиту, завдання, яке чекає, і ресурс, який блокує виконання. Якщо probe message повертається до ініціатора, це сигналізує про наявність потенційного deadlock, що дозволяє вчасно вжити заходів, таких як перенаправлення ресурсів або міграція задач.

Динамічна міграція задач реалізується через компонент MigrationManager, який дозволяє переміщати завдання між вузлами для балансування навантаження. Це особливо важливо у розподілених системах, де деякі вузли можуть бути перевантажені, а інші залишаються вільними. MigrationManager забезпечує передачу стану завдання на інший вузол через IPC і оновлює ResourceManager, щоб відобразити зміни у ресурсах. Міграція може бути активована як у випадку перевантаження вузла, так і у разі потенційного deadlock, коли перенаправлення завдань може вирішити конфлікт ресурсів.

TCP IPC реалізується через стандартні сокети. GlobalScheduler виступає як TCP–сервер, що слухає певний порт і обробляє запити від WorkerNode. Кожен WorkerNode підключається до сервера, надсилає запит у форматі текстового повідомлення, що містить тип команди (REQUEST або RELEASE), ідентифікатор

завдання, пріоритет та кількість ресурсів для кожного типу. Сервер приймає запит, обробляє його за допомогою ResourceManager, LamportClock, 2PL та пріоритетного планування, а потім надсилає відповідь у вигляді GRANT або DENY. Обробка кількох клієнтів реалізована через багатопоточність, де для кожного підключеного WorkerNode створюється окремий потік, який відповідає за читання повідомлень та обробку запитів. Це дозволяє глобальному планувальнику одночасно обробляти декілька запитів, що імітує реальну розподілену систему.

WorkerNode реалізований як TCP–клієнт, який встановлює з'єднання з GlobalScheduler, надсилає запит на виділення ресурсів, отримує відповідь і при необхідності відправляє повідомлення про звільнення ресурсів після завершення завдання. Повідомлення формуються у простому текстовому форматі, що забезпечує легку інтеграцію та налагодження. WorkerNode може відправляти декілька запитів послідовно або паралельно, що дозволяє тестувати поведінку глобального планувальника при великій кількості одночасних запитів.

Інтеграція всіх компонентів забезпечує цілісну роботу системи. ResourceManager гарантує безпечний стан ресурсів та реалізує алгоритм банкіра разом з двофазним блокуванням. Lamport Clock відслідковує події та синхронізує запити від різних вузлів. Пріоритетне планування визначає порядок обробки завдань, а ProtocolHandler забезпечує розподілене виявлення deadlock через Chandy–Misra–Haas. MigrationManager дозволяє динамічно переміщати задачі між вузлами, забезпечуючи баланс навантаження та уникнення станів блокування. TCP IPC забезпечує реальний канал взаємодії між WorkerNode та GlobalScheduler, що дозволяє відтворити поведінку повноцінної розподіленої системи.

Така система є масштабованою. Нові WorkerNode можна підключати до GlobalScheduler у будь–який момент, і система буде обробляти їх запити у відповідності з пріоритетами та станом ресурсів. Додавання нових типів ресурсів або зміна кількості ресурсів також не потребує кардинальної перебудови архітектури, оскільки ResourceManager реалізований у вигляді універсального компонента, який працює з довільною кількістю ресурсів. Векторні часові мітки Lamport Clock гарантують коректну синхронізацію подій навіть у випадку

затримок мережі, а протокол Chandy–Misra–Haas дозволяє системі виявляти deadlock до того, як він стане критичним.

Описана система дозволяє реалізувати експериментальні сценарії для перевірки різних стратегій управління ресурсами, пріоритетного планування та динамічної міграції завдань. Наприклад, можна протестувати, як система реагує на перевантаження окремих вузлів, чи які завдання отримають ресурси першими при конфліктних запитах, а також перевірити ефективність алгоритму виявлення взаємоблокувань завдань при великій кількості одночасних запитів. Система також дозволяє вести логування подій, що корисно для аналізу продуктивності та оптимізації алгоритмів розподіленого планування.

Таким чином, інтеграція ResourceManager, LamportClock, пріоритетного планування, двофазного блокування, Chandy–Misra–Haas, динамічної міграції задач та TCP IPC забезпечує повноцінну розподілену систему, яка здатна ефективно координувати виконання завдань, уникати deadlock, балансувати навантаження між вузлами та підтримувати реальний канал взаємодії між планувальником та виконуючими вузлами. Така система може бути використана як модель для досліджень, експериментальної перевірки алгоритмів управління ресурсами у розподілених середовищах, а також як основа для подальшої розробки реальних розподілених систем з високими вимогами до надійності та ефективності.

Інтегрована модель надає повний цикл управління завданнями, тобто від надходження запиту через TCP IPC до виділення ресурсів за алгоритмом банкіра, синхронізації подій за допомогою Lamport Clock, контролю блокування через 2PL, обробки пріоритетів, виявлення потенційного deadlock через Chandy–Misra–Haas та можливості динамічної міграції завдань. Такий підхід дозволяє моделювати реальні розподілені системи та проводити експерименти з різними стратегіями планування і управління ресурсами.

Завдяки архітектурі, побудованій на модульних компонентах, можлива подальша розширюваність системи. Наприклад, можна додати механізми відмовостійкості, коли GlobalScheduler буде дубльований на декілька вузлів з синхронізацією стану, або реалізувати складніші стратегії пріоритетів, що

враховують історію виконання завдань. Крім того, TCP IPC можна замінити на високопродуктивні бібліотеки для міжпроцесного зв’язку, такі як ZeroMQ або gRPC, що дозволить масштабувати систему до великої кількості вузлів у кластері.

Описана модель демонструє, як поєднання класичних алгоритмів управління ресурсами, протоколів блокування, синхронізації часу, пріоритетного планування та розподіленої взаємодії через TCP IPC дозволяє побудувати цілісну, гнучку і надійну систему управління завданнями у розподіленому середовищі.

4.3 Висновки до четвертого розділу

Результати аналізу підтверджують, що інтеграція подієво–орієнтованої моделі розподілу задач із формалізованим протоколом міжпроцесної взаємодії забезпечує відсутність циклів очікування без втрати масштабованості та демонструє стабільність при різних навантаженнях, передбачуване зростання накладних витрат і лінійну динаміку продуктивності при розширенні системи. Використання ResourceManager, LamportClock, пріоритетного планування, двофазного блокування, Chandy–Misra–Haas, динамічної міграції задач та TCP IPC дозволяє ефективно координувати виконання завдань, балансувати навантаження між вузлами, уникати deadlock і підтримувати реальний канал взаємодії між планувальником та виконавчими вузлами. Архітектура забезпечує повний цикл управління завданнями, тобто від надходження запиту до виділення ресурсів, синхронізації подій, контролю блокувань, обробки пріоритетів і динамічної міграції задач, що дозволяє моделювати реальні розподілені системи та експериментувати з різними стратегіями управління ресурсами. Завдяки модульній структурі система є гнучкою та розширюваною, з можливістю додавання відмовостійких механізмів, складніших стратегій пріоритетів та високопродуктивних протоколів міжпроцесного зв’язку для масштабування на великі кластери, що робить її придатною як для дослідницьких експериментів, так і для реального використання в сучасних гетерогенних розподілених обчислювальних середовищах.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень розроблено новий метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії та отримано такі результати.

1. Проаналізовано відомі методи та засоби уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.

2. Розроблено метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії. Це дало можливість забезпечити структурну відсутність циклів очікування, передбачуване зростання накладних витрат і лінійну динаміку продуктивності при розширенні системи, що підвищує ефективність використання ресурсів і робить систему стабільною під різними навантаженнями.

3. Розроблено архітектуру та компоненти розподілених систем, включно з реалізованим методом уникнення взаємоблокувань. Це забезпечило повноцінну координацію виконання завдань, балансування навантаження між вузлами, підтримку реального каналу взаємодії між планувальником та виконавчими вузлами, а також можливість моделювати і тестувати різні стратегії управління ресурсами в гетерогенних середовищах.

4. Досліджено метод уникнення взаємоблокувань в розподілених системах та провести з ним експериментальні дослідження. Це дозволило перевірити практичну придатність розробленого методу, підтвердити його стабільність та ефективність у широкому діапазоні навантажень, а також продемонструвати його можливість масштабування, що робить систему придатною для подальшого застосування як експериментальної платформи і основи для реальних розподілених систем з високими вимогами до надійності та ефективності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Alzalab E., Abubakar U. E. H., Li Z. El–Meligy M., El–Sherbeeny A. Modeling of fault recovery and repair for automated manufacturing cells with load–sharing redundant elements using Petri nets. *Processes*. 2023, 11, 1501.

2. Zhou Q., Chai B., Ran K., Guo Y., Zhou S., Wu W., Wang K., Ni Y. Research on a Multi–Dimensional Information Fusion Mechanical Wear Fault–Diagnosis Algorithm Based on Data Regeneration. *Sensors*. 2025, 25, 3745.

3. Chuang W., Tseng C., Tan K., Pan Y. Design of a novel transition–based deadlock recovery policy for flexible manufacturing systems. *Processes.* 2025, 13, 1610.

4. Chen C., Hu H. Extended Place–Invariant Control in Automated Manufacturing Systems Using Petri Nets. *IEEE Trans. Syst. Man Cybern. Syst*. 2020, 52, 1807–1822.

5. Kaid H., Al–Ahmari A., Li Z., Davidrajuh R. Single controller–based colored petri nets for deadlock control in automated manufacturing systems. *Processes.* 2020, 8, 21.

6. Feng Y., Ren S., Cao Y., Xing K., Yang Y. Deadlock control for flexible assembly systems with multiple resource requirements and separately–loaded parts. *IEEE Trans. Autom. Sci. Eng.* 2024, 22, 9275–9284.

7. Hu S., Li Z., Zhang Z. Design of online supervisors for enforcing diagnosability in Petri nets with unknown initial markings. *IEEE Internet Things J.* 2025, 12, 11108–11120.

8. Lin X., Zhang Y., Gao Y., Chi X. Automatic construction of Petri net models for computational simulations of molecular inter–action network. *Syst. Biol. Appl*. 2024, 10, 131.

9. Hong J, Liu F., Ma Y., Chen X., Wang Y. Composite failure analysis of an aero-engine inter-shaft bearing inner ring. *Eng. Fail. Anal.* 2024, 165, 108707.

10. Xing K., Wang F., Zhou M., Lei H., Luo J. Deadlock characterization and control of flexible assembly systems with Petri nets. *Automatica.* 2018. № 87. P. 358–364.

11. Khushbash S., Hameed A., Mumtaz A., Ali Khan H., Shahzad A. Investigation of temperature assisted corrosion failure of an aircraft turbine blade. *Engineering Failure Analysis*. 2025. Vol. 167. Art. 108909.

12. Cong X., Gu C., Uzam M., Chen Y., Al–Ahmari A.M., Wu N., Zhou M., Li Z. Design of Optimal Petri Net Supervisors for Flexible Manufacturing Systems via Weighted Inhibitor Arcs. *Asian J. Control*. 2017, 20, 511–530.

13. Gan C., Ding S., Qiu T., Liu P., Ma Q. Model-based safety analysis with time resolution (MBSA-TR) method for complex aerothermal–mechanical systems of aero-engines. *Reliability Engineering & System Safety*. 2024. Vol. 243. Art. 109864.

14. Yang P., Li C., Bin G., Miao H., Gu F. Erosion wear patterns of turboshaft engine compressor under different radial inlet distortion conditions. *Tribology International*. 2024. Vol. 198. Art. 109907.

15. Sun Y., Wang W. Role of image feature enhancement in intelligent fault diagnosis for mechanical equipment: A review. *Engineering Failure Analysis*. 2024. Vol. 156. Art. 107815.

16. Bedratyuk L., Savenko O., *MATCH Commun. Math. Comput. Chem*. 2018, 79, 407–414. URL: https://match.pmf.kg.ac.rs/electronic_versions/Match79/n2/match79n2_407–414.pdf (дата звернення 23.03.2026)

17. Shao T., Zhang L., Wang S., Wu T., Wang Q., Han C. Fully unsupervised wear anomaly assessment of aero-bearings enhanced by multi-representation learning of deep features. *Tribology International*. 2024. Vol. 196. Art. 109724.

18. Zhang Y., Bin G., Xu Y., Pan Y., Li C. Study on vibration suppression of an aero-engine rotor system with adjustable oil film thickness driven by piezoelectric actuator through critical speed . *Tribology International.* 2024. Vol. 193. Art. 109447.

19. Li G., Duan H., Pan L., Zhan S., Liu Z., Cheng B., Jia D. Tribological properties of silver coatings prepared by low temperature pulsed DC magnetron sputtering and electroplating for aero-engines fasteners. *Tribology International*. 2024. Vol. 193. Art. 109431.

20. Wang Z., Luo Q., Chen H., Zhao J., Yao L., Zhang J., Chu F. A high-accuracy intelligent fault diagnosis method for aero-engine bearings with limited samples. *Computers in Industry*. 2024. Vol. 159–160. Art. 104099.

21. Liao Z., Zhan K., Zhao H., Deng Y., Geng J., Chen X., Song Z. Addressing class-imbalanced learning in real-time aero-engine gas-path fault diagnosis via feature filtering and mapping. *Reliability Engineering & System Safety*. 2024. Vol. 249. Art. 110189.

22. Guan X., Wu W., Ni Y. A Novel Methodology to Deadlock Analysis and Avoidance for Automatic Control Systems Based on Petri Net. *Processes*. 2025. 13(10):3368. https://doi.org/10.3390/pr13103368

23. Giebas D, Wojszczyk R. Deadlocks Detection in Multithreaded Applications Based on Source Code Analysis. *Applied Sciences*. 2020; 10(2):532. https://doi.org/10.3390/app10020532

24. Silva B.D., Happi A.W., Braeken A. Touhafi A. Evaluation of Classical Machine Learning Techniques towards Urban Sound Recognitionon Embedded Systems. *Appl. Sci*. 2019, 9, 3885.

25. Zhao B., Wu Q., Zhao K., Mo Z., Zhang Z., Zhang X. MNHP-GAE: A novel manipulator intelligent health state diagnosis method in highly imbalanced scenarios. *IEEE Internet of Things Journal*. 2024. Vol. 11. P. 24073–24082.

26. Lefebvre D. Dynamical Scheduling and Robust Control in Uncertain Environments with Petri Nets for DESs. *Processes.* 2017. 5, 54.

27. Zhao B., Wu Q., Zhao K., Li J., Zhang Z., Shao H. A novel cross-receptive field fusion cascade network with adaptive mask update for transfer health state diagnosis of manipulators. *Mechanical Systems and Signal Processing*. 2025. Vol. 224. Art. 111976.

28. Hou Z., Wang H., Yue Y., Xiong M., Zhang W. A novel framework based on two-stage multi-view feature optimization and improved support vector data description for aeroengine bearing early fault detection. *Reliability Engineering & System Safety*. 2024. Vol. 245. Art. 110027.

29. Giebas D., Wojszczyk R. Multithreaded Application Model. *In Proceedings of the 16th International Conference: Distributed Computing and Artificial Intelligence, Avila, Spain, 26–28 June 2019; Springer: Berlin/Heidelberg, Germany, 2019*. Volume 1004, pp. 93–103.

30. Qian M., Jiang B., Sun C., Shi J., Bo C. Reinforcement Learning-Based Fault Tolerant Control Design for Aero-Engines with Multiple Types of Faults. *IEEE Transactions on Circuits and Systems I: Regular Papers*. 2024. Vol. 71. P. 5770–5779.

31. Li B., Xue S. K., Fu Y. H., Tang Y. D., Zhao Y. P. Kernel adapted extreme learning machine for cross-domain fault diagnosis of aero-engines. *Aerospace Science and Technology*. 2024. Vol. 146. Art. 108970.

32. Shi H., Cao S., Zuo H., Ma J., Lin C. Deep subdomain adversarial network with self-supervised learning for aero-engine high speed bearing fault diagnosis with unknown working conditions. *Measurement.* 2025. Vol. 241. Art. 115668.

33. Li K., Deng Q., Zhang L., Fan Q., Gong G., Ding S. An effective MCTS-based algorithm for minimizing makespan in dynamic flexible job shop scheduling problem. *Computers & Industrial Engineering*. 2021. Vol. 155. Article 107157.

34. Li J., Liu X., Jiang L., Liu B., Yang Z., Hu X. An Intelligent Deadlock Locating Scheme for Multithreaded Programs. *In Proceedings of the 2019 3rd International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence, Male, Maldives,* 23–24 March 2019. Pp. 14–18.

35. Tongping L., Zhou J., Silvestro S., Liu H. Defeating Deadlocks in Production Software. *U.S. Patent App*. 16/159,234, 18 April 2019.

36. Zheng W. Research trend of large–scale supercomputers and applications from the Top500 and Gordon Bell Prize. *Sci. China Inf. Sci.* 2020, 63, 128–141.

37. Wu W., Dai T., Chen Z., Huang X., Xiao J., Ma F., Ouyang R. Adaptive Patch Contrast for Weakly Supervised Semantic Segmentation *Engineering Applications of Artificial Intelligence.* 2025. Vol. 139. Art. 109626.

38. De Sensi D., di Girolamo S., McMahon K.H., Roweth D., Hoefle T. An in–depth analysis of the slingshot interconnect. In Proceedings of the International

Conference for High Performance Computing, Networking, Storage and Analysis, Atlanta, *GA, USA*, 9–19 November 2020; pp. 1–14.

39. Birrittella M.S., Debbage M., Huggahalli R., Kunz J., Lovett T., Rimmer T., Underwood K.D., Zak R.C. Enabling Scalable High–Performance Systems with the Intel Omni–Path Architecture. IEEE Micro 2019, 36, 38–47.

40. Wu W., Dai T., Chen Z., Huang X., Ma F., Xiao J. Generative Prompt Controlled Diffusion for weakly supervised semantic segmentation. *Neurocomputing*. 2025. Vol. 638. Art. 130103.

41. Ding J., Wang Y., Qin Y., Tang B. Multi-objective optimal deep deconvolution and its application to early fault signal enhancement of rotating machineries. *Mechanical Systems and Signal Processing*. 2024. Vol. 221. Art. 111722.

42. Ghaleb M., Taghipour S. Dynamic shop-floor scheduling using real-time information—A case study from the thermoplastic industry. *Journal of Manufacturing Systems*. 2023. Vol. 69. P. 271–286.

43. Dongarra J. Report on the Sunway TaihuLight System. http://www.netlib.org/utk/people/JackDongarra/PAPERS/sunwayreport–2016.pdf

44. Wang R., Lu K., Chen J., Zhang W., Li J., Yuan Y., Lu P., Huang L., Li S., Fan X. Brief Introduction of TianHe exascale Prototype System. *Tsinghua Sci. Technol*. 2021, 26, 361–369.

45. Li Y., Li Y., Cheng J., Wu P. Order Assignment and Scheduling for Personal Protective Equipment Production During the Outbreak of Epidemics. *IEEE Transactions on Engineering Management*. 2022. Vol. 69. P. 1285–1298.

46. Pang Z., Xie M., Zhang J., Zheng Y., Wang G., Dong D., Suo G. The TH Express high–performance interconnect networks. *Front. Comput. Sci*. 2014. 8, 357–366.

47. Wang S., Lian G., Cheng C., Chen H. A novel method of rolling bearings fault diagnosis based on singular spectrum decomposition and optimized stochastic configuration network. *Neurocomputing*. 2024. Vol. 574. Art. 127278.

48. Technology Cloud Reports. The Past, Present, and Future of Top500 HPC [EB/OL]. https://36kr.com/p/1300381427843716

49. Stunkel C.B., Graham, R.L., Shainer G., Kagan M., Sharkawi S.S., Rosenburg B., Chochia G.A. The high–speed networks of the Summit and Sierra supercomputers. *IBM J. Res. Dev.* 2020, 64, 1–10.

50. Lu K., Wang R., Dong Y., Zhang W., Yang B., Lu P., Zhang W., Wu H. Challenges and opportunities in the development of exascale high performance computer systems. In Development Report of 2019–2020 China Computer Science and Technology; CCF: *Beijing, China*. 2020. Pp. 418–437.

51. Krause D. Supercomputing Support. JUWELS: Modular Tier–0/1 Supercomputer at the Jülich Supercomputing Centre. *J. Large–Scale Res. Facil*. 2019, 5, A135.

52. Faanes G., Bataineh A., Roweth D., Court T., Froese E., Alverson B., Johnson T., Kopnisck J., Higgins M., Reinhard J. Cray Cascade: A Scalable HPC System Based on a Dragonfly Network. *In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, USA, 10–16 November 2012.*

53. Akarvardar K., Wong H. S. P. Technology prospects for data-intensive computing. *Proceedings of the IEEE*. 2023. № *111*(1). Pp. 92–112.

54. Murphy P. Cornelis networks omni–path: Purpose built high–performance fabrics for HPC/HPDA/AI. *In Proceedings of the Supercomputing Frontiers Europe, Warszawa, Poland, 19–23 July 2021*.

55. Yao Y., Feng J., Zhang H., Xing Y. Adaptive neighborhood-perceived contrastive network for early stage fault diagnosis of rolling bearing with limited labeled data. *Engineering Applications of Artificial Intelligence*. 2024. Vol. 137. Art. 109063.

56. Feng J., Yao Y., Lu S., Liu Y. Domain knowledge-based deep-broad learning framework for fault diagnosis. *IEEE Transactions on Industrial Electronics*. 2020. Vol. 68. P. 3454–3464.

57. Chai B., Zheng Q., Nie X., Jia J., Pan G. Domain Adaptation for Infrared-Radar Cross-Scene Multimodal Detection. *Proceedings of the 2024 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE International*

Conference on Robotics, Automation and Mechatronics (RAM). Hangzhou, China, 8–11 August 2024. New York: IEEE, 2024. P. 386–391.

58. Huang Z., Shao J., Zhu J., Zhang W., Li X. Tool wear condition monitoring across machining processes based on feature transfer by deep adversarial domain confusion network. *Journal of Intelligent Manufacturing*. 2023. Vol. 35. P. 1079–1105.

59. Ren J., Zhao B., Quan L., Lan Y., Li Y., Hao Y. Research on proportional valve spool wear diagnostic method hybrid-driven by the valve port energy loss mechanism model and data. *Mechanical Systems and Signal Processing*. 2024. Vol. 219. Art. 111606.

60. Tan G., Xue W., Zhai J., Shao E., Liu E., Zhang Y., Qian D. Status and trends of high performance computing. In Development Report of 2018–2019 China Computer Science and Technology; CCF: *Beijing, China*, 2019. Pp. 75–102.

61. Gao J.G., Zheng F., Qi F.B., Ding Y., Li H., Lu H., He W., Wei H., Jin L., Liu X. et al. Sunway supercomputer architecture towards exascale computing: Analysis and practice. *Sci. China Inf. Sci*. 2021, 64, 177–197.

62. Fu H., Liao J., Yang J., Wang L., Song Z., Huang X., Yang C., Xue W., Liu F., Qiao F. et al. The Sunway TaihuLight supercomputer: System and applications. *Sci. China Inf. Sci*. 2016. 59, 072001.

63. Liao X., Lu K., Yang C., Li J., Yuan Y., Lain M., Huang L., Lu P., Fang J., Ren J. et al. Moving from exascale to zettascale computing: Challenges and techniques. *Front. Inf. Technol. Electron. Eng*. 2018. 19, 1236–1244.

64. Lu Y. Paving the way for China exascale computing. *CCF Trans. High Perform. Comput.* 2019. 1, 63–72.

65. Hennessy J.L., Patterson D.A. A New Golden Age for Computer Architecture. Commun. *ACM*. 2019. 62, 48–60.

66. Asch M., Moore T., Badia R., Beck M., Beckman P., Bidot T., Bodin F., Cappello F., Shoudhary A., de Supinski B. et al. Big data and extreme–scale computing: Pathways to convergence—Toward a shaping strategy for a future software and data ecosystem for scientific inquiry. *Int. J. High Perform. Comput*. 2018. 32, 435–479.

67. Hennessy J. The End of the four eras of computer architecture and the rise of the fifth Era. *Commun. Chin. Comput. Fed*. 2021. 17, 38–44.

68. Yao Y., Feng J., Liu Y. Domain knowledge-guided contrastive learning framework based on complementary views for fault diagnosis with limited labeled data. *IEEE Transactions on Industrial Informatics*. 2024. Vol. 20. P. 8055–8063.

69. Nathan T., Torsten W. OIF Next Generation Interconnect Framework; OIF–FD–Client–400G/1T–01.0; OIF: Fremont, *CA*, *USA*. 2013. URL:https://www.yumpu.com/en/document/view/51079392/oif–next–generation–interconnect–framework (дата звернення: 12.04.2026)

70. Sun L., Li M., Wu H., Zhou P., Huang S., Zhang L., Pan Q., Li J., Zhang Z. Frontiers and Trends of Microelectronics in Post Moore Era. *Sci. Found. China*. 2020, 34, 652–659.

71. Vishwanatha Rao A. N. et al. Challenges in Engine Health Monitoring Instrumentation During Developmental Testing of Gas Turbine Engines. National Aerospace Propulsion Conference. Singapore: *Springer*, 2021. P. 275–295.

72. Wang C., Sun Y., Wang X. Image deep learning in fault diagnosis of mechanical equipment. *Journal of Intelligent Manufacturing*. 2023. Vol. 35. P. 2475–2515.

73. Kumar S., Ganga D. Classification of Rolling Bearing Fault Based on Long Short Term Memory Neural Network. *Proceedings of the 2nd International Conference for Innovation in Technology (INOCON).* Bangalore, India, 3–5 March 2023. P. 1–5.

74. Geurtsen M., Adan J., Akçay A. Integrated maintenance and production scheduling for unrelated parallel machines with setup times. *Flexible Services and Manufacturing Journal*. 2024. Vol. 36. P. 1046–1079.

75. Stephen. RAMCloud: Scalable High–Performance Storage Entirely in DRAM. Diego Ongaro Mentel Rosenblum. Available online: http://www.cs.uci.edu./bin.pdf/seminarseries2011/RAMCloud–Irvine.pdf (accessed on 25 March 2026).

76. Liu S. DARPA: A global innovation differentiator. *IEEE Eng. Manag. Rev*. 2020, 48, 65–71.

77. Uzun İ. Damage Detection in Aircraft Engine Borescope Inspection Using Deep Learning: *PhD Thesis. Aksaray*, 2023.

78. Chuang W–Y., Tseng C–Y., Tan K–H., Pan Y–L. Design of a Novel Transition–Based Deadlock Recovery Policy for Flexible Manufacturing Systems. *Processes*. 2025; 13(5):1610. https://doi.org/10.3390/pr13051610

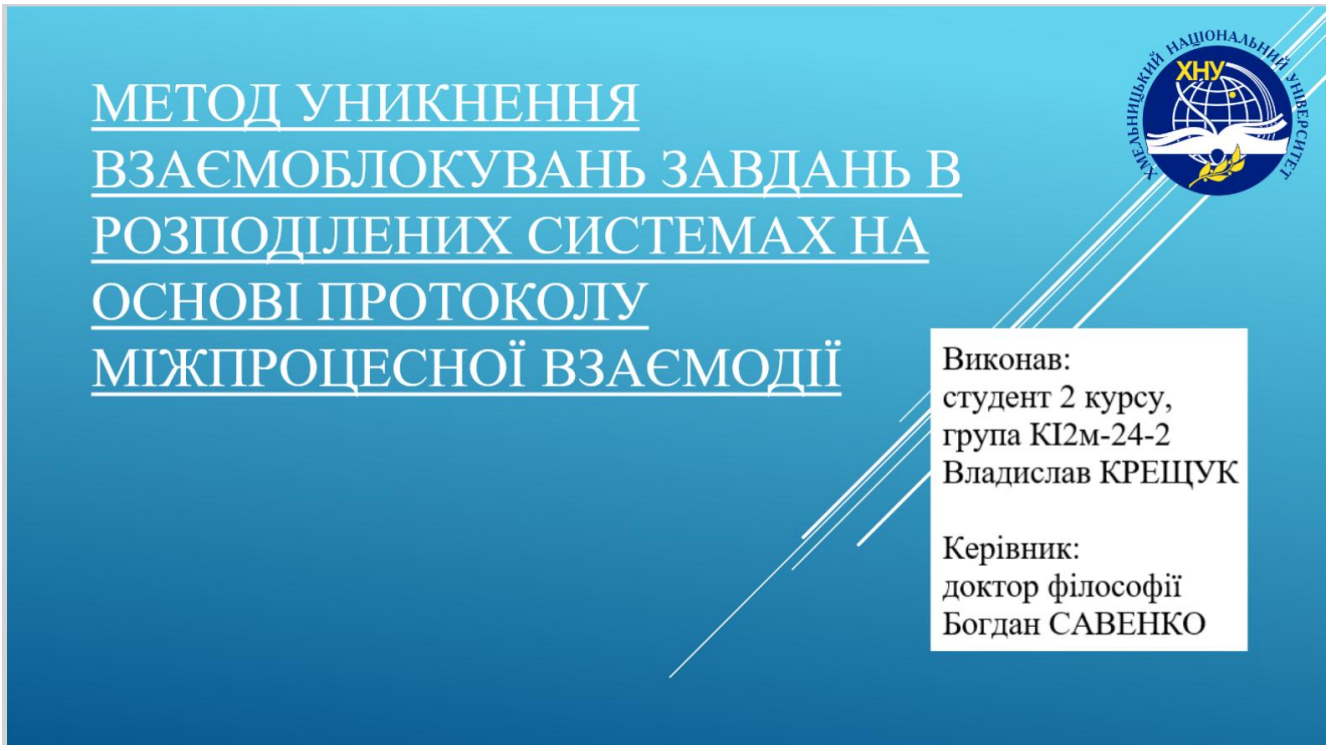
79. Tseng C.–Y., Chen J.–C., Pan Y.–L. An Improved Deadlock Recovery Policy of Flexible Manufacturing Systems Based on Resource Flow Graphs. *IEEE Access* 2024, 12, 65202–65212.

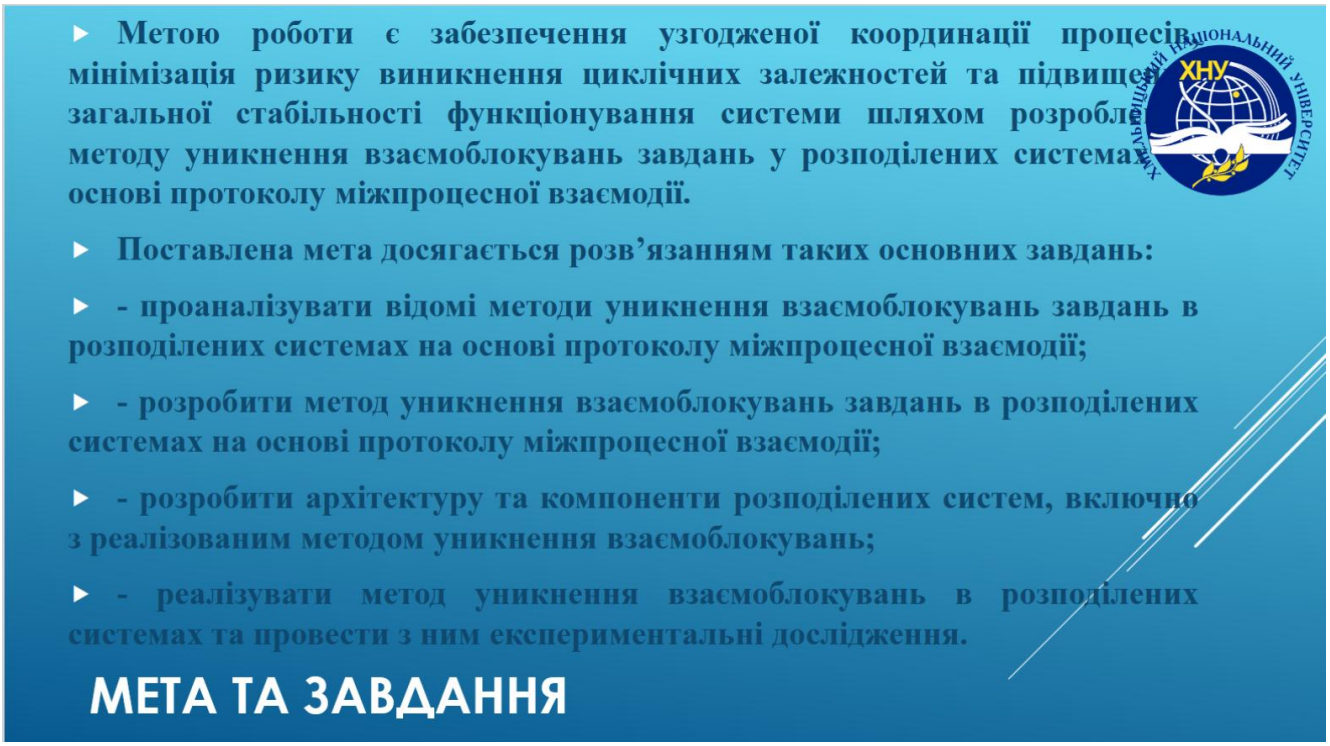
80. Sturdee M. A Step Toward Formalising Visual Data Analysis Practices in Human Computer Interaction. *Interact. Comput*. 2025. 1–14.

81. Крещук В., Лигун О., Сорочинський О. Метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії. *Вимірювальна та обчислювальна техніка в технологічних процесах*. 2026. № 1. С. 294–307. DOI: https://doi.org/10.31891/2219–9365–2026–85–37

ДОДАТОК А
(обов'язковий)

Презентація до роботи





- **Об'єктом дослідження є процес уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.**
- **Предметом дослідження є методи та засоби уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.**
- **Наукова новизна отриманих результатів:**
- **- розроблено новий метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії, особливістю якого є включення глобального контролера в архітектуру розподілених систем.**

ОБ'ЄКТ, ПРЕДМЕТ

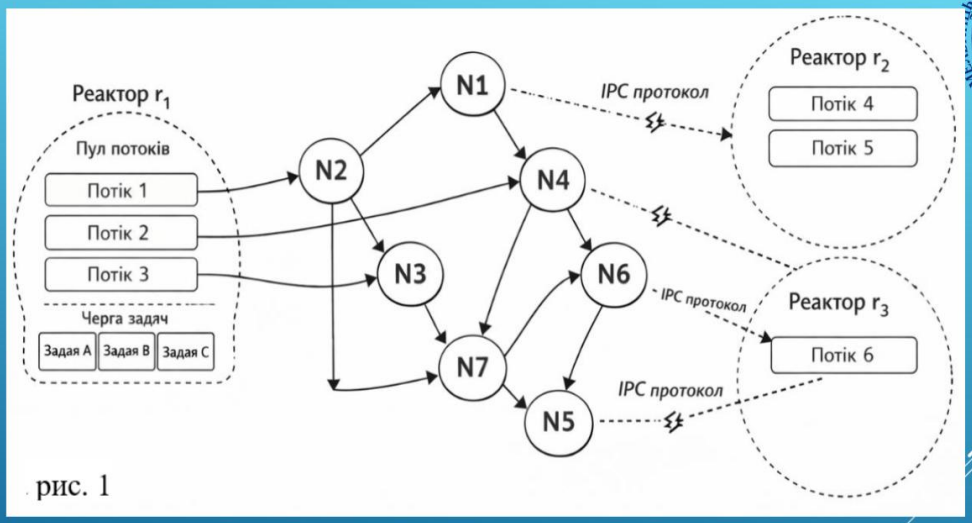


рис. 1

ГРАФ РОЗПОДІЛЕНОГО СЕРЕДОВИЩА

МЕТРИКИ

$$M_H = \{m_{H,1}, m_{H,2}, \dots, m_{H,N_{M_H}}\}, \quad (1)$$

$$q_H = f_H(m_{H,1}, m_{H,2}, \dots, m_{H,N_{M_H}}), \quad (2)$$

$$b_i = \frac{m_{H,i}}{q_H}, \quad (3)$$

$$b = \sum_{i=1}^{N_{M_H}} b_i, \quad (4)$$

$$V_i = V \cdot \frac{b_i}{b}; \quad (5)$$

$$r = V - \sum_{i=1}^{N_{M_H}} V_i, \quad (6)$$

$$r < N_{M_H}, \quad (7)$$

$$f_i = \frac{b_i}{b}, \quad (8)$$

$$T_i = \frac{1}{f_i}, \quad (9)$$

$$T = (T_1, T_2, \dots, T_{N_{M_H}}), \quad (10)$$

$$y = \frac{L}{s} + 1, \quad (11)$$

МЕТОД

Крок 1. Формування подієво-орієнтованої моделі системи.

Крок 2. Обчислення локальних і глобальних метрик.

Крок 3. Реєстрація наміру доступу до ресурсу.

Крок 4. Узгодження глобального порядку доступу.

Крок 5. Локальне та глобальне виявлення потенційних циклів.

Крок 6. Прийняття рішення щодо доступу до ресурсу.

Крок 7. Оновлення стану та адаптація параметрів.

Для комплексної інтегральної оцінки застосовуємо узагальнений показник ефективності:

$$E_{total} = w_1 \cdot (1 - P_{deadlock}) + w_2 \cdot \frac{T_k}{T_{k,max}} + w_3 U - w_4 R_v,$$

де E_{total} - інтегральний показник ефективності; R_v - частка службового трафіку; U - коефіцієнт використання ресурсу; w_1, w_2, w_3, w_4 вагові коефіцієнти, що відображають пріоритетність відповідних критеріїв; T_k - кількість завершених задач за одиницю часу; $T_{k,max}$ - максимальне зафіксоване значення пропускної здатності в контрольному експерименті.

РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТУ

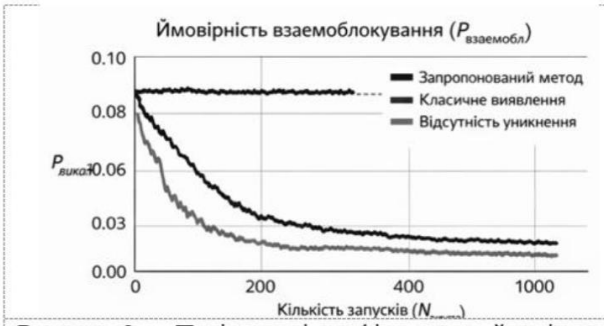


Рисунок 2 – Порівняльні графіки щодо ймовірності виникнення взаємоблокувань

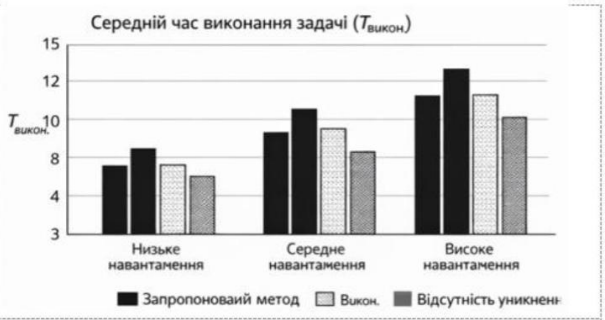


Рисунок 3 – Порівняльні графіки щодо середнього часу виконання завдань

РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТУ

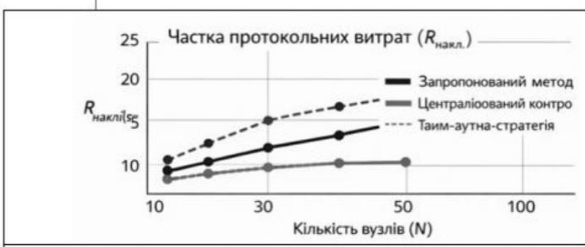


Рисунок 4 – Порівняльні графіки щодо частки протокольних витрат

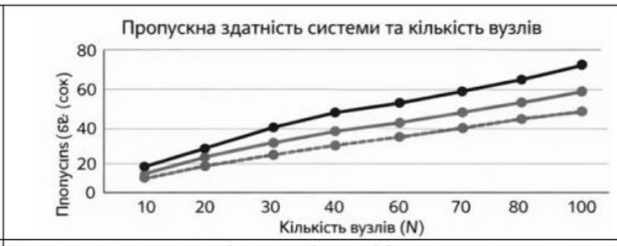


Рисунок 5 – Порівняльні графіки щодо пропускної здатності системи

РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТУ

Таблиця 1

Загальні кількісні показники

Показник	Розроблений	Класичне виявлення	Відсутність уникнення
$P_{deadlock}$	0,002	0,056	0,081
Середній час виконання	5,2 с	8,4 с	11,5 с
Пропускна здатність	45,6	38,2	29,5
Частка витрат	6,7 %	15,4 %	20,3 %

РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТУ

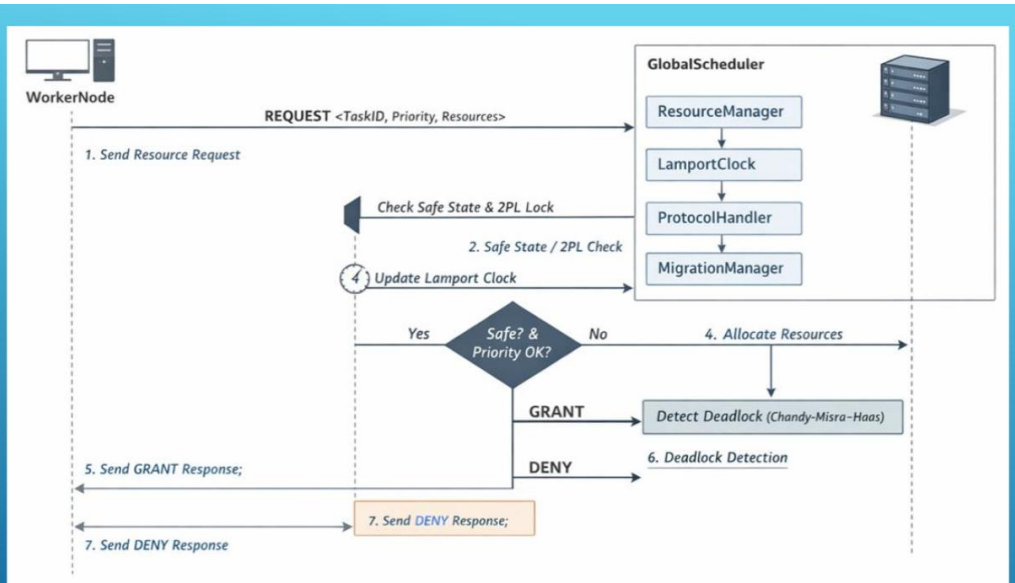


Рисунок 6 – Схема розподіленого середовища виконання завдань

РЕАЛІЗАЦІЯ

- У роботі за результатами виконаних теоретичних та практичних досліджень розроблено новий метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії та отримано наступні результати.
- 1. Проаналізовано відомі методи та засоби уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії.
- 2. Розроблено метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії. Це дало можливість забезпечити структурну відсутність циклів очікування, передбачуване зростання накладних витрат і лінійну динаміку продуктивності при розширенні системи, що підвищує ефективність використання ресурсів і робить систему стабільною під різними навантаженнями.
- 3. Розроблено архітектуру та компоненти розподілених систем, включно з реалізованим методом уникнення взаємоблокувань. Це забезпечило повноцінну координацію виконання завдань, балансування навантаження між вузлами, підтримку реального каналу взаємодії між планувальником та виконавчими вузлами, а також можливість моделювати і тестувати різні стратегії управління ресурсами в гетерогенних середовищах.
- 4. Досліджено метод уникнення взаємоблокувань в розподілених системах та провести з ним експериментальні дослідження. Це дозволило перевірити практичну придатність розробленого методу, підтвердити його стабільність та ефективність у широкому діапазоні навантажень, а також продемонструвати його можливість масштабування, що робить систему придатною для подальшого застосування як експериментальної платформи і основи для реальних розподілених систем з високими вимогами до надійності та ефективності.

ВИСНОВКИ

ДОДАТОК Б

(обов'язковий)

Наукова праця здобувача

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

https://doi.org/10.31891/2219-9365-2026-85-37
УДК 004.75

КРЕЩУК Владислав
Хмельницький національний університет
https://orcid.org/0009-0000-2691-5450
e-mail: vladyslavkreshchuk@gmail.com
ЛИГУН Олексій
Хмельницький національний університет
https://orcid.org/0009-0004-5727-5096
e-mail: oleksii.lyhun@gmail.com
СОРОЧИНСЬКИЙ Олександр
Хмельницький національний університет
https://orcid.org/0009-0003-7966-4861
e-mail: sorochinskiy159@gmail.com

МЕТОД УНИКНЕННЯ ВЗАЄМОБЛОКУВАНЬ ЗАВДАНЬ В РОЗПОДІЛЕНИХ СИСТЕМАХ НА ОСНОВІ ПРОТОКОЛУ МІЖПРОЦЕСНОЇ ВЗАЄМОДІЇ

У статті представлено результати розроблення та дослідження методу уникнення взаємоблокувань завдань у розподілених обчислювальних системах, що ґрунтується на інтеграції подієво-орієнтованої моделі виконання з формалізованим протоколом міжпроцесної взаємодії. Запропонований підхід формує цілісну методологічну основу керування розподіленими обчисленнями, у якій реактор розглядається як центральний координаційний механізм узгодження структурних залежностей між задачами та процедур доступу до ресурсів. Ключовим результатом є забезпечення структурної відсутності циклів очікування за рахунок поєднання алгоритмів виявлення потенційних конфліктів із адаптивним протоколом координації.

У роботі сформовано комплексну систему метрик, що дозволяє кількісно оцінювати стан вузлів, параметри потоків і рівень ресурсного навантаження, що, у свою чергу, створює основу для обґрунтованого вибору стратегій планування, балансування та масштабування. Експериментальні результати, представлені у графічній та табличній формах, підтверджують збереження лінійної динаміки продуктивності при збільшенні кількості вузлів і прогнозоване зростання накладних витрат без втрати стійкості системи. Доведено, що інтеграція формальної моделі залежностей із протоколом обміну повідомленнями гарантує просування критичних шляхів виконання навіть у гетерогенних середовищах.

Отримані результати свідчать про практичну придатність запропонованого методу для побудови масштабованих і надійних розподілених систем, у яких поєднуються вимоги високої ефективності використання ресурсів і стійкості до взаємоблокувань. Перспективи подальших досліджень пов'язані з адаптацією підходу до різнорідних архітектур вузлів і розширених конфігурацій із великою кількістю компонентів.

Ключові слова: розподілена система, комп'ютерна система, взаємоблокування, процеси, протокол, завдання, реактор.

KRESHCHUK Vladyslav, LYHUN Oleksii, SOROCHYNSKYI Oleksandr
Khmelnytskyi National University

METHOD OF AVOIDING INTERLOCKS OF TASKS IN DISTRIBUTED SYSTEMS BASED ON INTERPROCESS INTERACTION PROTOCOL

The article presents the results of the development and research of the method of avoiding mutual blocking of tasks in distributed computing systems, which is based on the integration of an event-oriented model of execution with a formalized protocol of interprocess interaction. The proposed approach forms a holistic methodological basis for managing distributed computing, in which the reactor is considered as a central coordination mechanism for coordinating structural dependencies between tasks and resource access procedures. The key result is ensuring the structural absence of waiting cycles due to the combination of algorithms for detecting potential conflicts with an adaptive coordination protocol.

In the work, a complex system of metrics was formed, which allows to quantitatively evaluate the state of nodes, flow parameters and the level of resource load, which, in turn, creates a basis for a reasonable choice of planning, balancing and scaling strategies. The experimental results, presented in graphic and tabular forms, confirm the preservation of the linear dynamics of performance with an increase in the number of nodes and the predicted increase in overhead costs without loss of system stability. Integrating a formal dependency model with a messaging protocol has been proven to guarantee the advancement of critical execution paths even in heterogeneous environments.

The obtained results indicate the practical suitability of the proposed method for building scalable and reliable distributed systems that combine the requirements of high efficiency of resource use and resistance to mutual blocking. Prospects for further research are related to the adaptation of the approach to heterogeneous node architectures and advanced configurations with a large number of components.

Keywords: distributed system, computer system, interlocking, processes, protocol, task, reactor.

Стаття надійшла до редакції / Received 20.12.2025
Прийнята до друку / Accepted 16.01.2026
Опубліковано / Published 05.03.2026

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

З кожним роком стає дедалі складніше підвищувати обчислювальну потужність окремої машини виключно за рахунок збільшення тактової частоти процесора чи кількості ядер, то сучасні інформаційні технології активно переходять до використання паралельних і розподілених систем [1, 2]. Такий підхід дозволяє досягти суттєвого приросту продуктивності шляхом декомпозиції складної задачі на менші підзадачі, які можуть виконуватися одночасно на кількох обчислювальних вузлах або процесах. Розподілені обчислення відкривають широкі можливості для масштабування сервісів, підвищення відмовостійкості та ефективного використання ресурсів, що є особливо актуальним в умовах стрімкого розвитку хмарних технологій, великих даних та високонавантажених систем [3, 4].

Водночас перехід від послідовної моделі виконання до паралельної та розподіленої суттєво ускладнює процес проєктування, реалізації та супроводу програмного забезпечення. Якщо в послідовних системах основна увага приділяється алгоритмічній оптимізації та ефективному використанню пам'яті, то в паралельних середовищах додаються проблеми синхронізації, координації доступу до спільних ресурсів, забезпечення узгодженості даних і коректної міжпроцесної взаємодії. Неправильна організація обміну повідомленнями або керування ресурсами може призвести до критичних збоїв у роботі системи, які складно виявити та усунути.

Однією з найбільш небезпечних і водночас поширених проблем у паралельних і розподілених системах є взаємоблокування. Воно визначається як стан, у якому виконання програми повністю або частково зупиняється через те, що система не може надати необхідні ресурси для продовження роботи процесів, або ж виникає циклічна залежність між ними. У такій ситуації кожен із процесів очікує на ресурс, який утримується іншим процесом, що, своєю чергою, також перебуває в стані очікування. У результаті жоден із процесів не може продовжити виконання, а система переходить у стан блокування.

Особливу складність становить те, що в розподілених системах взаємоблокування можуть виникати не лише через спільне використання апаратних ресурсів, а й через некоректну логіку протоколів обміну повідомленнями, затримки в мережі або порушення порядку виконання операцій. Виявлення таких ситуацій ускладнюється відсутністю централізованого контролю та необхідністю аналізу глобального стану системи. Наслідками взаємоблокувань можуть бути зниження продуктивності, втрата доступності сервісів, порушення цілісності даних та фінансові збитки.

У зв'язку з цим особливої актуальності набуває розробка ефективних методів запобігання виникненню взаємоблокувань ще на етапі проєктування системи. На відміну від підходів, що передбачають лише виявлення та усунення взаємоблокувань після їх виникнення, методи уникнення дозволяють організувати взаємодію процесів таким чином, щоб виключити саму можливість формування циклічних залежностей. Одним із перспективних напрямів у цьому контексті є використання формалізованих протоколів міжпроцесної взаємодії, які регламентують порядок обміну повідомленнями, виділення та звільнення ресурсів, а також узгодження станів процесів.

Таким чином, актуальність роботи зумовлена необхідністю підвищення надійності та ефективності розподілених систем шляхом запобігання взаємоблокуванням.

АНАЛІЗ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

Розподілені системи стають дедалі поширенішими в сучасному інформаційному просторі, оскільки обсяги даних та інтенсивність обчислювальних навантажень постійно зростають, а підвищення продуктивності однієї окремої машини стає економічно та технічно менш доцільним. Масштабування «вгору» поступово поступається масштабуванню «в ширину», коли обчислювальні ресурси нарощуються шляхом об'єднання багатьох вузлів у єдину систему. Такий підхід дозволяє досягати високої продуктивності, гнучкості та відмовостійкості, однак водночас породжує низку нових викликів, які відсутні або менш виражені у традиційних послідовних виконаннях [5, 6].

У розподіленому середовищі виконання завдань відбувається паралельно на кількох вузлах, які взаємодіють між собою через мережу. Це створює додаткові складнощі, пов'язані з координацією доступу до ресурсів, синхронізацією станів, затримками передавання повідомлень і частковою відмовою окремих компонентів. Однією з критичних проблем є ситуація, коли система більше не може просуватися вперед через те, що завдання були розподілені таким чином, що вони взаємно блокують одне одного [7, 8]. У такому випадку процеси перебувають у стані очікування ресурсів або повідомлень, які ніколи не будуть отримані, що призводить до повної або часткової зупинки системи.

Зі збільшенням масштабу та складності розподілених систем зростає й складність виявлення та аналізу таких станів. Відсутність централізованого контролю, асинхронність обміну даними та динамічний характер розподілу ресурсів ускладнюють своєчасне врахування можливості виникнення глухого кута ще на етапі проєктування. У результаті навіть незначні помилки в логіці планування або розподілу ресурсів можуть мати суттєві наслідки для всієї системи.

У статті [9] було запропоновано схему диспетчеризації, що грунтується на принципах уникнення

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

взаємоблокувань та забезпечує гарантії живучості системи. Запропонований підхід спрямований на те, щоб система ніколи не переходила в стан взаємоблокування, навіть за умов інтенсивного розподілу задач і ресурсів. Вона поєднує механізми планування та контролю залежності між процесами, забезпечуючи формальні гарантії коректності виконання. Ця робота розпочинається з детального аналізу підходу та його теоретичних засад. Зокрема, розглядається певний вимір проблеми від розподілу сайтів між вузлами до формально визначеного з прямими наслідками для здійсненності та ефективності системи. Уточнення цього аспекту дозволяє глибше зрозуміти обмеження моделі та визначити умови, за яких гарантії уникнення взаємоблокувань зберігаються. В роботі пропонується конкретна стратегія розподілу вузлів на сайти в однорідному середовищі, де всі обчислювальні ресурси мають подібні характеристики. На основі цієї моделі здійснюється узагальнення для гетерогенного середовища, в якому вузли можуть відрізнятися за продуктивністю, обсягом пам'яті чи мережними параметрами. Окрему увагу приділено можливостям оптимізації ефективності в межах запропонованих розподілів без порушення гарантій живості системи. У роботі також представлено відповідні евристики планування, які дозволяють зменшити накладні витрати на координацію процесів та покращити балансування навантаження. Завершальним етапом є розробка моделі динамічної підграфової доцільності, що базується на реалізації цієї моделі і враховує зміну стану системи в реальному часі. Поєднання здійсненності, яку забезпечує стратегія розподілу реакторів, із гарантіями уникнення взаємоблокування та максимальної живучості системи формує цілісне й оптимальне середовище для ефективного розподіленого виконання складних обчислювальних задач.

У складних системах розподілу ресурсів [10, 11], зокрема в автоматизованих виробничих системах, гнучких виробничих модулях, логістичних комплексах, а також у високонавантажених програмно-апаратних середовищах, ефективне управління потоками завдань є критично важливим для забезпечення безперервності технологічних процесів. Використання багатопотокових і паралельних програм дозволяє значно підвищити продуктивність системи, скоротити час виконання операцій, оптимізувати використання обладнання та зменшити енергоспоживання. Завдяки одночасному виконанню декількох процесів досягається краща утилізація ресурсів, знижується час простою машин і підвищується загальна пропускна здатність системи [12].

Проте впровадження паралелізму неминуче супроводжується появою додаткових ризиків і технічних ускладнень. До найбільш поширених проблем належать умови конкуренції, некоректна синхронізація доступу до спільних ресурсів, взаємоблокування, а також ситуації голодування [13, 14]. Такі явища можуть суттєво знижувати продуктивність системи, призводити до втрати узгодженості даних, збільшення часу простою обладнання та навіть створювати загрозу безпеці виробничих процесів. Особливо критичними ці наслідки є для автоматизованих виробничих систем, де порушення логіки керування може спричинити не лише економічні збитки, а й аварійні ситуації. Саме тому питання діагностики несправностей, виявлення причин блокувань, розробки методів їх усунення та запобігання взаємоблокуванням у паралельних програмах викликають є актуальними [15, 16].

З метою уникнення тупікових ситуацій у складних динамічних системах активно застосовується теорія дискретного керування [17, 18], яка забезпечує формальні засоби моделювання, аналізу та синтезу керуючих стратегій. Одним із найпотужніших інструментів у цій галузі є мережі Петрі, що завдяки своїй наочності та високій виразній здатності широко використовуються для моделювання автоматизованих виробничих систем [19], багатопоточних програмних систем [20], систем обслуговування, транспортних мереж та інших прикладних задач [21]. Мережі Петрі дозволяють формалізувати паралельність, конкуренцію, синхронізацію та розподіл ресурсів у межах єдиної математичної моделі, що значно полегшує аналіз поведінки системи.

Таким чином, проблема синтезу ефективного, структурно простого й обчислювально прийнятного супервізора для великомасштабних розподілених систем залишається не вирішеною. Це зумовлює необхідність подальших досліджень, спрямованих на розробку нових методів керування, які поєднуватимуть гарантії живучості, відсутності взаємоблокувань та мінімальної структурної складності. Також, необхідно провести аналіз щодо дисперсії розмірів пулів потоків та її вплив на максимальну доцільність і здійсненність системи.

ФОРМУЛЮВАННЯ ЦІЛЕЙ СТАТТІ

Метою роботи є забезпечення узгодженої координації процесів, мінімізація ризику виникнення циклічних залежностей та підвищення загальної стабільності функціонування системи шляхом розроблення методу уникнення взаємоблокувань завдань у розподілених системах на основі протоколу міжпроцесної взаємодії.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

Моделювання взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії

Моделювання взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії здійснимо з використанням теорії графів та реакторів, що дають змогу відокремити події у вузлах.

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

Для загальних орієнтованих ациклічних графів (ОАГ), що моделюють графи викликів у розподілених системах, питання конфлікту при призначенні вузлів реакторам набуває принципового значення. На відміну від структур типу дерева, ОАГ допускають існування вузлів із кількома вхідними шляхами, що породжує потенційну неоднозначність у визначенні реактора для такого вузла. Якщо в дереві кожна вершина має єдиний шлях від кореня, то в ОАГ одна й та сама вершина може належати кільком шляхам різної довжини. Відтак виникає потреба у формальному механізмі визначення пріоритету між шляхами з метою забезпечення узгодженості розподілу.

Пропонована стратегія ґрунтується на принципі пріоритету довшого шляху. Інтуїтивно це відповідає орієнтації на критичний шлях, тобто на ту послідовність вузлів, яка визначає мінімальний можливий час завершення всього графа. Якщо вузол належить двом або більше шляхам, то реактор для нього визначається тим шляхом, який має більшу довжину. У випадку рівності довжин застосовується довільний, але детермінований порядок, що не впливає на фінальне забарвлення. Такий підхід забезпечує, що конфлікти вирішуються не локально, а з урахуванням глобальної структури графа.

Ключовою метою є збереження інваріанта нерозподілу, який формулюється як умова $\lceil L / n \rceil \geq \max(R(p))$ для будь-якого шляху p, де L - довжина критичного шляху, n - кількість реакторів, а $\max(R(p))$ - максимальна кількість вузлів шляху p, призначених одному реактору. Цей інваріант гарантує, що якщо ресурси достатні для обслуговування критичного шляху, то стратегія розподілу не створить штучного вузького місця через невдале призначення реакторів.

Реалізаційно стратегію базуватимемо на двофазному алгоритмі. На першій фазі для кожного джерельного вузла ОАГ виконаємо DFS-обхід. Уздовж кожного шляху використовується циклічний лічильник із періодом n, що забезпечує рівномірне чергування реакторів. Кожному шляху присвоюється унікальний ідентифікатор, а його довжина накопичується як рекурсивний параметр. Результатом цієї фази є хеш-таблиця, що відображає ідентифікатори шляхів у їхню довжину.

На другій фазі здійснюється повторний обхід графа, під час якого кожна вершина отримує реактор відповідно до того шляху, що має максимальну довжину серед усіх шляхів, які проходять через цю вершину. Завдяки доведеному факту, що шляхи однакової довжини можна впорядковувати довільно без зміни кінцевого забарвлення, алгоритм є детермінованим і не залежить від конкретного порядку обходу.

Асимптотична складність процедури у найгіршому випадку становить $O(|V|(|V|+|E|))$, оскільки алгоритм запускається з кожного джерела. Додатковий обхід та операції з хеш-таблицею не змінюють порядку складності. Важливо, що алгоритм може виконуватися офлайн, під час етапу завантаження або компіляції графа, що усуває його вплив на час виконання системи.

Доведення коректності базується на аналізі можливих спроб порушення інваріанта нерозподілу. Припустимо, що існує шлях p, для якого можна сконструювати набір шляхів p_i, що перезаписують призначення його вершин так, щоб $\max(R(p))$ перевищив допустиму межу. Для цього кожен шлях p_i повинен бути довшим за p, оскільки коротший або рівний шлях не може змінити забарвлення через правило пріоритету. Більше того, щоб змінити колір вершини, довжина нового шляху має перевищувати попередню щонайменше на n (через циклічність мод n). Це означає, що кожна така ітерація збільшує L щонайменше на n. Відтак будь-яка спроба примусового перезапису неминуче призводить до зростання L, що зберігає виконання інваріанта. Таким чином, не існує конструкції, яка могла б порушити інваріант без одночасного збільшення критичного шляху.

Для систем, близьких до межі здійсненності, випадковий розподіл дедалі частіше порушує інваріант нерозподілу, особливо зі збільшенням кількості реакторів. Натомість запропонована стратегія не демонструє жодного порушення в експериментальних серіях. Це свідчить про те, що стратегічний, структурно обґрунтований розподіл є не лише оптимізаційною перевагою, а необхідною умовою забезпечення коректності у масштабованих розподілених системах.

Отже, запропонований підхід забезпечує формально доведену узгодженість призначення реакторів у довільних ОАГ, зберігає інваріант нерозподілу та гарантує пріоритет критичного шляху. Це створює теоретичну основу для побудови відмовостійких і продуктивних механізмів планування в розподілених середовищах.

Реактор у контексті графа викликів або обчислювального графа походить із архітектурного шаблону Reactor pattern та є одним із базових підходів до побудови високопродуктивних подієво-орієнтованих систем. У класичному розумінні реактор це механізм організації обробки подій, що поєднує мультиплексування джерел подій, їх диспетчеризацію та виконання відповідних обробників у межах контрольованого середовища виконання. Однак у дослідницькому контексті розподілених і паралельних систем це поняття набуває розширеного змісту та трансформується в абстрактну модель керування виконанням залежних задач.

У моделі графа викликів реактор розглядається як логічна одиниця виконання, якій призначається підмножина вузлів графа. Вузли інтерпретуються як окремі обчислювальні операції або задачі, а ребра як відношення залежності чи причинно-наслідкового виклику. Таким чином, реактор виступає абстрактним ресурсом планування, що визначає середовище, у якому виконується відповідна група вузлів. У фізичній реалізації реактор може відповідати окремому потоку, пулу потоків або ізольованому циклу обробки подій,

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

проте в теоретичній моделі він трактується як ресурс із власною чергою задач і обмеженою пропускною здатністю. Запровадження цієї абстракції є необхідним для формалізації розподілу паралелізму та контролю конкуренції в розподілених системах. Відсутність глобального часу, асинхронність обміну повідомленнями, часткові відмови та варіативність затримок мережі створюють передумови для виникнення взаємоблокувань. Якщо кілька логічно залежних вузлів виконуються в одному середовищі без чітко визначеної дисципліни планування, можливе утворення циклічних очікувань. Призначення вузлів різним реакторам дозволяє структуровано розмежувати області виконання, мінімізувати локальну конкуренцію за ресурси та формалізувати умови, за яких неможливе виникнення циклічного очікування. Саме тому введення поняття реактора є методологічною передумовою розроблення методу уникнення взаємоблокувань у розподілених системах.

З формальної точки зору реактор задамо функцією $R: V \to \{r_1, r_2, \ldots, r_n\}$, де V - множина вузлів графа, а $\{r_1, \ldots, r_n\}$ - множина реакторів. Це відображення задає розбиття графа на підмножини, кожна з яких виконується в межах окремого середовища планування. На відміну від класичного розфарбовування графа, де кольори мають лише топологічне значення, реактори мають операційний сенс. Вони відповідають конкретним обчислювальним ресурсам, що впливають на часові характеристики виконання, довжину критичного шляху та можливість блокувань. Коректно вибрана стратегія відображення дозволяє обмежити максимальну кількість послідовних вузлів одного шляху, що виконуються в одному реакторі, тим самим знижуючи ризик локального накопичення залежності. На рис. 1 зображено приклад з використанням реактора.

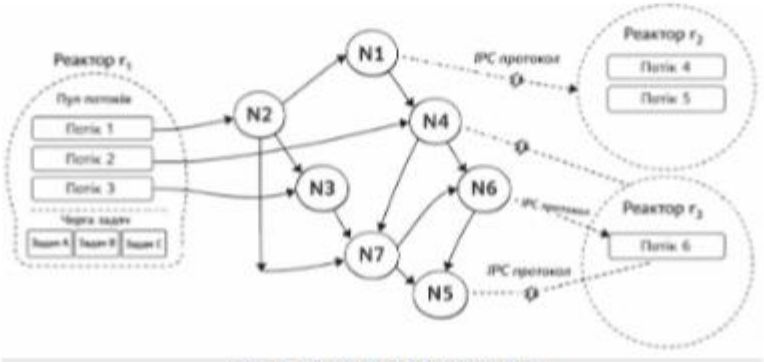


Рис. 1. Приклад графа з реактором

Поданий граф на рис. 1 є прикладом формалізованої моделі виконання задач у розподіленій системі з використанням реакторної архітектури. Він поєднує дві сутності, а саме: граф залежності (граф викликів); множину реакторів як ізольованих середовищ виконання, пов'язаних протоколом міжпроцесної взаємодії (IPC). Така структура дозволяє одночасно описувати логіку обчислень і механізм їх фізичного виконання.

У центрі схеми на рис. 1 розташований граф викликів, що складається з вузлів N1–N7 та орієнтованих ребер між ними. Вузол інтерпретується як елементарна обчислювальна операція, сервісний виклик або транзакційний крок. Орієнтоване ребро відображає залежність типу «після виконання A може бути виконано B» або «B потребує результату A». Таким чином, граф задає частковий порядок виконання задач. Якщо розглядати його формально, то це орієнтований ациклічний граф (ОАГ), що визначає множину допустимих послідовностей виконання.

Ліва та права частини схеми відображають реактори r_1, r_2 та r_3. Кожен реактор представлений як ізольоване середовище з власним пулом потоків і чергою задач. Пул потоків визначає внутрішній рівень паралелізму, тоді як черга задач задає дисципліну планування. Реактор отримує задачі (вузли графа), обробляє їх відповідно до внутрішнього алгоритму планування та може ініціювати взаємодію з іншими реакторами через IPC-протокол.

Пунктирні стрілки між вузлами графа на рис. 1 та іншими реакторами позначають міжпроцесну взаємодію. Це означає, що виконання певного вузла вимагає відправлення повідомлення до іншого реактора та очікування відповіді. Саме в цій точці логічний граф залежностей перетворюється на мережу ресурсних очікувань.

Граф може описувати блокування в розподілених системах через накладання двох структур, а саме: графа залежності; графа ресурсного очікування. Якщо вузол N4 у реакторі r_1 надсилає запит до реактора r_2 і блокується до отримання відповіді, а в той самий час у r_2 виконується вузол, який потребує результату з r_1, то виникає циклічне очікування між реакторами. Формально це означає, що в системі формується цикл у графі очікування ресурсів, навіть якщо первинний граф викликів є ациклічним.

Блокування можливе через кілька механізмів. По-перше, через обмеженість потоків у межах реактора. Якщо всі потоки зайняті задачами, що очікують відповіді від інших реакторів, то нова задача, яка могла б

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

розірвати цикл залежностей, не буде запущена. По-друге, через синхронну модель IPC, коли реактор утримує ресурс під час очікування відповіді. По-третє, через конкуренцію за спільні логічні ресурси, які не відображені явно у графі викликів, але проявляються у виконанні.

У такій моделі виділимо два типи графів так:

1) граф причинно-наслідкових залежностей;

2) граф очікування ресурсів, який формується динамічно під час виконання.

Блокування виникає тоді, коли у графі очікування ресурсів з'являється цикл. Представлена схема демонструє, як такий цикл може виникнути між реакторами через IPC-виклики. Якщо призначення вузлів реакторам здійснюється за визначеною стратегією, наприклад, циклічним розподілом із пріоритетом критичного шляху, тоді можна формально довести, що цикл очікування не утворюється або що завжди існує хоча б один реактор, здатний продовжити виконання. Тобто реакторна декомпозиція графа дозволяє контролювати структуру можливих ресурсних конфліктів.

Отже, даний граф є формальною моделлю взаємодії обчислювальних задач і ресурсів у розподіленому середовищі. Він демонструє, як логічні залежності трансформуються у ресурсні очікування, і показує механізм виникнення взаємоблокувань через циклічні міжреакторні залежності. Саме тому така модель є придатною для аналізу коректності протоколів міжпроцесної взаємодії та розроблення методів гарантування прогресу виконання.

Ключовим доповненням до цієї моделі є протокол міжпроцесної взаємодії, який визначає правила обміну повідомленнями між реакторами. Оскільки вузли графа, призначені різним реакторам, можуть мати залежності, то їх взаємодія відбувається через передачу подій або повідомлень. Протокол міжпроцесної взаємодії встановлює порядок ініціювання викликів, підтвердження завершення операцій, обробки помилок і тайм-аутів. У контексті уникнення взаємоблокувань важливо, щоб цей протокол виключав можливість двостороннього синхронного очікування між реакторами. Це досягається шляхом використання асинхронної моделі запит–відповідь, односторонніх повідомлень або дисципліни «неблокуючого виклику», за якої реактор не утримує ресурс під час очікування відповіді від іншого реактора.

Таким чином, реактор виступає не лише абстрактною одиницею обчислювального ресурсу, а й елементом формальної моделі, що поєднує структуру графа залежностей із протоколом міжпроцесної взаємодії. Саме інтеграція стратегії розподілу реакторів із коректно визначеним протоколом обміну повідомленнями створює основу для побудови методу уникнення взаємоблокувань. Вона дозволяє довести, що за певних умов призначення вузлів та правил взаємодії між реакторами система не утворює циклів очікування, а отже, забезпечує гарантовану можливість просування виконання критичного шляху навіть у складних розподілених середовищах.

Метрики для організації оцінювання виконання потоків в розподілених системах

Визначимо підходи до організації роботи систем із гетерогенними серверами швидкості. У дослідженні високопродуктивних обчислювальних систем особливу увагу приділяють задачі ефективного розподілу потоків між серверами з різними швидкісними характеристиками. Такі системи називають гетерогенними, оскільки їхні обчислювальні вузли мають відмінні параметри продуктивності. Проблема оптимального розподілу ресурсів у цьому контексті може бути формалізована через два базові підходи, кожен з яких відповідає різним початковим умовам конфігурації системи. Визначимо два підходи в залежності від кількості потоків в розподілених системах.

Перший підхід до розподілу потоків визначимо з урахуванням того, що система функціонує з фіксованою кількістю потоків. Тоді, у межах першого підходу вважатимемо, що задано множину серверів із різними швидкостями обробки та загальну кількість потоків (S), які необхідно розподілити між серверами. І на основі такого початкового вибору задамо моделі. Нехай множину швидкостей серверів визначимо так:

$$M_H = \left\{m_{H,1}, m_{H,2}, \dots, m_{H,N_{M_H}}\right\}, \quad (1)$$

де $m_{H,i}$ - швидкість і-го сервера; N_{M_H} – кількість серверів; $i = 1,2, \dots, N_{M_H}$.

З метою коректного порівняння продуктивності серверів проведемо нормалізацію відносно найбільшого спільного дільника для елементів множини M_H так:

$$q_H = f_H(m_{H,1}, m_{H,2}, \dots, m_{H,N_{M_H}}), \quad (2)$$

де f_H – функція знаходження найменшого за значенням елементу серед елементів множини M_H.

Тоді визначаємо коефіцієнти відносної швидкості серверів так:

$$b_i = \frac{m_{H,i}}{q_H}, \quad (3)$$

де $m_{H,i}$ - швидкість і-го сервера; N_{M_H} – кількість серверів; $i = 1,2, \dots, N_{M_H}$; q_H - найменший за значенням елемент серед елементів множини M_H.

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

Сумарний коефіцієнт відносної швидкості серверів дає змогу оцінити частку кожного сервера в загальній продуктивності розподілених систем. Визначимо його так:

$$b = \sum_{i=1}^{N_{M_H}} b_i, \quad (4)$$

де b_i - коефіцієнти відносної швидкості і-го сервера; N_{M_H} – кількість серверів; $i = 1,2, \dots, N_{M_H}$.

Тоді частка $\frac{b_i}{b}$ буде відображати частку загальної продуктивності системи, що припадає на (і)-й сервер.

Розподіл потоків здійснимо за такою схемою:

1) первинний розподіл потоків виконуватимемо пропорційно відносним швидкостям:

$$V_i = V \cdot \frac{b_i}{b}; \quad (5)$$

2) оскільки використовується операція округлення, то виникає залишок потоків:

$$r = V - \sum_{i=1}^{N_{M_H}} V_i, \quad (6)$$

3) потрібно здійснити перевірку щодо коректності отриманого результату так:

$$r < N_{M_H}, \quad (7)$$

де b_i - коефіцієнти відносної швидкості і-го сервера; N_{M_H} – кількість серверів; $i = 1,2, \dots, N_{M_H}$.

Згідно формули (7) похибка округлення кожного доданка не може бути більшою за одиницю. Якщо нерівність не виконується, тоді в системі наявна проблема. Залишкові потоки розподіляються ітеративно, починаючи з серверів із найбільшою швидкістю, що мінімізує очікуваний час виконання.

Така схема дає змогу редукувати гетерогенну систему до еквівалентної однорідної моделі шляхом уявного "розщеплення" кожного сервера на b_i віртуальних підсерверів із однаковими характеристиками. Це створює аналітичну можливість застосування методів, які розроблені для однорідних систем, при збереженні максимальної заповненості та балансування навантаження.

За другим підходом розглядатимемо розподіл реакторів у системі з попередньо визначеними пулами потоків. Приймемо, що сервери мають різні швидкості і те, що кожен сервер уже оснащений пулом потоків однакового розміру. Завдання полягає у розподілі реакторів або обчислювальних вузлів між серверами таким чином, щоб зберігалася інваріантність здійсненності та враховувалися швидкісні відмінності серверів. Введемо метрики для фіксації цих дій в розподілених системах.

Аналіз заповненості системи

Нехай L - довжина критичного шляху, що вимірюється в потоках, S - загальна кількість потоків.

Якщо L= S, то система повністю заповнена і будь-яка спроба нерівномірного розподілу може порушити гарантії здійсненності.

Якщо ж L< S, то виникає резерв продуктивності, що дозволяє здійснити контрольований дисбаланс на користь швидших серверів.

Здійснимо формалізацію частот і періодів як метрик для процесів чи завдань в розподілених системах. Аналогічно до першого підходу визначимо так відносну частоту планування сервера:

$$f_i = \frac{b_i}{b}, \quad (8)$$

де b_i - коефіцієнти відносної швидкості і-го сервера; N_{M_H} – кількість серверів; $i = 1,2, \dots, N_{M_H}$; b - загальна продуктивності розподілених систем.

Ці значення f_i відображають відносну частоту планування сервера.

Період обслуговування сервера:

$$T_i = \frac{1}{f_i}, \quad (9)$$

де f_i - коефіцієнти відносної швидкості і-го сервера; N_{M_H} – кількість серверів; $i = 1,2, \dots N_{M_H}$.

Таким чином, швидші сервери мають менші періоди та повинні плануватися частіше.

Для формування простору можливих рішень використаємо експоненціальний розподіл періодів, адаптований не до розмірів пулів, а до швидкісного розподілу. Визначимо його вектором так:

$$T = (T_1, T_2, \dots, T_{N_{M_H}}), \quad (10)$$

де N_{M_H} – кількість серверів; $i = 1,2, \dots N_{M_H}$; T_i - період обслуговування і-го сервера.

Проте виникає конфлікт оптимізаційних критеріїв в контексті максимізації продуктивності і

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

забезпечення здійсненності.

Розглянемо забезпечення здійсненності в однорідному випадку. Оскільки всі сервери мають однаковий розмір пулу потоків, то перевірку здійсненності можна спростити. Для цього введемо мінімальний допустимий період так:

$$y = \frac{L}{S} + 1, \tag{11}$$

де L - довжина критичного шляху, що вимірюється в потоках, S - загальна кількість потоків.

Якщо всі періоди $T_i \geq y$, то гарантується відсутність конфліктів планування. Найменший період відповідає найшвидшому серверу. Обмеження мінімального періоду зменшує простір пошуку та дозволяє будувати лише гарантовано здійсненні розподіли.

Критерій оптимальності дає змогу перевіряти наближення поточного стану до ідеально можливого варіанту. Якість отриманого експоненціального розподілу оцінимо за ступенем наближення до ідеального швидкісного профілю $T = (T_1, T_2, \ldots, T_{N_{M_H}})$ (формула (10)).

Чим ближчий фактичний розподіл до теоретично оптимального, за умови дотримання інваріантів здійсненності, тим вища сумарна продуктивність системи.

Обидва підходи демонструють різні стратегії адаптації до гетерогенності. Перший підхід орієнтований на пропорційний розподіл потоків за умови централізованого контролю. Другий підхід досліджує можливості контрольованого дисбалансу за умови збереження інваріантів здійсненності. Обидва підходи є основою формалізованої моделі, яка дозволяє одночасно враховувати неоднорідність обчислювальних ресурсів, мінімізувати час виконання, гарантувати коректність та стабільність функціонування системи.

Оцінювання виконання потоків у розподілених системах із гетерогенними серверами швидкості базується на системі кількісних метрик, що дозволяють формально описати продуктивність, баланс навантаження, ефективність використання ресурсів і якість обслуговування. У таких системах кожен сервер характеризується власною швидкістю обробки за параметром середньої кількості завдань або інструкцій, які сервер здатен виконати за одиницю часу. Якщо система складається з m серверів, то їхня сукупна теоретична продуктивність дорівнює сумі індивідуальних швидкостей, однак фактична продуктивність залежить від політики розподілу потоків і рівня завантаження.

Базовою метрикою є інтенсивність вхідного потоку завдань, що визначає середню кількість запитів або потоків, які надходять до системи за одиницю часу. Цей показник дозволяє визначити відповідність загальної обчислювальної потужності системи поточному навантаженню. Якщо інтенсивність надходження перевищує сумарну здатність серверів обробляти задачі, то у системі починають накопичуватися черги, що призводить до зростання затримок та зниження якості обслуговування. Тому першочерговим критерієм є забезпечення стабільності системи, за якої кожен сервер здатен обробляти закріплену за ним частину потоку без накопичення нескінченної черги.

Важливою метрикою є коефіцієнт завантаження окремого сервера. Він показує, яку частку свого потенціалу використовує вузол у конкретний момент часу. У гетерогенній системі одні сервери можуть працювати майже на межі своїх можливостей, тоді як інші залишаються частково недовантаженими. Такий дисбаланс негативно впливає на загальну ефективність. Тому при оцінюванні системи аналізують не лише середнє завантаження, а й рівномірність його розподілу між вузлами. Оптимальною вважається ситуація, коли навантаження розподіляється пропорційно швидкості серверів, тобто швидші вузли отримують більшу частку потоків.

Наступною суттєвою характеристикою є середній час перебування потоку в системі. Він охоплює як час очікування в черзі, так і безпосередній час обробки на сервері. У гетерогенних системах цей показник може істотно відрізнятися залежно від того, на який вузол потрапляє завдання. Якщо алгоритм планування не враховує різницю швидкостей, повільні сервери можуть створювати «вузькі місця», що збільшує загальну затримку. Тому середній час обробки є одним із ключових критеріїв оцінювання ефективності політики розподілу потоків.

Пропускна здатність системи відображає кількість задач, які завершуються за певний проміжок часу. У стабільному режимі вона приблизно дорівнює інтенсивності надходження запитів. Проте у випадку перевантаження фактична пропускна здатність може знижуватися через накопичення черг і збільшення часу очікування. У гетерогенному середовищі досягнення максимальної пропускної здатності залежить від здатності балансувальника навантаження враховувати різну продуктивність серверів. Якщо потоки розподіляються без урахування цих відмінностей, частина ресурсів може використовуватися неефективно.

Для аналізу рівномірності використання ресурсів застосовують метрики дисбалансу навантаження. Вони показують, наскільки відрізняється завантаження окремих серверів від середнього по системі. Чим менша ця різниця з урахуванням продуктивності вузлів, тим ефективніше організовано розподіл потоків. У гетерогенних системах абсолютна рівність завантаження не є ціллю; натомість важливо забезпечити пропорційність між швидкістю сервера та обсягом оброблюваних ним задач.

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

Ще однією вагомою метрикою є ефективність використання ресурсів. Вона відображає, яку частину свого потенціалу система реально використовує для виконання корисної роботи. Якщо сервери простоюють або працюють нерівномірно, загальна ефективність знижується. У сучасних розподілених середовищах керування контейнерами та мікросервісами, таких як Kubernetes, подібні показники використовуються для автоматичного масштабування та оптимального розміщення навантаження. У системах потокової обробки даних, наприклад у Apache Kafka, метрики продуктивності застосовуються для балансування партицій між брокерами та контролю швидкості обробки повідомлень.

Окремо аналізуватимемо показник прискорення, який дозволяє порівняти продуктивність гетерогенної системи з базовою конфігурацією, наприклад із виконанням на одному сервері. Ця метрика демонструє, наскільки ефективно система використовує паралелізм. В ідеальному випадку збільшення кількості серверів приводить до пропорційного скорочення часу виконання задачі, проте в реальних умовах існують накладні витрати на координацію, обмін повідомленнями та синхронізацію, що знижує фактичний виграш.

Крім того, важливою характеристикою є масштабованість системи. Вона показує, як змінюється продуктивність при зростанні кількості потоків або при додаванні нових серверів. У добре спроєктованій системі продуктивність зростає майже лінійно до досягнення апаратних або мережевих обмежень. У гетерогенному середовищі масштабованість ускладнюється різницею в швидкості вузлів і потребує адаптивних алгоритмів балансування.

Розглядаючи підходи до організації роботи систем залежно від кількості потоків, можна виділити два принципово різні режими. У першому випадку кількість потоків не перевищує кількість серверів або є близькою до неї. Тут головною метою є мінімізація часу завершення кожного окремого завдання. Раціональним рішенням стає призначення більш складних або ресурсоємних задач на швидші сервери. У другому випадку, коли потоків значно більше, ніж серверів, формується чергова модель обслуговування. Основний акцент переноситься на підтримання стабільності системи, мінімізацію середнього часу очікування та забезпечення високої пропускної здатності. У цьому режимі критично важливим стає пропорційний розподіл навантаження відповідно до продуктивності вузлів.

Отже, система метрик оцінювання виконання потоків у гетерогенних розподілених середовищах охоплює показники навантаження, затримки, пропускної здатності, ефективності, масштабованості та рівномірності використання ресурсів. Їхній комплексний аналіз дозволяє обґрунтовано вибирати алгоритми планування, політики балансування та стратегії масштабування, що особливо важливо в умовах зростаючої складності сучасних обчислювальних інфраструктур.

Метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії

У розподілених системах міжпроцесна взаємодія завдань відбувається за відсутності спільної глобальної пам'яті та єдиного фізичного годинника, що ускладнює координацію доступу до ресурсів і створює передумови для виникнення взаємоблокувань. Взаємоблокування виникає тоді, коли множина процесів перебуває в стані циклічного очікування. Кожен процес утримує певні ресурси та одночасно очікує на ресурс, який утримує інший процес з тієї ж множини. Саме усунення можливості утворення такого циклу або гарантування його розриву лежить в основі більшості протоколів уникнення взаємоблокувань. Більшість протоколів у розподілених системах спрямовані саме на усунення умови циклічного очікування або на обмеження утримання ресурсів під час очікування інших.

Послідовність основних кроків розробленого методу уникнення взаємоблокувань завдань у розподілених системах та логічні зв'язки між ними. Метод побудований як циклічний керуючий процес із вбудованими механізмами зворотного зв'язку.

Крок 1. Формування подієво-орієнтованої моделі системи.

На початковому етапі кожен вузол формує локальну модель процесів, ресурсів і подій. Визначаються типи подій: створення процесу; запит ресурсу; виділення ресурсу; звільнення ресурсу; передача повідомлення; завершення процесу. Для кожної події фіксується логічна часова мітка та ідентифікатор процесу. Сформована подієва модель створює інформаційну основу для обчислення метрик і побудови локального графа очікування.

Крок 2. Обчислення локальних і глобальних метрик.

На основі потоку подій у вузлі розраховуються такі метрики: середній час очікування ресурсу; інтенсивність надходження запитів; коефіцієнт утримання ресурсів; довжина ланцюга залежностей процесу; локальний індекс ризику блокування. Періодично вузли обмінюються агрегованими показниками для формування глобального індексу циклічної загрози. Метрики є похідними від подієвої моделі. Результати оцінювання використовуються для прийняття рішення щодо дозволу або відкладення запиту ресурсу в межах протоколу міжпроцесної взаємодії.

Крок 3. Реєстрація наміру доступу до ресурсу.

Перед фактичним захопленням ресурсу процес ініціює подію «запит наміру» і надсилає відповідне

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

повідомлення іншим вузлам або координатору. У повідомленні містяться дані про ідентифікатор процесу, перелік утримуваних ресурсів, перелік запитуваних ресурсів, логічна мітка часу та локальні метрики. Передача наміру активується на основі поточного рівня ризику, визначеного метриками. Отримані повідомлення формують основу для узгодження порядку доступу.

Крок 4. Узгодження глобального порядку доступу.

Використовуючи часові мітки, пріоритети та показники навантаження, формується частковий або повний порядок обслуговування запитів. У системах із координатором рішення приймає центральний вузол. У децентралізованих системах застосовується алгоритм розподіленого узгодження. Порядок визначається на основі отриманих запитів наміру. Після визначення порядку виконується перевірка на можливість утворення циклу очікування.

Крок 5. Локальне та глобальне виявлення потенційних циклів.

Кожен вузол підтримує локальний граф очікування. За потреби виконується розподілена агрегація часткових графів для формування глобального графа залежностей. Використовується алгоритм пошуку циклів. Якщо цикл прогнозується або вже існує, формується сигнал ризику. Граф будується з урахуванням узгодженого порядку доступу. Результат аналізу визначає, чи буде запит дозволено, відкладено або ініційовано процедуру розв'язання конфлікту.

Крок 6. Прийняття рішення щодо доступу до ресурсу.

Можливі такі варіанти: підтвердження доступу; відкладення запиту; примусовий відкат одного з процесів; коригування пріоритетів. Рішення базується на мінімізації глобального індексу ризику та збереженні ациклічності графа очікування. Вибір дії визначається результатом перевірки на циклічність. Після виконання рішення оновлюється подієвий журнал і метрики.

Крок 7. Оновлення стану та адаптація параметрів.

Після кожної завершеної операції або зміни стану перераховуються метрики, оновлюється граф очікування, коригуються порогові значення ризику. Система переходить до наступного циклу аналізу. Оновлення відображає наслідки прийнятого рішення. Цикл замикається так: нові події знову потрапляють у подієву модель, ініціюючи наступний цикл оцінювання.

Між кроками існують прямі функціональні зв'язки, тобто результат попереднього кроку є вхідними даними наступного, та зворотні зв'язки, тобто метрики й пороги коригуються залежно від результатів розв'язання конфліктів.

Метод представляє собою замкнену систему керування доступом до ресурсів у розподіленому середовищі, де подієва модель забезпечує інформаційну повноту, метрики, тобто кількісну оцінку ризику, алгоритми виявлення забезпечують структурний контроль циклів, а протокол міжпроцесної взаємодії забезпечує механізм узгодженої реалізації рішень. Таким чином, розроблений метод інтегрує подієво-орієнтовану модель розподілених систем, систему кількісних метрик, алгоритми розподіленого виявлення циклів та адаптивний протокол координації процесів. Це створює основу для побудови масштабованих і стійких до взаємоблокувань розподілених обчислювальних середовищ.

ЕФЕКТИВНІСТЬ ТА ЕКСПЕРИМЕНТ

Методика оцінювання ефективності розробленого методу уникнення взаємоблокувань завдань у розподілених системах на основі протоколу міжпроцесної взаємодії повинна бути спрямованою не лише на емпіричну перевірку працездатності, а й на підтвердження теоретичних положень, покладених в основу запропонованого підходу. Її призначення полягає у встановленні причинно-наслідкових зв'язків між структурою графа залежностей, правилами розподілу реакторів, особливостями міжпроцесної взаємодії та кінцевими характеристиками функціонування системи. Такий підхід дозволяє розглядати оцінювання не як суто прикладне тестування, а як інструмент верифікації коректності моделі та перевірки її узгодженості з реальними умовами функціонування гетерогенних розподілених середовищ.

Розглянемо методику оцінювання ефективності розробленого методу уникнення взаємоблокувань завдань у розподілених системах на основі протоколу міжпроцесної взаємодії. Вона спрямована на формальне підтвердження коректності запропонованого підходу, кількісне вимірювання його впливу на продуктивність системи та дослідження масштабованості в умовах гетерогенного розподіленого середовища.

Оцінювання функціональної коректності здійснимо через визначення ймовірності виникнення взаємоблокування так:

$$P_{deadlock} = \frac{N_{deadlock}}{N_{runs}}, \tag{12}$$

де $P_{deadlock}$ - ймовірність виникнення взаємоблокування; $N_{deadlock}$ - кількість експериментальних запусків, у яких було зафіксовано хоча б один цикл очікування ресурсів; N_{runs} - загальна кількість незалежних експериментальних запусків системи.

Змінна $N_{deadlock}$ є критично важливою, оскільки відображає реальні випадки порушення прогресу виконання задач. Змінна N_{runs} забезпечує статистичну репрезентативність результатів, дозволяючи оцінити стійкість методу до варіацій навантаження та конфігурацій. Якщо занчення $P_{deadlock}$ прямує до нуля при

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

зростанні кількості запусків, то це підтверджує коректність протоколу міжпроцесної взаємодії.

Продуктивність системи аналізуємо через середній час виконання задач:

$$T_{exec} = \frac{\sum_{i=1}^{N_{completed}} T_i}{N_{completed}}. \quad (13)$$

де T_{exec} - середній час виконання задач; T_i - фактичний час виконання і-ї задачі від моменту ініціації до завершення; $N_{completed}$ - кількість успішно завершених задач у межах експерименту.

Змінна T_i дозволяє врахувати індивідуальні особливості виконання задач різної складності, а $N_{completed}$ характеризує фактичну результативність системи. Середній час виконання відображає вплив протоколу координації на затримки доступу до ресурсів.

Пропускну здатність визначимо так:

$$T_k = \frac{N_{completed}}{T_{total}}, \quad (14)$$

де T_k - кількість завершених задач за одиницю часу; T_{total} - загальна тривалість експериментального інтервалу; $N_{completed}$ - відображає реальний рівень корисної роботи системи в кількості задач.

Змінна T_{total} задає часову нормалізацію вимірювання, а $N_{completed}$ відображає реальний рівень корисної роботи системи.

Для оцінювання ефективності використання ресурсів вводиться коефіцієнт завантаження:

$$U = \frac{T_{base}}{T_{total}}, \quad (15)$$

де U - коефіцієнт використання ресурсу; T_{base} - сумарний час, протягом якого ресурс перебував у стані активного виконання задач; T_{total} - загальний час спостереження.

Змінна T_{base} демонструє фактичну корисну зайнятість ресурсу, а співвідношення з T_{total} дозволяє оцінити ступінь простоїв або перевантаження.

Окремо визначимо додаткові витрати протоколу міжпроцесної взаємодії так:

$$R_v = \frac{M_{protocol}}{M_{total}}, \quad 16)$$

де R_v - частка службового трафіку; $M_{protocol}$ - кількість або обсяг службових повідомлень, що генеруються протоколом координації; M_{total} - загальний обсяг переданих повідомлень у системі.

Змінна $M_{protocol}$ відображає витрати на підтримання узгодженості та контроль циклів, тоді як M_{total} характеризує повне комунікаційне навантаження. Аналіз цього співвідношення дозволяє оцінити наявність потенційного перевищення службових витрат допустимого рівня.

Для комплексної інтегральної оцінки застосовується узагальнений показник ефективності:

$$E_{total} = w_1 \cdot (1 - P_{deadlock}) + w_2 \cdot \frac{T_k}{T_{k,max}} + w_3 U - w_4 R_v, \quad (17)$$

де E_{total} - інтегральний показник ефективності; R_v - частка службового трафіку; U - коефіцієнт використання ресурсу; w_1, w_2, w_3, w_4 вагові коефіцієнти, що відображають пріоритетність відповідних критеріїв; T_k - кількість завершених задач за одиницю часу; $T_{k,max}$ - максимальне зафіксоване значення пропускної здатності в контрольному експерименті.

Вагові коефіцієнти дозволяють адаптувати оцінювання до специфіки застосування системи, наприклад, надаючи більшої ваги гарантованій відсутності взаємоблокувань або мінімізації накладних витрат.

Таким чином, розширена методика забезпечує всебічний аналіз розробленого методу, дозволяє кількісно підтвердити його коректність, оцінити вплив на продуктивність і встановити межі масштабованості. Деталізація змінних у формулах гарантує прозорість інтерпретації результатів та відтворюваність експериментів, що є необхідною умовою наукової обґрунтованості дослідження.

На рис. 2 - рис. 5 зображені графіки результатів експериментального дослідження ефективності розробленого методу уникнення взаємоблокувань у розподілених системах на основі протоколу міжпроцесної взаємодії. Візуалізація охоплює чотири ключові групи показників: ймовірність виникнення взаємоблокувань; середній час виконання задач; частку протокольних додаткових витрат; пропускну здатність системи залежно від масштабування. Сукупний аналіз цих залежностей дозволяє комплексно оцінити функціональну коректність, продуктивність та масштабованість запропонованого підходу.

Графік на рис. 1 відображає залежність ймовірності виникнення взаємоблокування $P_{deadlock}$ від кількості незалежних запусків системи N_{runs}. Крива для запропонованого методу демонструє стрімке зниження показника на початкових етапах експерименту з подальшою стабілізацією на рівні, близькому до нуля. Така динаміка має принципове значення. Вона свідчить про те, що зі зростанням статистичної вибірки

підтверджується системна властивість протоколу, яка полягає у відсутності формування циклів очікування за умов дотримання визначених правил взаємодії між реакторами. Початкові ненульові значення пояснюються перехідними процесами, що виникають у фазі ініціалізації або при моделюванні граничних сценаріїв конкуренції ресурсів. Однак подальша стабілізація кривої на мінімальному рівні підтверджує, що метод не просто зменшує частоту взаємоблокувань, а фактично запобігає їх системному виникненню.

Для порівняння, крива класичного підходу виявлення взаємоблокувань демонструє суттєво вищий рівень $P_{deadlock}$. Це пояснюється тим, що механізм виявлення та відновлення реагує вже на сформований цикл, а не запобігає його появі. Тому в певній частці експериментів система переходить у стан блокування, який потім усувається процедурою відновлення. Ще більш виражений негативний результат спостерігається для системи без механізму уникнення, де ймовірність залишається стабільно високою та майже не залежить від кількості запусків, що вказує на структурну схильність до формування циклів очікування.

Графік на рис. 3 демонструє середній час виконання задач T_{exec} за умов низького, середнього та високого навантаження. Для розробленого методу спостерігається помірне зростання часу виконання зі збільшенням навантаження, що є природним наслідком підвищеної конкуренції за ресурси. Водночас навіть за високого навантаження значення T_{exec} залишається нижчим порівняно з альтернативними підходами. Це означає, що протокол міжпроцесної взаємодії не створює надмірних синхронізаційних затримок і не призводить до каскадного накопичення очікувань.

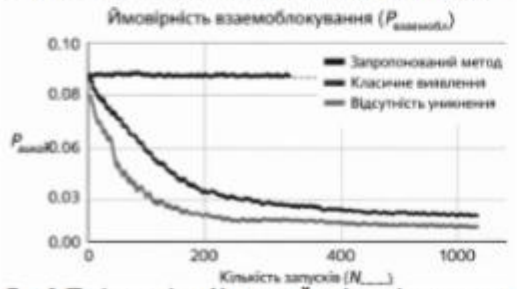


Рис. 2. Порівняльні графіки щодо ймовірності виникнення взаємоблокувань

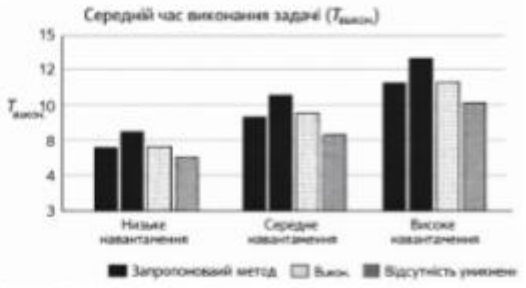


Рис. 3. Порівняльні графіки щодо середнього часу виконання завдань

У класичному методі виявлення взаємоблокувань середній час виконання зростає більш інтенсивно. Причиною цього є додаткові витрати на аналіз глобального стану та процедури відновлення після виявлення циклу. Система без механізму уникнення демонструє ще гірші результати, оскільки періодичні блокування спричиняють значні простої. Таким чином, другий графік підтверджує, що запропонований метод забезпечує не лише коректність, а й підвищену часову ефективність.

Графік на рис. 4 зображає залежність частки протокольних додаткових витрат R_p від кількості вузлів N. Для запропонованого методу спостерігається плавне зростання додаткових витрат із подальшою стабілізацією. Це свідчить про те, що механізм координації масштабуються передбачувано та не призводить до експоненційного зростання службового трафіку. Фактично протокол забезпечує локалізацію взаємодії та мінімізує глобальні синхронізації.

Натомість централізований контроль демонструє стрімке зростання додаткових витрат із масштабуванням системи. Це пов'язано з концентрацією запитів у центральному вузлі, що створює комунікаційне перевантаження. Часова стратегія має менші додаткові витрати на початкових етапах, проте зі збільшенням кількості вузлів частота повторних запитів і перезапусків зростає, що також призводить до накопичення службового трафіку. Отже, графік на рис. 4 підтверджує структурну перевагу розподіленого протоколу координації.

Графік на рис. 5 відображає залежність пропускної здатності системи від кількості вузлів. Для запропонованого методу спостерігається майже лінійне зростання продуктивності зі збільшенням числа вузлів, що свідчить про ефективну масштабованість. Кожне додавання нового вузла приводить до пропорційного зростання кількості виконаних задач за одиницю часу. Це означає, що координаційні механізми не обмежують паралелізм і не створюють вузьких місць.

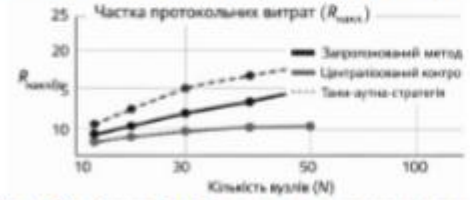


Рис. 4. Порівняльні графіки щодо частки протокольних витрат

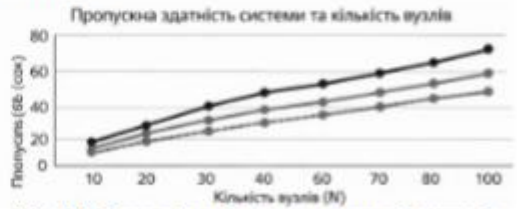


Рис. 5. Порівняльні графіки щодо пропускної здатності системи

Міжнародний науково-технічний журнал
«Вимірювальна та обчислювальна техніка в технологічних процесах»
ISSN 2219-9365

У випадку централізованого або реактивного підходу масштабування є менш ефективним. Пропускна здатність зростає повільніше, що свідчить про накопичення координаційних витрат або періодичні втрати продуктивності через взаємоблокування.

Підсумкова табл. 1 узагальнює кількісні значення основних показників. Значення $P_{deadlock}$, близьке до нуля для запропонованого методу, підтверджує функціональну коректність. Найменший середній час виконання задач та найбільша пропускна здатність демонструють ефективність з точки зору продуктивності. Водночас частка накладних витрат залишається контрольованою і значно нижчою, ніж у централізованих підходах.

Таблиця 1

Загальні кількісні показники

Показник	Розроблений	Класичне виявлення	Відсутність уникнення
$P_{deadlock}$	0,002	0,056	0,081
Середній час виконання	5,2 с	8,4 с	11,5 с
Пропускна здатність	45,6	38,2	29,5
Частка витрат	6,7 %	15,4 %	20,3 %

У сукупності результати графічного та табличного аналізу підтверджують гіпотезу про те, що інтеграція подієво-орієнтованої моделі розподілу задач із формалізованим протоколом міжпроцесної взаємодії дозволяє забезпечити структурну відсутність циклів очікування без втрати масштабованості. Запропонований метод демонструє стабільність у широкому діапазоні навантажень, передбачуване зростання накладних витрат і лінійну динаміку продуктивності при розширенні системи. Це свідчить про його придатність до застосування в сучасних гетерогенних розподілених обчислювальних середовищах, де поєднуються вимоги високої надійності, масштабованості та ефективності використання ресурсів.

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Запропонований підхід формує цілісну методологічну основу керування розподіленими обчисленнями, у якій реактор виступає ключовим елементом поєднання структурної моделі залежностей і механізмів міжпроцесної взаємодії. Інтеграція стратегії розподілу реакторів із формалізованим протоколом обміну повідомленнями забезпечує не лише теоретично обґрунтоване уникнення взаємоблокувань, а й практичну гарантованість просування виконання критичних шляхів у складних гетерогенних середовищах.

Комплексна система метрик створює кількісну основу для оцінювання стану системи, вибору алгоритмів планування, балансування та масштабування, а алгоритми виявлення циклів разом із адаптивним протоколом координації забезпечують структурний і процедурний контроль доступу до ресурсів.

Розроблений метод уникнення взаємоблокувань завдань в розподілених системах представляє собою замкнену, узгоджену систему керування, що поєднує формальну модель, аналітичний інструментарій і механізми практичної реалізації, формуючи підґрунтя для побудови масштабованих, ефективних і стійких до взаємоблокувань розподілених обчислювальних середовищ.

Напрямами подальших досліджень є адаптація розробленого методу для розподілених систем з різними типами вузлів та збільшенням кількості вузлів.

References

1. Alzalab E., Abubakar U. E. H., Li Z. El-Meligy M., El-Sherbeeny A. Modeling of fault recovery and repair for automated manufacturing cells with load-sharing redundant elements using Petri nets. *Processes*. 2023, 11, 1501. https://doi.org/10.3390/pr11051501

2. Zhou Q., Chai B., Ran K., Guo Y., Zhou S., Wu W., Wang K., Ni Y. Research on a Multi-Dimensional Information Fusion Mechanical Wear Fault-Diagnosis Algorithm Based on Data Regeneration. *Sensors*. 2025, 25, 3745. https://doi.org/10.3390/s25123745

3. Chuang W., Tseng C., Tan K., Pan Y. Design of a novel transition-based deadlock recovery policy for flexible manufacturing systems. *Processes*. 2025, 13, 1610. https://doi.org/10.3390/pr13051610

4. Chen C., Hu H. Extended Place-Invariant Control in Automated Manufacturing Systems Using Petri Nets. *IEEE Trans. Syst. Man Cybern. Syst.* 2020, 52, 1807–1822.

5. Kaid H., Al-Ahmari A., Li Z., Davidrajuh R. Single controller-based colored petri nets for deadlock control in automated manufacturing systems. *Processes*. 2020, 8, 21. https://doi.org/10.3390/pr8010021

6. Feng Y., Ren S., Cao Y., Xing K., Yang Y. Deadlock control for flexible assembly systems with multiple resource requirements and separately-loaded parts. *IEEE Trans. Autom. Sci. Eng.* 2024, 22, 9275–9284. https://doi.org/10.1109/TASE.2024.3504714

7. Hu S., Li Z., Zhang Z. Design of online supervisors for enforcing diagnosability in Petri nets with unknown initial markings. *IEEE Internet Things J.* 2025, 12, 11108–11120. https://doi.org/10.1109/JIOT.2024.3515194

8. Lin X., Zhang Y., Gao Y., Chi X. Automatic construction of Petri net models for computational simulations of molecular inter-action network. *Syst. Biol. Appl.* 2024, 10, 131. https://doi.org/10.1038/s41540-024-00464-z

9. Hu H., Zhou M. A Petri net-based discrete event control of automated manufacturing systems with assembly operations. *IEEE Trans. Control Syst. Technol.* 2015, 23, 513–524. https://doi.org/10.1109/TCST.2014.2342664

10. Xing K., Wang F., Zhou M., Lei H., Luo J. Deadlock characterization and control of flexible assembly systems with Petri nets. *Automatica*. 2018, 87, 358–364. https://doi.org/10.1016/j.automatica.2017.09.001

11. Liu G., Li Z., Barkaoui K., Al-Ahmari A. Robustness of deadlock control for a class of Petri nets with unreliable resources. *Inf. Sci.* 2013, 235, 259–279. https://doi.org/10.1016/j.ins.2013.01.003

12. Cong, X. Y., Gu, C., Uzam, M., Chen, Y. F., Al-Ahmari, A. M., Wu, N. Q., Zhou, M. C., & Li, Z. W. (2018). Design of Optimal Petri Net Supervisors for Flexible Manufacturing Systems via Weighted Inhibitor Arcs. *Asian Journal of Control*, *20*(1), 511-

530. https://doi.org/10.1002/asjc.1583

13. Li, Z. W., & Zhou, M. C. (2004). Elementary Siphons of Petri Nets and Their Application to Deadlock Prevention in Flexible Manufacturing Systems. *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans.*, *34*(1), 38-51. https://doi.org/10.1109/TSMCA.2003.820576

14. Huang Y., Jeng M., Xie X., Chung S. A deadlock prevention pollicy for flexible manufacturing systems using siphons. *In Proceedings of the IEEE International Conference on Robotics and Automation*, Seoul, Korea, 21–26 May 2001; pp. 541–546.

15. Piroddi L., Cordone R., Fumagalli I. Selective siphon control for deadlock preveention in Petri Nets. *IEEE Trans. Syst. Man Cybern. A. Syst. Hum.* 2008, 38, 1337–1348.

16. Bedratyuk L., Savenko O., *MATCH Commun. Math. Comput. Chem.* 2018, 79, 407–414. URL: https://match.pmf.kg.ac.rs/electronic_versions/Match79/n2/match79n2_407-414.pdf

17. Ezpeleta J., Colom J., Martinez J. A petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Trans. Robot. Autom.* 1995, 11, 173–184. https://doi.org/10.1109/70.370500

18. Liu G., Li Z., Zhong C. New controllability condition for siphons in a class of generalised Petri nets. *IET Control Theory Appl.* 2010, 4, 854–864. https://doi.org/10.1049/iet-cta.2009.0264

19. Uzam M. An Optimal Deadlock Prevention Policy for Flexible Manufacturing Systems Using Petri Net Models with Resources and the Theory of Regions. *Int. J. Adv. Manuf. Technol.* 2002, 19, 192–208. https://doi.org/10.1007/s001700200014

20. Liao H., Wang Y., Stanley J., Lafortune S., Reveliotis S., Kelly T., Mahlke S. Eliminating concurrency bugs in multithreaded software: A new approach based on discrete-event control. *IEEE Trans. Control Syst. Technol.* 2013, 21, 2067–2082. https://doi.org/10.1109/TCST.2012.2226034

21. Chen Y., Li W., Khalgui M., Mosbahi O. Design of a maximally permissive liveness enforcing Petri net supervisor for flexible manufacturing systems. *IEEE Trans. Autom. Sci. Eng.* 2011, 8, 374–393. https://doi.org/10.1109/TASE.2010.2060332

ДОДАТОК В

(обов’язковий)

Програмний код

Глобальний планувальник на C++, який включає:

1. Управління ресурсами (алгоритм банкіра)
2. Пріоритетне планування завдань
3. Векторні часові мітки (Lamport Clock)
4. Протокол двофазного блокування (2PL)
5. Виявлення deadlock розподіленим алгоритмом
6. Динамічну міграцію завдань
7. IPC через TCP–сокети для обміну повідомленнями між GlobalScheduler та WorkerNode

Структура проєкту

GlobalScheduler/

├── main.cpp // точка входу, ініціалізація планувальника
├── Task.h // структура завдання, пріоритет, потреби
├── ResourceManager.h // алгоритм банкіра та 2PL
├── LamportClock.h // векторний годинник
├── MigrationManager.h // динамічна міграція задач
├── ProtocolHandler.h // Chandy–Misra–Haas, IPC
├── WorkerNode.cpp // приклад вузла, що надсилає запити

Інтегрована модель даних

```
#include <vector>
#include <map>
#include <mutex>
#include <iostream>
#include <algorithm>

struct Task {
    int id;
    int priority;
    std::vector<int> maxDemand;
    std::vector<int> allocation;
    std::vector<int> need;
    int nodeId; // Node на якому виконується
};
```

```
 Lamport Clock
class LamportClock {
private:
    int time;
public:
    LamportClock() : time(0) {}
    int tick() { return ++time; }
    void update(int receivedTime) {
        time = std::max(time, receivedTime) + 1;
    }
    int getTime() const { return time; }
};

ResourceManager + 2PL + Safe State
class ResourceManager {
private:
    std::vector<int> available;
    std::vector<Task*> tasks;
    std::mutex mtx;
    std::map<int, bool> locks; // Two-phase locking

public:
    ResourceManager(const std::vector<int>& resources) :
available(resources) {}

    void registerTask(Task* task) {
        tasks.push_back(task);
        locks[task->id] = false;
    }

    bool lockResource(int taskId) {
        std::lock_guard<std::mutex> lock(mtx);
        if (locks[taskId]) return false;
        locks[taskId] = true;
        return true;
    }

    void releaseResource(int taskId) {
        std::lock_guard<std::mutex> lock(mtx);
        locks[taskId] = false;
    }

    bool requestResources(Task* task, const std::vector<int>&
request) {
        std::lock_guard<std::mutex> lock(mtx);
```

```
            for (size_t i = 0; i < request.size(); i++) {
                if (request[i] > task->need[i] || request[i] >
available[i])
                    return false;
            }

            for (size_t i = 0; i < request.size(); i++) {
                available[i] -= request[i];
                task->allocation[i] += request[i];
                task->need[i] -= request[i];
            }

            if (!isSafeState()) {
                for (size_t i = 0; i < request.size(); i++) {
                    available[i] += request[i];
                    task->allocation[i] -= request[i];
                    task->need[i] += request[i];
                }
                return false;
            }

            return true;
        }

        bool isSafeState() {
            std::vector<int> work = available;
            std::vector<bool> finish(tasks.size(), false);

            bool progress;
            do {
                progress = false;
                for (size_t i = 0; i < tasks.size(); i++) {
                    if (!finish[i]) {
                        bool canFinish = true;
                        for (size_t j = 0; j < work.size(); j++) {
                            if (tasks[i]->need[j] > work[j]) {
                                canFinish = false;
                                break;
                            }
                        }
                        if (canFinish) {
                            for (size_t j = 0; j < work.size(); j++)
                                work[j] += tasks[i]->allocation[j];
                            finish[i] = true;
                            progress = true;
                        }
```

```
                    }
                }
            } while (progress);

            return std::all_of(finish.begin(), finish.end(), [](bool
f){ return f; });
        }
    };

     Пріоритетне планування
    Task* selectHighestPriorityTask(const std::vector<Task*>& tasks)
{
        Task* best = nullptr;
        for (auto t : tasks) {
            if (!best || t->priority > best->priority)
                best = t;
        }
        return best;
    }

    Динамічна міграція задач
    class MigrationManager {
    public:
        void migrateTask(Task* task, int toNode) {
            int fromNode = task->nodeId;
            task->nodeId = toNode;
            std::cout << "Task " << task->id << " migrated from Node
"
                      << fromNode << " to Node " << toNode << "\n";
        }
    };

     Chandy-Misra-Haas (розподілене виявлення блокування завдань)
    struct Probe {
        int initiatorTaskId;
        int waitingTaskId;
        int resourceId;
    };

    class ProtocolHandler {
    public:
        void sendProbe(const Probe& probe) {
            // тут має бути реалізація через TCP / ZeroMQ
            std::cout    <<    "Probe    sent:    initiator    "    <<
probe.initiatorTaskId
                      << " waiting " << probe.waitingTaskId << "\n";
```

```
        }
    };

     Приклад інтеграції в main.cpp
    int main() {
        std::vector<int> totalResources = {10, 5, 7};
        ResourceManager manager(totalResources);

        LamportClock clock;
        MigrationManager migration;
        ProtocolHandler protocol;

        Task t0{0, 2, {7,5,3}, {0,0,0}, {7,5,3}, 1};
        Task t1{1, 1, {3,2,2}, {0,0,0}, {3,2,2}, 1};
        Task t2{2, 3, {9,0,2}, {0,0,0}, {9,0,2}, 2};

        manager.registerTask(&t0);
        manager.registerTask(&t1);
        manager.registerTask(&t2);

        Task* next = selectHighestPriorityTask({&t0,&t1,&t2});
        std::vector<int> request = {3,0,2};

        clock.tick();
        if(manager.lockResource(next->id)                              &&
manager.requestResources(next, request)) {
            std::cout << "Resources granted to Task " << next->id <<
"\n";
        } else {
            std::cout << "Request denied for Task " << next->id <<
"\n";
            Probe p{next->id, -1, -1};
            protocol.sendProbe(p);
        }
        manager.releaseResource(next->id);

        migration.migrateTask(&t1, 2);
    }


    Структура проєкту з TCP IPC
    DistributedScheduler/
    ├── GlobalScheduler.cpp     // TCP-сервер + ResourceManager +
ProtocolHandler
    ├── WorkerNode.cpp        // TCP-клієнт + Task
    ├── Task.h                // структура Task
```

```
├── ResourceManager.h     // алгоритм банкіра + 2PL
├── LamportClock.h
├── MigrationManager.h
├── ProtocolHandler.h
```

Формат повідомлень

Використано текстовий протокол для простоти:

REQUEST <taskId> <priority> <r1> <r2> <r3> ...

RELEASE <taskId> <r1> <r2> <r3> ...

RESPONSE GRANT|DENY

r1 r2 r3 – кількість ресурсів, які запитується / звільняється

GlobalScheduler (TCP-сервер)

```
#include <iostream>
#include <thread>
#include <vector>
#include <string>
#include <sstream>
#include <netinet/in.h>
#include <unistd.h>
#include "ResourceManager.h"
#include "Task.h"
#include "LamportClock.h"

#define PORT 54000

ResourceManager manager({10,5,7});
LamportClock clock;

void handleClient(int clientSocket) {
    char buffer[1024];
    while (true) {
        int bytesRead = read(clientSocket, buffer, sizeof(buffer)-
1);
        if (bytesRead <= 0) break;
        buffer[bytesRead] = '\0';

        std::istringstream iss(buffer);
        std::string cmd;
        iss >> cmd;

        if(cmd == "REQUEST") {
            int taskId, priority;
            iss >> taskId >> priority;
            std::vector<int> request;
            int r; while (iss >> r) request.push_back(r);
```

```
                clock.tick();

                Task t{taskId, priority, request,
std::vector<int>(request.size(),0), request, 1};
                manager.registerTask(&t);

                std::string response;
                if(manager.lockResource(taskId) &&
manager.requestResources(&t, request)) {
                    response = "GRANT\n";
                    manager.releaseResource(taskId);
                } else {
                    response = "DENY\n";
                }

                send(clientSocket, response.c_str(), response.size(),
0);
            }
        }
        close(clientSocket);
    }

    int main() {
        int server_fd, clientSocket;
        struct sockaddr_in address;
        int opt = 1;
        int addrlen = sizeof(address);

        server_fd = socket(AF_INET, SOCK_STREAM, 0);
        setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR |
SO_REUSEPORT, &opt, sizeof(opt));
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = INADDR_ANY;
        address.sin_port = htons(PORT);

        bind(server_fd, (struct sockaddr*)&address, sizeof(address));
        listen(server_fd, 5);

        std::cout << "GlobalScheduler listening on port " << PORT <<
"\n";

        while(true) {
            clientSocket = accept(server_fd, (struct
sockaddr*)&address, (socklen_t*)&addrlen);
            std::thread(handleClient, clientSocket).detach();
        }
```

```
        return 0;
    }
    WorkerNode (TCP-клієнт)
    #include <iostream>
    #include <sys/socket.h>
    #include <arpa/inet.h>
    #include <unistd.h>
    #include <string>
    #include <vector>
    #include <sstream>

    #define PORT 54000

    void sendRequest(int taskId, int priority, const std::vector<int>&
request) {
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in serverAddr;
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(PORT);
        inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);

        connect(sock,          (struct          sockaddr*)&serverAddr,
sizeof(serverAddr));

        std::ostringstream oss;
        oss << "REQUEST " << taskId << " " << priority;
        for(int r : request) oss << " " << r;
        oss << "\n";

        std::string msg = oss.str();
        send(sock, msg.c_str(), msg.size(), 0);

        char buffer[1024];
        int bytesRead = read(sock, buffer, sizeof(buffer)-1);
        buffer[bytesRead] = '\0';

        std::cout << "Task " << taskId << " received response: " <<
buffer;
        close(sock);
    }

    int main() {
        sendRequest(1, 2, {3,0,2});
        sendRequest(2, 1, {2,1,1});
        sendRequest(3, 3, {4,2,2});
```

```
}
```

Особливості інтеграції:

1, GlobalScheduler обробляє одночасно кілька WorkerNode через потоки (std::thread)

2. Використовується LamportClock для відстеження подій

3. ResourceManager забезпечує safe state + 2PL

4. Пріоритетне планування можна інтегрувати, сортувавши чергу запитів перед виділенням ресурсів

5. Алгоритм можна додати при відхиленні запиту відправляти probe message іншим вузлам

6. Динамічна міграція можлива через окремий IPC-повідомлення MIGRATE <taskId> <toNode>

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Владислав КРЕЩУК

Співавтор:

Назва: Метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії

Експерт: Богдан САВЕНКО

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:5.74%

Коефіцієнт подібності 2:1.21%

Мікропробіли: 165

Заміна букв: 2

Інтервали: 0

Білі знаки: 1

Дата створення звіту: 2026-04-20 19:54:20.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2026-04-20 Доцент Андрій Нічепорук

Дата експерт

Mon Apr 20 20:17:08 EEST 2026, Медзатий Дмитро Миколайович, Хмельницький національний університет,

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 29.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

	Документ		Сумарний збіг по Базі Даних	
ID: 270564 Назва: МКР Метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії Додано в БД: 2026-04-20 Автора: Владислав КРЕЩУК Керівники: Богдан САВЕНКО Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	149663	1129	43973 (29%)	332 (29%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
269837	Назва: Звіт з НДП Метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії Додано в БД: 2026-03-17 Автора: Крещука В.О. Керівники: Павлова О.О. Консультанти: Опоненти:	43181 (29.0%)	329 (29.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Здобувач: Владислав КРЕЩУК

Тема: Метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи магістра:

Кількість листів креслень – ; кількість сторінок записки 73

1. Короткий зміст роботи та прийнятих рішень У роботі запропоновано метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії

2. Висновок про відповідність роботи дипломному завданню Кваліфікаційна робота магістра відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі подано об'єкт та предмет дослідження, мету, наукову новизну та практичну цінність роботи, а також характеристику структури роботи.

У першому розділі проведено аналіз відомих рішень щодо уникнення взаємоблокувань завдань та взаємодії процесів в розподілених системах на основі протоколу міжпроцесної взаємодії.

У другому розділі здійснено дослідження предметної області та запропоновано архітектуру планувальника для процесів в розподілених системах на основі протоколу міжпроцесної взаємодії.

У третьому розділі розроблено метод уникнення взаєблокувань завдань в розподілених системах представляє собою замкнену, узгоджену систему керування, що поєднує формальну модель, аналітичний інструментарій і механізми практичної реалізації, формуючи підґрунтя для побудови масштабованих, ефективних і стійких до взаємоблокувань розподілених обчислювальних середовищ.

У четвертому розділі здійснено розроблення архітектури системи, яка забезпечує повний цикл управління завданнями, тобто від надходження запиту до виділення ресурсів, синхронізації подій, контролю блокувань, обробки пріоритетів і динамічної міграції задач, що дозволяє моделювати реальні розподілені системи та експериментувати з різними стратегіями управління ресурсами.

У висновках підведено підсумки досягнення результатів з розв'язання завдань дослідження.

4. Позитивні сторони роботи: рзроблений метод дозволяє автоматизувати процес застосування політик безпеки без необхідності постійного ручного адміністрування, забезпечити безперервний контроль відповідності функціонування системи формалізованим критеріям безпеки, зменшити ризик виникнення інцидентів, спричинених помилками конфігурації, скоротити час виявлення та локалізації порушень політик доступу,

5. Негативні сторони роботи:

· 6. Оцінка графічного оформлення та пояснювальної записки роботи: __–____

7. Відгук про роботу в цілому: __В загальному робота виконана на високому рівні.

8. Інші зауваження: ___–

9. Оцінка кваліфікаційної роботи магістра:

Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи магістра вважаю, що робота заслуговує оцінки «відмінно» 90.00 (А)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) ____ д.т.н., професор, Мартинюк В.В., професор кафедри автоматизації, комп'ютерно-інтегрованих технологій та робототехніки

"__1 травня__"_____2026р. _____

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Владислав КРЕЩУК

ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-24-2

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1 травня 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Метод уникнення взаємоблокувань завдань в розподілених системах на основі протоколу міжпроцесної взаємодії
Автор Владислав КРЕЩУК
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: д.ф. Богдан САВЕНКО

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;

3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 5,74% та системою Anti-Plagiarism складає 0%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

30.04.2026

Завідувач кафедри _____ Підпис — Ольга ПАВЛОВА Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Підпис — Олег САВЕНКО Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Підпис — Богдан САВЕНКО Ім'я, ПРІЗВИЩЕ