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

КВАЛІФІКАЦІЙНА РОБОТА

Метод послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника
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 КвРКІ 240122. 22. 01.08 ПЗ

Виконав здобувач IV курсу, група КІ2м-24-1 _____ Євгеній СЕМЕНЮК
Підпис Ініціали, прізвище

Керівник канд.-техн. наук, доцент _____ Олег САВЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер канд. фіз.-мат. наук, доцент _____ Тетяна КИСІЛЬ
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КІІС _____ Ольга ПАВЛОВА
«01» 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 КПІС
_____ Ольга ПАВЛОВА

"12" 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Семенюк Євгеній Валерійович
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Метод послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника
Керівник проекту (роботи) Савенко Олег Станіславович, д.т.н., проф.
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)
Метод послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника
Проєктування методу послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника
Програмно-апаратна метод послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника

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

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	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	15.01.2026	виконано
3	Робота над розділом 1 - дослідження теоретичної основи та постановка задачі	01.02.2026	виконано
4	Робота над розділом 2 - розроблення методу послідовного розподілу ресурсів	01.03.2026	виконано
5	Робота над розділом 3 - алгоритмічне та програмне забезпечення реалізації методу	29.03.2026	виконано
6	Робота над розділом 4 - програмна реалізація та експериментальна перевірка		
7	Оформлення пояснювальної записки згідно вимог	25.04.2026	виконано
8	Попередній захист ВКР	26.04.2025	виконано
9	Захист ВКР на засіданні ЕК	травень 2026 року	

Здобувач _____ Євгеній СЕМЕНЮК
Підпис Ім'я, ПРІЗВИЩЕ

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

РЕФЕРАТ

Тема Метод послідовного розподілу ресурсів у багатоагентних комп’ютерних системах на основі справедливого планувальника

Автор роботи: Семенюк Євгеній Валерійович

Керівник роботи: Савенко Олег Станіславович

Пояснювальна записка: 82 с., 7 рис., 1 дод., 80 джерел.

БАГАТОАГЕНТНА СИСТЕМА, РОЗПОДІЛ РЕСУРСІВ, СПРАВЕДЛИВИЙ ПЛАНУВАЛЬНИК, ДОМІНАНТНА ЧАСТКА, БОРГ І КРЕДИТ, БАГАТОРЕСУРСНЕ СЕРЕДОВИЩЕ, ЗАЯВКА, ПЛАНУВАННЯ, ОБЧИСЛЮВАЛЬНІ РЕСУРСИ, ЗДІЙСНЕННІСТЬ ЗАПУСКУ.

Об’єктом роботи є процеси розподілу обчислювальних ресурсів у багатоагентних комп’ютерних системах за умов динамічного навантаження.

Предметом роботи є методи та алгоритми послідовного розподілу ресурсів на основі справедливого планувальника з урахуванням багаторесурсних обмежень і часових відхилень обслуговування агентів.

Метою роботи є розроблення методу послідовного розподілу ресурсів у багатоагентних комп’ютерних системах на основі справедливого планувальника, який забезпечує збалансований доступ агентів до ресурсів, обмежує накопичення перекосів у часі та зберігає прикладну працездатність за наявності багаторесурсних обмежень.

Для розв’язання поставлених завдань було використано методи аналізу багатоагентних систем, методи теорії планування, методи формалізації багаторесурсного середовища, методи моделювання розподілу ресурсів, алгоритмічні методи обчислення домінантної частки, а також методи програмної реалізації та експериментальної перевірки результатів.

Наукова новизна отриманих результатів полягає в тому, що:

- розроблено метод послідовного розподілу ресурсів у багатоагентних комп’ютерних системах, який, на відміну від відомих підходів, поєднує оцінювання багаторесурсного стану агента на основі домінантної частки з

механізмом компенсації часових відхилень у формі боргу і кредиту та перевіркою здійсненності запуску заявки;

- удосконалено підхід до вибору заявки на обслуговування шляхом інтеграції критерію справедливості з фактичними обмеженнями ресурсного середовища.

Практична значимість отриманих результатів полягає у створенні програмного засобу для моделювання послідовного розподілу ресурсів у багатоагентному середовищі, який може бути використаний для аналізу поведінки планувальників, дослідження політик розподілу ресурсів і перевірки ефективності різних стратегій обслуговування заявок.

У вступі подано обґрунтування актуальності теми, об’єкт і предмет роботи, мету, завдання, методи, наукову новизну та практичну значимість, а також характеристику структури роботи.

У першому розділі проведено аналіз предметної області, розглянуто багатоагентні комп’ютерні системи, підходи до справедливості в задачах розподілу ресурсів, особливості справедливих планувальників і сформульовано постановку задачі.

У другому розділі виконано формалізацію багатоагентного середовища та ресурсних обмежень, обґрунтовано критерії справедливості, побудовано модель домінантної частки, розроблено механізм компенсації часових відхилень і сформовано узагальнений метод послідовного розподілу ресурсів.

У третьому розділі розроблено архітектуру програмного засобу, алгоритм роботи справедливого планувальника, алгоритми обчислення домінантної частки та оновлення стану агентів, а також описано програмну реалізацію основних модулів системи.

У четвертому розділі реалізовано програмний засіб і експериментальний стенд, налаштовано сценарії моделювання навантаження, проведено аналіз результатів роботи системи за різних режимів, виконано порівняння з базовими підходами розподілу ресурсів та оцінено ефективність запропонованого рішення.

У висновках узагальнено основні результати роботи та підтверджено досягнення поставленої мети.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

CPU - центральний - процесор

RAM - оперативна пам’ять

GPU - графічний процесор

TPU - тензорний процесор

I/O - операції введення-виведення

QoS - якість обслуговування

API - прикладний програмний інтерфейс

SLA - угода про рівень обслуговування

LLM - велика мовна модель

FIFO - обслуговування в порядку надходження

RR - Round Robin, циклічне обслуговування

WFQ - Weighted Fair Queuing, зважене справедливе обслуговування

EDF - Earliest Deadline First, обслуговування за найранішим дедлайном

ВСТУП

Сучасні багатоагентні комп’ютерні системи активно використовуються в умовах обмежених обчислювальних ресурсів, де одночасно виконуються численні задачі з різними вимогами до процесорного часу, пам’яті та інших типів ресурсів. Базові підходи до розподілу ресурсів у таких системах часто розроблялися без урахування складної багаторесурсної природи середовища та динамічного характеру надходження заявок. Вони продовжують широко застосовуватися, однак у сучасних умовах це призводить до накопичення перекосів між агентами, неефективного використання ресурсів і зростання часу очікування.

У реальних умовах функціонування багатоагентних середовищ окремі агенти можуть отримувати непропорційний доступ до ресурсів, що негативно впливає на загальну стабільність системи. Наприклад, при нерівномірному надходженні заявок або за наявності дефіциту окремих ресурсів частина задач може тривалий час залишатися без обслуговування, тоді як інші отримують перевагу. Це створює необхідність удосконалення підходів до планування, зокрема впровадження механізмів, які забезпечують більш справедливий і збалансований розподіл ресурсів у часі.

Перспективним напрямом є використання справедливих планувальників, які враховують як поточний стан використання ресурсів, так і історію обслуговування агентів. Розширення практики використання багатоагентних систем підкреслює потребу в ефективних методах розподілу ресурсів, які забезпечують як справедливість, так і стабільність роботи системи в умовах змінного навантаження.

Актуальність роботи полягає в розробці методу послідовного розподілу ресурсів у багатоагентних комп’ютерних системах на основі справедливого планувальника.

Метою роботи є покращення ефективності розподілу ресурсів у багатоагентних комп’ютерних системах шляхом розроблення методу, який забезпечує справедливий доступ агентів до ресурсів з урахуванням багаторесурсних обмежень і часових відхилень.

Поставлена мета досягається розв’язанням таких основних завдань:

– проаналізувати відомі підходи до розподілу ресурсів у багатоагентних системах та справедливі планувальники;

– розробити метод послідовного розподілу ресурсів на основі багаторесурсної оцінки та механізму компенсації відхилень;

Об’єктом роботи є процес розподілу обчислювальних ресурсів у багатоагентних комп’ютерних системах.

Предметом роботи є методи та алгоритми послідовного розподілу ресурсів на основі справедливого планувальника.

Наукова новизна отриманих результатів полягає в тому, що:

– розроблено метод послідовного розподілу ресурсів, у якому, на відміну від відомих підходів, поєднано оцінювання домінантної частки агента з механізмом боргу і кредиту та перевіркою здійсненності запуску;

– удосконалено підхід до вибору задачі на обслуговування за рахунок урахування як поточного стану системи, так і накопичених часових відхилень.

На основі виконаної роботи було розроблено метод і програмний засіб для моделювання розподілу ресурсів у багатоагентному середовищі.

Практична значимість отриманих результатів полягає у створенні програмного засобу, який може бути використаний для аналізу та оптимізації процесів розподілу ресурсів у багатоагентних комп’ютерних системах.

Для розв’язання поставлених задач використовувалися методи теорії планування, методи моделювання багатоагентних систем, алгоритмічні методи обчислення показників справедливості, а також методи програмної реалізації та експериментального дослідження.

За темою кваліфікаційної роботи опубліковано одну публікацію [81] у Збірнику наукових праць за матеріалами XIX Всеукраїнська науково-практична web конференція аспірантів, студентів та молодих вчених комп’ютерні інтелектуальні системи та мережі. (Кривий ріг - 2026. с. 31-32).

1. ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

1.1 Загальна характеристика багатоагентних комп'ютерних систем і ресурсних обмежень

Багатоагентні комп'ютерні системи розглядаються як клас розподілених обчислювальних середовищ, у яких одночасно функціонує множина відносно автономних компонентів (агентів), що приймають локальні рішення та взаємодіють через спільні ресурси й канали обміну даними. Агентами можуть виступати програмні сервіси, контейнери, віртуальні машини, окремі потоки обробки запитів, підсистеми моніторингу, планувальні домени, а також групи користувачів (тенанти), яким надається спільна інфраструктура. Важливою ознакою такого середовища є конкуренція: навіть за наявності розвинених механізмів ізоляції агенти впливають один на одного через черги, обмежену місткість вузлів і спільні політики доступу. Унаслідок цього поведінка системи визначається не тільки продуктивністю окремих компонентів, а й тим, як організовано розподіл ресурсів між паралельними заявками [21, 22].

У прикладних інфраструктурах багатоагентність найчастіше проявляється через багатокористувацькі кластери та мікросервісні архітектури. У контейнерних кластерах один фізичний вузол одночасно обслуговує десятки або сотні подів, кожен із яких представляє певний сервіс чи частину конвеєра обробки даних. У таких системах керуючий контур (оркестратор) приймає рішення щодо розміщення задач, повторного планування та відновлення після відмов, тоді як агенти «знизу» створюють змінне навантаження, що не завжди підпорядковується одному центру [77]. У крайових (edge) середовищах додатково з'являється географічна розподіленість і дефіцит ресурсів на периферії; агенти можуть бути прив'язані до конкретних вузлів або зон, а ресурси мають значно жорсткіші обмеження [16, 47]. Для хмарних платформ типовою є мультиорендність, коли різні агенти конкурують за кластери загального призначення, а правила справедливості стають частиною політики експлуатації та економіки сервісу.

Ресурсні обмеження в багатоагентних системах зручно поділяти на кілька груп. По-перше, це обчислювальні ресурси: процесорний час (CPU), обсяг оперативної пам’яті (RAM), прискорювачі (GPU/TPU), а також локальні або мережеві накопичувачі. По-друге, це комунікаційні ресурси: пропускна здатність мережі, затримки, квоти на кількість з’єднань, ліміти на обсяг переданих даних. По-третє, це ресурсні «слоти» керуючого контуру: ліміти на одночасну кількість задач, що можуть бути запущені, на частоту перепланування, на паралельність виконання контрольних процедур, на час технічних вікон тощо. На практиці саме поєднання цих груп формує ситуації, коли система начебто має вільний ресурс за одним виміром, але не може прийняти нову задачу через дефіцит іншого ресурсу або через політичні/топологічні обмеження розміщення [49, 50].

Ключовою складністю є багаторесурсний характер задач. Для кожного агента характерний власний «профіль» споживання: одні навантаження є CPU-інтенсивними, інші - пам’яттєво-інтенсивними, треті - чутливими до мережевих затримок чи I/O. У таких умовах прості стратегії, що розглядають лише один ресурс (наприклад, CPU), призводять до перекосів: окремі агенти займають «необхідний» ресурс і блокують решту, або, навпаки, залишають фрагментовані залишки, непридатні для запуску нових задач. Для подолання цієї проблеми застосовуються моделі багаторесурсної справедливості, де порівняння агентів виконується не за абсолютними значеннями, а за частками споживання в нормованому просторі ресурсів; саме це лежить в основі підходу DRF і його сучасних модифікацій [5, 6, 28].

Окремо доцільно виокремити відмінність між подільними та дискретними ресурсами. Подільні ресурси (час CPU, частка пропускної здатності, частка GPU-часу) допускають гнучке «нарізання» та квотування, що дозволяє відносно плавно вирівнювати частки агентів. Дискретні ресурси (екземпляри задач, слоти, конкретні пристрої, вузли з унікальними можливостями) розподіляються пакетами: або ресурс виділяється повністю, або не виділяється взагалі. Через це навіть ідеально сформульована справедливість на рівні часток може «лама́тися» при реальному запуску задач, коли доводиться вирішувати задачу пакування та враховувати

цілісність вимог (наприклад, мінімальний обсяг RAM або необхідність конкретного типу GPU). У теоретичних підходах до дискретного розподілу акцент робиться на властивостях envy-free та наближених гарантіях, що дозволяє описувати компроміси між справедливістю та ефективністю в умовах дискретності [38, 45].

Важливою властивістю сучасних навантажень є нестаціонарність. Інтенсивність запитів змінюється хвилеподібно, компоненти масштабуються, агенти приєднуються та залишають систему, а профілі споживання змінюються разом із фазами виконання (ініціалізація, прогрів, пік, завершення). Унаслідок цього розподіл ресурсів має підтримувати справедливість «у часі», а не лише в одному моменті: критичним стає запобігання накопиченню боргу для окремих агентів і контроль ситуацій, коли короткі піки одних задач витісняють тривалі фонова навантаження інших. У моделях онлайн-розподілу підкреслюється, що безпосередня реакція на поточні запити без механізму вирівнювання з часом призводить до систематичних перекосів, навіть якщо локальні рішення здаються «раціональними» [9, 10, 11].

Ресурсні обмеження проявляються також через правила та політики доступу. У кластерах можуть діяти квоти на простори імен, пріоритетні класи, обмеження за групами, правила антиафініті, вимоги до розміщення біля даних (locality), а також обмеження на «шумні» задачі, що створюють надмірний тиск на кеші, мережу або дискову підсистему. У сервісах інференсу великих моделей особливо помітними стають GPU-дефіцит і залежність від пакетування запитів, через що питання справедливого доступу до черги та контроль затримок перетворюються на головний фактор якості обслуговування [24, 25, 78]. Це демонструє, що справедливість не може бути зведена лише до формули; вона має бути узгоджена з політикою експлуатації, технічними обмеженнями та метриками сервісу.

У підсумку багатоагентна комп’ютерна система може бути подана як сукупність агентів, набору ресурсів та множини обмежень, що визначають допустимі рішення планування. Для кожної задачі або запиту формується вектор вимог за ресурсами, а планувальник має виконувати послідовне прийняття рішень

щодо виділення ресурсів і запуску задач у часі. Саме на цьому рівні виникає потреба у методі послідовного розподілу ресурсів на основі справедливого планувальника, який поєднує багаторесурсність, дискретні обмеження та динаміку навантажень, зберігаючи керованість і передбачуваність поведінки системи [21, 22, 5, 9].

1.2 Підходи до справедливості в задачах розподілу ресурсів

Справедливість у задачах розподілу ресурсів у багатоагентних комп'ютерних системах розглядається як сукупність правил, що обмежують перекоси доступу до спільних ресурсів і забезпечують передбачувану поведінку системи за конкуренції агентів. У прикладних умовах поняття справедливості має практичний зміст: воно визначає, чи зберігаються стабільні частки ресурсів для різних користувачів і сервісів, чи не виникає голодування, а також наскільки узгоджуються пріоритети із фактичним розподілом часу й місткості. У сучасних роботах підкреслюється, що справедливість майже завжди вступає в компроміс з ефективністю використання ресурсів, тому коректно сформульовані критерії стають основою для порівняння алгоритмів і вибору політики планування [10, 11].

Базові підходи до справедливості історично формувалися навколо ідеї рівного доступу до одного ресурсу, найчастіше - процесорного часу або пропускної здатності. У цьому класі задач застосовуються принципи рівного поділу (equal share), зваженого поділу (weighted share) та «справедливості за чергою», коли порядок обслуговування наближає частки агентів до заданих квот. Такі механізми є природними для подільних ресурсів, де виділення частки може виконуватися поступово і коригуватися в часі. Для прикладних систем важливо, що ці підходи легко поєднуються з поняттям боргу: якщо агент недоотримав ресурс у попередніх інтервалах, надалі його пріоритет може тимчасово зростати, що зменшує ризик тривалої деградації [10].

Іншим базовим напрямом є max-min fairness, де розподіл вважається справедливим, якщо неможливо збільшити частку одного агента без зменшення

частки іншого агента, який уже має не більшу частку. Така логіка є корисною в ситуаціях дефіциту, коли потрібно гарантувати мінімально прийнятний рівень ресурсів для всіх учасників. Для системного планування це відповідає інтуїції «спочатку вирівнюється мінімум», а потім - підвищуються частки тих, хто отримує менше. Разом із тим max-min fairness у чистому вигляді не завжди узгоджується з метою високої утилізації ресурсів і може бути надто консервативною при неоднорідних профілях навантаження [69].

У багаторесурсних середовищах (CPU+RAM+мережа+I/O+GPU) справедливість не може визначатися одним числом, тому застосовуються узагальнення, що працюють у багатовимірному просторі. Найбільш відомою моделлю є Dominant Resource Fairness (DRF), де для кожного агента обчислюється домінантна частка - найбільша частка споживання серед усіх ресурсів, після чого ці домінантні частки вирівнюються між агентами. Такий підхід дозволяє зіставляти агентів із різними профілями споживання та запобігати ситуації, коли один ресурс використовується як інструмент систематичного витіснення інших, залишаючи інші ресурси недозавантаженими [5, 6, 28]. Для практики також важливо, що DRF підтримує інтуїтивне трактування «чесності» у вигляді частки від кластера, однак вимагає коректної нормалізації ресурсів і стабільних оцінок потреб задач.

Подальший розвиток DRF пов’язаний із введенням ваг, пріоритетів та додаткових обмежень доступу. У реальних кластерах рідко існує вимога абсолютної рівності для всіх: частина сервісів має вищу критичність, частина користувачів має договірні квоти, а частина задач є фоновими. Через це використовуються вагові модифікації, де домінантні частки вирівнюються з урахуванням коефіцієнтів, а також оптимізаційні підходи, що підбирають ваги або коригувальні параметри, аби збалансувати справедливість та ефективність [5, 69]. Окремий клас рішень враховує еластичний попит і зміну набору активних агентів, що є типовим для хмарних і serverless-навантажень [7, 8]. У таких сценаріях справедливість має підтримуватися не лише в моменті, а як властивість траєкторії розподілу в часі, щоб короткочасні сплески не формували стійкої переваги для окремих агентів [10, 11].

Для дискретних ресурсів (слоти, екземпляри, унікальні пристрої) класичні моделі часток працюють гірше, оскільки ресурс виділяється пакетно, а результат залежить від того, як саме упаковуються задачі у доступні об’єкти. У цьому класі задач важливими стають критерії envy-free (відсутність заздрощів), коли жоден агент не віддає перевагу набору ресурсів іншого агента, а також наближені варіанти envy-free, що допускають обмежені відхилення. Сучасні роботи демонструють, що обчислення точних envy-free розподілів може бути складним, тому на практиці застосовуються часткові або наближені гарантії, які забезпечують прийнятний баланс між справедливістю й обчислювальною складністю [38, 45, 75]. Для багатоагентних комп’ютерних систем це означає, що критерії справедливості мають узгоджуватися з механізмами планування, які працюють із дискретними вимогами (мінімальний RAM, конкретний тип GPU, афініті до вузла тощо).

Окрему групу підходів формують критерії, що поєднують справедливість і ефективність через цільові функції. Прикладом є ідеї оптимізації за Нешем (Nash social welfare), коли максимізується добуток або сума логарифмів корисностей агентів, що природно «штрафує» надмірні перекоси. У прикладних постановках корисність може відповідати пропускній здатності, часу відгуку, кількості виконаних задач або іншій метриці якості сервісу. Подібні підходи корисні тим, що дозволяють формалізувати компроміс між загальним виграшем і рівномірністю розподілу, а також зручно поєднуються з обмеженнями на мінімальні гарантії [69].

У задачах послідовного (онлайн) розподілу ресурсу справедливість зазвичай трактується як властивість процесу: у кожний момент часу рішення приймається за поточними заявками, але результат оцінюється на часовому інтервалі. У роботах з онлайн-розподілу підкреслюється, що наївні жадібні стратегії можуть бути локально раціональними, але приводити до систематичного голодування, якщо відсутній механізм накопичення й компенсації потреби [9, 10, 11]. Через це для послідовного розподілу важливими стають поняття довгострокової частки, «затримки справедливості» та межі допустимого відхилення від квот на ковзному вікні часу. Такі показники дозволяють оцінити не лише середню справедливість, а й стабільність у динаміці.

Сучасний розвиток багатоагентних підходів до оптимізації сприяв появі методів, у яких справедливість вводиться як обмеження або як частина функції винагороди, зокрема в multi-agent reinforcement learning. У цьому випадку політика розподілу ресурсів формується на даних або в симуляції, а критерії справедливості можуть задаватися як штрафи за нерівність, як вимоги до групової рівності чи як обмеження на мінімальні рівні обслуговування [1, 20, 22, 40]. Такі підходи корисні для формування метрик і для порівняння з алгоритмічними планувальниками, однак у прикладних системах постають питання відтворюваності, інтерпретованості та гарантованої поведінки у рідкісних режимах навантаження [22, 40].

У підсумку справедливість у задачах розподілу ресурсів у багатоагентних системах доцільно розглядати як набір взаємодоповнювальних критеріїв, що залежать від типу ресурсу (подільний або дискретний), режиму прийняття рішень (офлайн або онлайн) та політики експлуатації (квоти, пріоритети, доступність). Для подальших підрозділів роботи ключовими є підходи, що підтримують багаторесурсність, допускають послідовне прийняття рішень та забезпечують контроль відхилення часток у часі, оскільки саме ці властивості визначають практичну придатність справедливого планувальника як основи методу послідовного розподілу ресурсів [5, 10, 11, 69].

1.3 Справедливі планувальники як основа алгоритмів розподілу ресурсів

Справедливий планувальник у багатоагентних комп’ютерних системах розглядається як механізм, що перетворює абстрактні правила справедливості на конкретні рішення: кому, коли і в якому обсязі надавати доступ до ресурсу. На практиці планувальник виконує роль «перекладача» між вимогами агентів і обмеженнями середовища. Саме через нього реалізуються квоти, пріоритети, політики доступу та захист від голодування. Якщо механізм планування є непрозорим або не має вбудованих інструментів вирівнювання, навіть коректно визначена модель справедливості не забезпечує стабільної поведінки системи,

оскільки рішення ухвалюються в дискретні моменти часу й залежать від поточного стану черг [10, 11].

У загальному вигляді планувальник працює з чергами заявок і формує послідовність рішень про обслуговування. Для подільних ресурсів базовими інструментами є поділ часу на кванти, вагові коефіцієнти та правила вибору наступного агента з урахуванням накопиченої потреби. Для дискретних ресурсів до цього додається перевірка здійсненності запуску: задача має «вміститися» у доступні ресурси або відповідати топологічним/політичним вимогам. У результаті планування стає двоетапним: спочатку визначається кандидат за критеріями справедливості, потім перевіряється можливість виконання за обмеженнями середовища; якщо умови не виконуються, відбір повторюється. Такий цикл демонструє, що планувальник є не лише оптимізатором, а й інструментом узгодження правил із реальністю, де діють конфліктні вимоги [49, 50].

Підходи до справедливого планування традиційно формувалися на рівні операційних систем. Саме тут з’явилася ідея підтримувати приблизно рівний доступ конкурентних потоків до CPU, не допускаючи ситуацій, коли один процес систематично витісняє інші. У таких механізмах справедливість трактується як контроль частки процесорного часу з урахуванням ваг, а також як компенсація «боргу», коли недоотриманий ресурс у минулому підвищує шанс отримання ресурсу в майбутньому. Ця логіка є корисною як концептуальна основа, оскільки для послідовного розподілу ресурсів у багатоагентному середовищі важливо мати механізм, що не лише реагує на поточний стан черги, а й вирівнює доступ у часі [10]. Унаслідок цього підходи ОС-рівня часто використовуються як прототипи для побудови справедливих механізмів у розподілених системах.

На рівні кластерів і хмарних інфраструктур планувальник відповідає за розміщення задач на вузлах та за узгодження багаторесурсних вимог із фактично доступними ресурсами. У контейнерних середовищах типу Kubernetes планувальник приймає рішення про розміщення подів, керується вимогами до ресурсів, політиками афініті/антиафініті, обмеженнями доступу та іншими правилами, що задаються конфігурацією кластера [77]. Важливою особливістю є

розширюваність: через профілі та плагіни можна змінювати логіку відбору й ранжування вузлів, а також реалізовувати додаткові політики для пріоритизації й вирівнювання доступу. Це створює практичну основу для впровадження справедливих алгоритмів розподілу ресурсів у виробничих середовищах без руйнування базової архітектури системи [77].

Справедливість на кластерному рівні рідко обмежується одним ресурсом, тому планувальник має працювати з багатовимірними обмеженнями. У таких умовах виникає потреба у механізмах багаторесурсного вирівнювання, що враховують домінантні частки, ваги та обмеження доступу. Практичний сенс цих механізмів полягає у тому, що вони зменшують перекоси між агентами з різними профілями споживання: наприклад, агент, який активно споживає CPU, не має отримувати необмежений доступ лише через те, що пам’яті в системі багато. Розвиток багаторесурсних підходів до справедливості показує, що ефективний планувальник має зберігати баланс між утилізацією ресурсів та стабільністю часток, а також мати механізми реагування на еластичний попит [7, 28, 69].

Окремий пласт сучасних задач пов’язаний із високовартісними ресурсами, насамперед GPU, а також із навантаженнями, чутливими до затримок і пакетування. Для сервісів інференсу великих моделей підкреслюється, що справедливість обслуговування черг є не другорядною характеристикою, а умовою підтримання прийнятної якості сервісу в умовах дефіциту GPU-ресурсів. Тут планувальник змушений поєднувати fairness із формуванням батчів, локальністю даних і обмеженнями на пам’ять, а також контролювати деградацію затримок для різних класів запитів [24, 25, 78, 79]. Це демонструє, що справедливий планувальник має враховувати не лише частки ресурсів, а й технологічні аспекти виконання задач.

Для послідовного розподілу важливою є властивість стабільності у часі. Якщо планувальник орієнтується лише на «зріз» поточного стану, навіть при наявності ваг і пріоритетів можливі ситуації, коли короткочасні сплески одного агента витісняють інші агенти на тривалий час. Тому в сучасних підходах підкреслюється необхідність механізмів, що зберігають історію або агреговані

показники (борг/кредит, віртуальний час, накопичене відхилення від квоти) та використовують їх у рішенні про вибір наступного кандидата. Це дозволяє забезпечувати «справедливість на інтервалі» і контролювати величину відхилення часток у ковзному часовому вікні [9, 10, 11].

У прикладних системах справедливість часто реалізується через поєднання кількох інструментів: квоти як нижні гарантії, пріоритети як механізм швидкого реагування на критичні події, та справедливий відбір як стабілізатор, що не допускає систематичного перекосу. Така комбінація є особливо важливою для багатоагентних середовищ, де одночасно присутні різні класи задач і різні горизонти часу: аварійні/інтерактивні запити потребують мінімальної затримки, тоді як фонові та аналітичні задачі потребують гарантії доступу в довшому інтервалі. Унаслідок цього справедливий планувальник у багатоагентній системі доцільно розглядати як ядро алгоритму послідовного розподілу ресурсів, яке забезпечує узгодження локальних рішень із глобальними правилами справедливості [10, 11, 69].

У підсумку справедливі планувальники формують практичну основу для алгоритмів розподілу ресурсів, оскільки вони задають процедури відбору, ранжування та компенсації відхилень, що виникають при динамічних навантаженнях. Їх використання дозволяє поєднувати багаторесурсність, дискретні обмеження та політики доступу в єдиному механізмі, який підтримує керованість і передбачуваність. Саме тому побудова методу послідовного розподілу ресурсів на основі справедливого планувальника розглядається як логічний шлях до зменшення перекосів, обмеження голодування та стабілізації якості обслуговування в багатоагентних комп’ютерних системах [9, 10, 11].

1.4 Послідовний (онлайн) розподіл ресурсів у динамічному багатоагентному середовищі

Послідовний (онлайн) розподіл ресурсів розглядається як режим керування, за якого рішення про виділення ресурсу ухвалюються під час надходження заявок,

без повної інформації про майбутні потреби агентів. Для багатоагентних комп'ютерних систем це є типовим випадком: інтенсивність запитів змінюється хвилеподібно, частина агентів приєднується або припиняє активність, а профіль споживання ресурсів залежить від фази виконання задачі. У таких умовах «одноразово оптимальний» розподіл не існує, натомість важливо підтримувати керовану траєкторію розподілу в часі, щоб випадкові сплески не перетворювалися на стійку перевагу окремих агентів [9, 10].

Онлайн-постановка відрізняється від офлайн-розподілу тим, що планувальник працює з неповним станом: на вході є поточні черги, доступні ресурси та політики доступу, а інформація про майбутні надходження або невідома, або має імовірнісний характер. Це підсилює роль правил прийняття рішень, які мають бути простими для виконання в реальному часі, але достатньо «розумними», щоб накопичені ефекти не призводили до деградації сервісу. У роботах з онлайн-розподілу підкреслюється, що жадібні стратегії, які оптимізують лише поточну утилізацію, здатні породжувати несправедливість на інтервалі та збільшувати час очікування для агентів із менш «зручним» профілем задач [9, 11].

Ключовою проблемою послідовного розподілу є конкуренція між короткостроковою ефективністю та довгостроковою справедливістю. Висока утилізація ресурсу зазвичай досягається через пріоритизацію задач, які швидко заповнюють доступні «вікна» ресурсу, однак це може витісняти задачі з більшими вимогами або з нижчим пріоритетом у політиці доступу. Якщо відсутній механізм компенсації, формується ефект голодування: агент систематично недоотримує ресурс, а час очікування необмежено зростає. Конфлікт між «локально правильним» вибором і глобальною справедливістю описується як одна з центральних причин, чому онлайн-алгоритми потребують спеціальних інструментів вирівнювання [10, 11].

Для контролю справедливості в часі використовуються поняття боргу/кредиту, віртуального часу, ковзного вікна та обмеження відхилення часток. Практичний зміст таких механізмів полягає в тому, що агент, який недоотримав ресурс у попередніх інтервалах, накопичує «кредит» і отримує підвищену

ймовірність або пріоритет обслуговування пізніше. Це дозволяє не лише вирівнювати середні частки, а й обмежувати тривалість відхилення від квот. У сучасних підходах до онлайн-розподілу окремо розглядаються гарантії справедливості на інтервалі та оцінка «ціни» справедливості як втрати ефективності, що виникає через необхідність утримувати обмеження на нерівність [9, 10].

Багаторесурсність ускладнює онлайн-розподіл, оскільки рішення про виділення ресурсу не може спиратися на один вимір. Навіть якщо за CPU існує запас, дефіцит пам’яті або мережевої пропускної здатності здатний заблокувати запуск задачі. Унаслідок цього з’являється фрагментація: доступний ресурс розподіляється на залишки, які не підходять для типових заявок, що знижує утилізацію і підвищує затримки. Роботи, присвячені динамічному багаторесурсному розподілу та еластичному попиту, демонструють, що підтримання справедливості без урахування структури попиту призводить до нестабільних режимів, де система «перекошується» між ресурсами [7, 8]. Це робить актуальними підходи, які нормалізують вимоги та порівнюють агентів у просторі часток, зокрема домінантних часток, але вже в онлайн-режимі [5, 11].

Важливою особливістю динамічного середовища є зміна складу агентів. У багатокористувацьких кластерах агенти можуть з’являтися і зникати, змінювати інтенсивність запитів, масштабуватися горизонтально, а також змінювати пріоритети залежно від контексту. Для справедливого планувальника це означає, що «справедливість для всіх» не може бути зафіксована як константа: потрібні правила перерахунку часток та механізми плавної адаптації, які не створюють різких стрибків у доступі до ресурсу. У сучасних підходах до багаторесурсного розподілу ця проблема пов’язується з еластичністю попиту та необхідністю швидкого перерахунку квот без руйнування керованості [7, 28].

Практичні середовища також накладають додаткові політики: пріоритетні класи, квоти на групи, обмеження доступу та правила розміщення. У результаті онлайн-алгоритм має забезпечувати не лише справедливість у математичному сенсі, а й виконання експлуатаційних правил. Один із підходів полягає в поєднанні

«жорстких» гарантій (квоти/ліміти) з «м’яким» справедливим відбором усередині дозволеного простору. Це дозволяє зберігати керованість у кризових режимах, коли ресурс дефіцитний, і водночас не втрачати справедливість у нормальному режимі, коли ресурсів достатньо [33, 46].

Окремий клас онлайн-сценаріїв пов’язаний із високовартісними ресурсами та пакетуванням запитів, зокрема в обслуговуванні LLM. Тут рішення про виділення GPU-ресурсу тісно пов’язане з формуванням батчів, локальністю та обмеженнями пам’яті, а справедливість виражається не лише в частці ресурсу, а й у затримці відповіді для різних черг. Роботи про fair scheduling у LLM-serving підкреслюють, що без механізмів контролю черг окремі користувацькі потоки можуть систематично накопичувати затримку навіть за високої загальної пропускної здатності [24, 25, 78, 79]. Це підсилює вимогу до послідовного методу: справедливість має відстежуватися в часі та бути пов’язана з метриками якості сервісу.

У підсумку послідовний (онлайн) розподіл ресурсів у багатоагентному середовищі потребує алгоритму, який: (1) працює з неповною інформацією та динамічними чергами; (2) підтримує багаторесурсні обмеження й зменшує фрагментацію; (3) має механізм компенсації відхилень (борг/кредит, віртуальний час) і контролює голодування; (4) узгоджується з політиками доступу (квоти, пріоритети, групи); (5) забезпечує вимірювані гарантії справедливості на інтервалі при прийнятній втраті ефективності [9-11, 7, 33]. Саме ці вимоги формують основу для побудови методу послідовного розподілу ресурсів на базі справедливого планувальника в межах подальших підрозділів роботи.

1.5 Постановка задачі

У межах роботи розглядається задача організації послідовного розподілу обчислювальних ресурсів у багатоагентному середовищі, де паралельно функціонує множина агентів і формується конкуренція за спільні ресурси. Така постановка є типовою для кластерних і хмарних платформ, а також для крайових

інфраструктур, у яких рішення про виділення ресурсу приймається в онлайн-режимі на основі поточного стану черг, доступних ресурсів і заданих політик доступу. Внаслідок динамічного надходження заявок, змінного складу активних агентів і неоднорідних профілів споживання виникає потреба у механізмі, який підтримує справедливість не лише в окремий момент, а як властивість розподілу в часі, обмежуючи перекоси та запобігаючи голодуванню.

Об’єктом роботи є процеси розподілу обчислювальних ресурсів у багатоагентних комп’ютерних системах з динамічними навантаженнями. Предметом роботи є методи та алгоритми послідовного розподілу ресурсів на основі справедливого планувальника, які забезпечують контроль відхилення часток агентів у часі та стабільність доступу до ресурсів за наявності багаторесурсних обмежень, дискретності ресурсів і експлуатаційних правил. Метою роботи є розроблення та обґрунтування методу послідовного розподілу ресурсів у багатоагентних комп’ютерних системах на основі справедливого планувальника, який поєднує вимоги справедливості та ефективності й може бути застосований у прикладному контурі планування.

Постановка задачі передбачає наявність множини агентів, що надсилають заявки на виконання, та набору ресурсів, які мають обмежену місткість і різну природу. Для кожної заявки задаються потреби за кількома ресурсами одночасно, а також умови доступу, що можуть включати ваги, квоти, класи пріоритетів і додаткові обмеження здійсненності запуску, пов’язані з дискретністю ресурсів або політиками розміщення. Стан системи в кожний момент часу визначається доступними залишками ресурсів, чергами заявок і накопиченими показниками, що відображають історію отримання ресурсу агентами. На основі цього стану має виконуватися послідовний вибір заявки та рішення щодо її допуску до виконання так, щоб зберігалася керованість і передбачуваність розподілу.

Ключовою вимогою є забезпечення справедливості у часі, коли фактичні частки ресурсів агентів не відхиляються від цільових більш ніж на прийнятну величину протягом заданого інтервалу. Додатково має обмежуватися голодування, тобто ситуації, коли агент тривалий час не отримує ресурс через конкурентні

сплески інших агентів або через невдалу структуру поточних рішень. Паралельно має зберігатися ефективність використання ресурсів, оскільки надмірно жорстке вирівнювання може погіршувати утилізацію і підвищувати затримки. Окреме значення має узгодження алгоритму зі встановленими політиками доступу, щоб пріоритетність критичних задач забезпечувалась без перетворення системи на нестабільний режим постійного витіснення менш пріоритетних, але необхідних процесів.

Оцінювання запропонованого методу має виконуватися за сукупністю показників, що відображають справедливість, затримки та ефективність. Справедливість доцільно характеризувати рівномірністю розподілу та величиною відхилення часток у часі, затримки - середніми значеннями та перцентилями часу очікування, а ефективність - утилізацією ресурсів і рівнем фрагментації, яка проявляється як неможливість запуску задач за наявності «розбитих» залишків ресурсів. У підсумку задача зводиться до побудови послідовного механізму, який використовує логіку справедливого планувальника для стабілізації часток у часі, одночасно враховує багаторесурсні обмеження та забезпечує працездатність у реальному, динамічному навантаженні.

У постановці задачі окремої уваги потребує вибір рівня абстракції, на якому працює метод. У прикладних системах рішення планувальника можуть стосуватися як безпосереднього допуску заявки до виконання, так і визначення частки ресурсу, що виділяється агенту в певному часовому інтервалі. За наявності багаторесурсних обмежень важливим стає узгодження цих рішень між ресурсами, щоб поліпшення доступу за одним виміром не створювало прихованого блокування за іншим. Унаслідок цього планувальник має оперувати не лише поточним станом черг, а й інформацією про структуру попиту, типові “вузькі місця” та наслідки дискретності ресурсів для здійсненності запуску. Це дозволяє зробити послідовний розподіл більш стабільним, а поведінку системи - передбачуваною навіть у режимах короткочасних піків і нерівномірного надходження заявок.

2 РОЗРОБЛЕННЯ МЕТОДУ ПОСЛІДОВНОГО РОЗПОДІЛУ РЕСУРСІВ

2.1 Формалізація багатоагентного середовища та ресурсних обмежень

Для побудови методу послідовного розподілу ресурсів необхідно формалізувати середовище, у межах якого приймаються рішення планування. У багатоагентній комп’ютерній системі одночасно функціонує множина учасників, що формують запити на спільні ресурси та впливають один на одного через обмежену місткість інфраструктури. У ролі таких учасників можуть виступати сервіси, контейнери, окремі задачі, віртуальні машини або групи користувачів. За умов паралельного надходження заявок саме обмеженість ресурсів створює ситуацію конкуренції, у якій потрібен механізм впорядкованого та керованого вибору.

Багатоагентне середовище доцільно подати у вигляді сукупності агентів, ресурсів, заявок і обмежень запуску. Нехай задано множину агентів, де кожен агент генерує послідовність заявок на обслуговування. У межах цієї моделі агент розглядається як окремий споживач ресурсів, для якого можна відстежувати історію отримання ресурсу, рівень поточного навантаження та відхилення від бажаного режиму обслуговування. Такий підхід дозволяє працювати не лише з окремими заявками, а й з накопиченим станом кожного агента, що є важливим для забезпечення справедливості у часі.

Ресурсну основу системи пропонується задати множиною, де кожному ресурсу відповідає повна доступна місткість. До складу ресурсів можуть входити процесорний час, оперативна пам’ять, графічні прискорювачі, пропускна здатність мережі, дискові операції або інші обмежені компоненти інфраструктури. Кожна заявка описується вектором потреб де визначає потребу заявки в ресурсі типу. Таке подання дає змогу перейти від описового рівня до кількісної моделі, придатної для подальшого алгоритмічного опрацювання.

$$A = \{a_1, a_2, \ldots, a_n\}, \quad R = \{r_1, r_2, \ldots, r_m\}, \tag{2.1}$$

де A – множина агентів;

a_n – n-й агент;

R – множина ресурсів;

r_m – m-й ресурс.

$$d_k = (d_{k1}, d_{k2}, \ldots, d_{km}), \tag{2.2}$$

де d_k – вектор ресурсних потреб k-ї заявки;

d_{kj} – обсяг потреби k-ї заявки в j-му ресурсі;

m – кількість типів ресурсів.

Для практичного середовища недостатньо враховувати лише сумарний обсяг доступних ресурсів. Часто запуск заявки залежить також від додаткових умов: наявності конкретного типу GPU, мінімального вільного обсягу пам’яті, належності вузла до певної зони, правил афініті або антиафініті, а також локальних політик доступу. Через це множина теоретично можливих розподілів не збігається з множиною реально допустимих. Частина рішень, що здаються коректними з погляду вільного ресурсу, не може бути реалізована через структурні або політичні обмеження. Саме тому формалізація середовища повинна включати не тільки ресурсні залишки, а й набір допустимих станів, у межах яких запуск задачі вважається здійсненним.

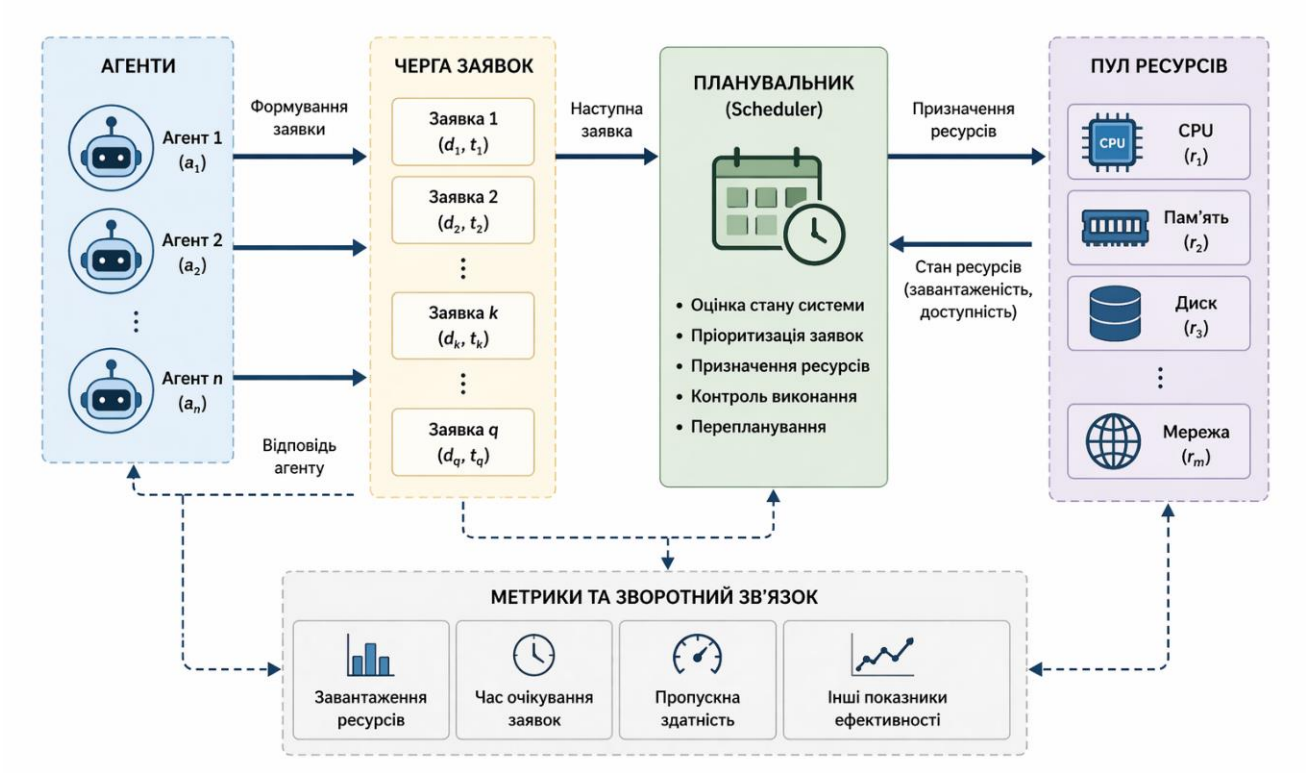
Для опису поточного стану системи у момент часу доцільно ввести стан S(t)S(t)S(t), який містить залишкові обсяги ресурсів, чергу необслугованих заявок, множину активних задач і службові характеристики агентів. Особливу роль у цій моделі відіграє не лише факт наявності заявки, а й історія попереднього обслуговування. У системі послідовного розподілу важливо враховувати, чи отримував агент ресурс рівномірно, чи, навпаки, тривалий час перебував у менш

вигідному становищі. Внаслідок цього опис стану має включати не тільки поточні параметри, а й показники, які відображають накопичений ефект попередніх рішень.

Для кожного агента та ресурсу доцільно визначити частку використання як відношення вже виділеного ресурсу до повної місткості. Таке нормування переводить усі ресурси до спільної відносної шкали та дозволяє порівнювати різнорідні профілі навантаження. На цій основі в подальших підрозділах буде визначено домінантну частку агента, тобто найбільшу з його нормованих ресурсних часток. Саме вона стане базою для оцінювання багаторесурсної справедливості та зіставлення агентів, які споживають різні типи ресурсів.

Оскільки система працює в умовах динамічного надходження заявок, важливим елементом формалізації є часовий аспект. Кожна заявка надходить у певний момент, очікує на розгляд, запускається або тимчасово відкладається через нестачу ресурсу чи невиконання умов розміщення. Тому модель має описувати не лише статичний баланс ресурсів, а й послідовність рішень у часі. На кожному кроці планування система повинна визначити, яка заявка є найбільш доцільною для розгляду, чи є її запуск здійсненним і як це рішення вплине на подальший стан середовища.

У межах запропонованого підходу доцільно також враховувати узагальнений показник відхилення агента від бажаного режиму обслуговування. Цей показник відображає, наскільки фактичний доступ агента до ресурсів відрізняється від очікуваного. Якщо агент протягом певного часу отримував менше ресурсу, ніж мав би отримати відповідно до прийнятої політики, його стан має це фіксувати. Якщо ж агент отримував ресурс інтенсивніше за інших, відповідне відхилення також повинно накопичуватися. Такий службовий параметр створює основу для подальшого механізму компенсації перекосів, що буде реалізовано в наступних підрозділах.



Рисунка 2.1 – Життєвий цикл заявки: надходження, перевірка умов, очікування, запуск, завершення.

Отже, багатоагентне середовище у цій роботі подається як сукупність агентів, типів ресурсів, заявок та обмежень запуску, що змінюються у часі. Така формалізація дозволяє пов’язати між собою ресурсний стан системи, історію обслуговування агентів і правила прийняття рішень. У підсумку вона створює основу для подальшого введення критерію справедливості, моделі домінантної частки та механізму компенсації відхилень, на яких буде побудовано метод послідовного розподілу ресурсів.

2.2 Критерії справедливості та модель домінантної частки ресурсів

Після формалізації багатоагентного середовища необхідно визначити, за яким критерієм оцінюватиметься правильність розподілу ресурсів між агентами. У системах, де одночасно існує кілька типів ресурсів, різні профілі навантаження та змінна інтенсивність надходження заявок, поняття справедливості не може

зводитися лише до рівного поділу. Формально однаковий доступ не завжди означає коректний розподіл, оскільки окремі агенти можуть споживати різні комбінації ресурсів і створювати нерівномірне навантаження на інфраструктуру. Саме тому справедливість у цій роботі доцільно трактувати як збалансованість доступу до спільних ресурсів, за якої жоден агент не отримує стійкої переваги лише через особливості свого профілю споживання.

У найпростішому випадку, коли розглядається лише один ресурс, справедливість часто описується через рівність часток або через відповідність наперед заданим вагам. Такий підхід може бути придатним для однорідного середовища, однак у багаторесурсній системі він не дає повної картини. Якщо оцінювання виконувати лише за процесорним часом, агент із високим споживанням пам’яті або GPU може формально виглядати «легким», хоча фактично буде займати критичну частину інфраструктури. Аналогічна проблема виникає й у зворотному випадку, коли контроль ведеться лише за пам’яттю або лише за мережею. Через це виникає потреба у такому узагальненому показнику, який дозволяє порівнювати між собою агентів із різними профілями навантаження.

Для цього доцільно використати модель домінантної частки ресурсу. Її суть полягає в тому, що для кожного агента розглядаються нормовані частки споживання всіх ресурсів, після чого серед них виділяється найбільша. Саме вона і вважається домінантною часткою. Такий підхід дає можливість оцінювати агента не за середнім або випадково вибраним показником, а за тим ресурсом, який у поточний момент є для нього найбільш обмежувальним. Якщо один агент найбільше навантажує CPU, а інший - GPU або оперативну пам’ять, обидва можуть бути коректно порівняні через свої домінантні частки, навіть попри відмінність характеру навантаження.

Домінантна частка є зручною ще й тому, що вона переводить багаторесурсну задачу до компактнішого вигляду. Для кожного агента замість набору розрізнених оцінок формується один узагальнений показник, який відображає його фактичне положення у системі. Це не означає втрати інформації про інші ресурси. Навпаки, у домінантній частці зберігається найкритичніший для агента вимір, тобто той,

який найбільше впливає на справедливість загального розподілу. Така модель добре узгоджується з реальними умовами багаторесурсного середовища, де дефіцит зазвичай формується не рівномірно, а концентрується навколо окремих типів ресурсів.

Для оцінювання справедливості розподілу важливо не лише визначити домінантну частку, а й установити правило її інтерпретації. У межах цієї роботи справедливішим вважається такий стан системи, за якого домінантні частки агентів не мають надмірного перекосу. Інакше кажучи, якщо один агент стабільно займає значно більшу частину найбільш критичного для себе ресурсу, ніж інші агенти, це свідчить про дисбаланс. Якщо ж домінантні частки наближені одна до одної або відповідають встановленим вагам, можна говорити про збалансований розподіл.

Разом із цим у реальній системі не завжди потрібна абсолютна рівність між усіма агентами. Частина сервісів може мати вищий пріоритет, жорсткіші вимоги до затримки або більшу вагу відповідно до політики експлуатації. Через це до моделі справедливості доцільно включити вагові коефіцієнти, які задають відносно допустимий рівень доступу для кожного агента. У такому випадку порівняння виконується не за «чистими» домінантними частками, а за їх співвідношенням із призначеними вагами. Це дозволяє поєднати ідею справедливого розподілу з практичними вимогами середовища, де різні класи задач можуть обслуговуватися неоднаково, але все одно в контрольованих межах.

На цій основі доцільно ввести узагальнений показник справедливості, який характеризує ступінь вирівнювання доступу агентів до ресурсів. У межах цього підрозділу важливо не стільки зафіксувати конкретну остаточну формулу, скільки задати її властивості. Такий показник повинен зменшуватися зі зростанням перекосу між агентами, враховувати багаторесурсний характер системи та бути придатним для порівняння різних режимів роботи планувальника. У подальшому саме він буде використовуватися під час аналізу ефективності запропонованого методу та порівняння його з базовими підходами.

Окремо слід підкреслити, що домінантна частка описує лише поточний багаторесурсний стан агента. Вона добре відображає, який агент у конкретний

момент часу найбільше навантажує систему, однак сама по собі не показує, чи був цей агент недообслуговний або переобслуговний раніше. Для послідовного розподілу це принципово важливо, оскільки справедливість має оцінюватися не тільки в одному часовому зрізі, а й на інтервалі роботи системи. Саме тому модель домінантної частки в цій роботі використовується як базова просторова характеристика, а її часове доповнення буде реалізоване далі через механізм боргу і кредиту.

Ще одна перевага такого підходу полягає в тому, що він зменшує ризик прихованих перекосів. Якщо оцінювання виконується лише за одним вибраним ресурсом, окремі агенти можуть стабільно отримувати вигідніше становище, фактично навантажуючи інші, не менш критичні компоненти інфраструктури. Орієнтація на домінантну частку послаблює цю проблему, оскільки агент завжди оцінюється саме за тим ресурсом, за яким він створює найбільший тиск на систему. Це робить механізм планування більш стійким до неоднорідності навантаження та ближчим до практичних умов функціонування розподілених обчислювальних середовищ.

Важливим продовженням розгляду моделі домінантної частки є врахування динамічного характеру змін навантаження в системі. У реальному середовищі значення споживання ресурсів агентами не є сталими, а змінюються залежно від типу задач, їх тривалості та моменту надходження. У зв’язку з цим домінантна частка повинна розглядатися як функція часу, що відображає поточний стан агента в системі. Такий підхід дозволяє більш точно оцінювати вплив кожного агента на загальний баланс ресурсів і своєчасно реагувати на появу перекосів.

У програмному та алгоритмічному контексті це означає, що обчислення домінантної частки повинно виконуватися регулярно, при кожній зміні стану системи. До таких подій належать надходження нових заявок, завершення виконання задач, а також перерозподіл ресурсів між агентами. Це дозволяє підтримувати актуальність оцінки та використовувати її як основу для прийняття рішень у процесі планування. Внаслідок цього система отримує можливість адаптуватися до змін у режимі реального часу.

Окремо варто відзначити, що домінантна частка може використовуватися не лише як критерій оцінювання, але і як елемент механізму прийняття рішень. Зокрема, вона може виступати базою для ранжування агентів при виборі наступної заявки на виконання. У такому випадку перевага надається тим агентам, чия домінантна частка є меншою, що дозволяє поступово вирівнювати доступ до ресурсів. Така стратегія є інтуїтивно зрозумілою та добре узгоджується з ідеєю справедливого розподілу.

Разом із цим необхідно враховувати, що використання лише домінантної частки як єдиного критерію може призводити до надмірної чутливості системи до короткочасних змін. Наприклад, різке зростання споживання одного ресурсу може тимчасово змінити пріоритет агента, навіть якщо в довгостроковій перспективі його поведінка є збалансованою. У зв’язку з цим доцільно розглядати можливість згладжування значень або введення додаткових механізмів, які враховують історію обслуговування. Це забезпечує більш стабільну роботу планувальника.

Також доцільно врахувати, що в умовах практичного застосування система може функціонувати з обмеженнями, пов’язаними з політикою доступу або технічними характеристиками інфраструктури. У таких випадках домінантна частка може бути доповнена додатковими обмеженнями або коригуючими коефіцієнтами, які враховують специфіку середовища. Це дозволяє адаптувати базову модель до реальних умов експлуатації без втрати її основних властивостей.

Отже, у межах цієї роботи критерієм справедливості доцільно вважати збалансованість домінантних часток агентів з урахуванням заданих ваг і політики доступу. Такий підхід дозволяє коректно порівнювати агентів із різними профілями споживання, не зводячи багаторесурсну задачу до одновимірного випадку. У підсумку модель домінантної частки створює основу для подальшого поєднання просторової справедливості з часовим механізмом компенсації відхилень, на якому буде побудовано наступний етап розроблення методу.

2.3 Модель компенсації відхилень на основі механізму боргу і кредиту

Модель домінантної частки дозволяє оцінити поточне положення агента в багаторесурсному середовищі, однак сама по собі вона не забезпечує справедливість на часовому інтервалі. У системі послідовного розподілу рішення приймаються крок за кроком, а склад активних агентів і набір доступних заявок постійно змінюються. Через це навіть за коректного врахування домінантної частки можуть виникати ситуації, коли окремий агент протягом певного часу систематично недоотримує ресурс, тоді як інший отримує перевагу завдяки більш вдалому моменту надходження заявок або кращій сумісності з поточним станом інфраструктури. Для усунення такого ефекту доцільно ввести додатковий механізм, який враховує історію обслуговування та компенсує накопичені перекоси.

У межах цієї роботи такий механізм пропонується реалізувати через модель боргу і кредиту. Її основна ідея полягає в тому, що для кожного агента накопичується службовий показник, який відображає відхилення фактичного доступу до ресурсу від очікуваного. Якщо агент протягом деякого інтервалу часу отримував менше, ніж мав би отримати відповідно до прийнятої політики розподілу, формується кредит, який підвищує його шанси на обслуговування в наступних кроках. Якщо ж агент одержував ресурс активніше за інших і фактично випереджав свою очікувану частку, формується борг, що зменшує його перевагу при подальшому виборі кандидата. У результаті планувальник перестає бути суто реактивним і починає враховувати не лише поточний стан черги, а й попередню історію розподілу.

Для формалізації такого механізму для кожного агента вводиться змінна, яка відображає його накопичене відхилення у момент часу. Додатне значення цієї змінної доцільно трактувати як кредит, тобто недоотримання ресурсу в минулому. Від’ємне значення відповідає боргу, тобто ситуації, коли агент уже отримав більшу частку доступу, ніж це випливало з цільового режиму обслуговування. Нульове значення означає, що агент перебуває в стані відносної рівноваги щодо очікуваної

частки. Таке подання є зручним тим, що дозволяє використовувати один узагальнений показник замість двох незалежних лічильників і водночас прямо пов’язує модель із процедурою вибору задачі на наступному кроці.

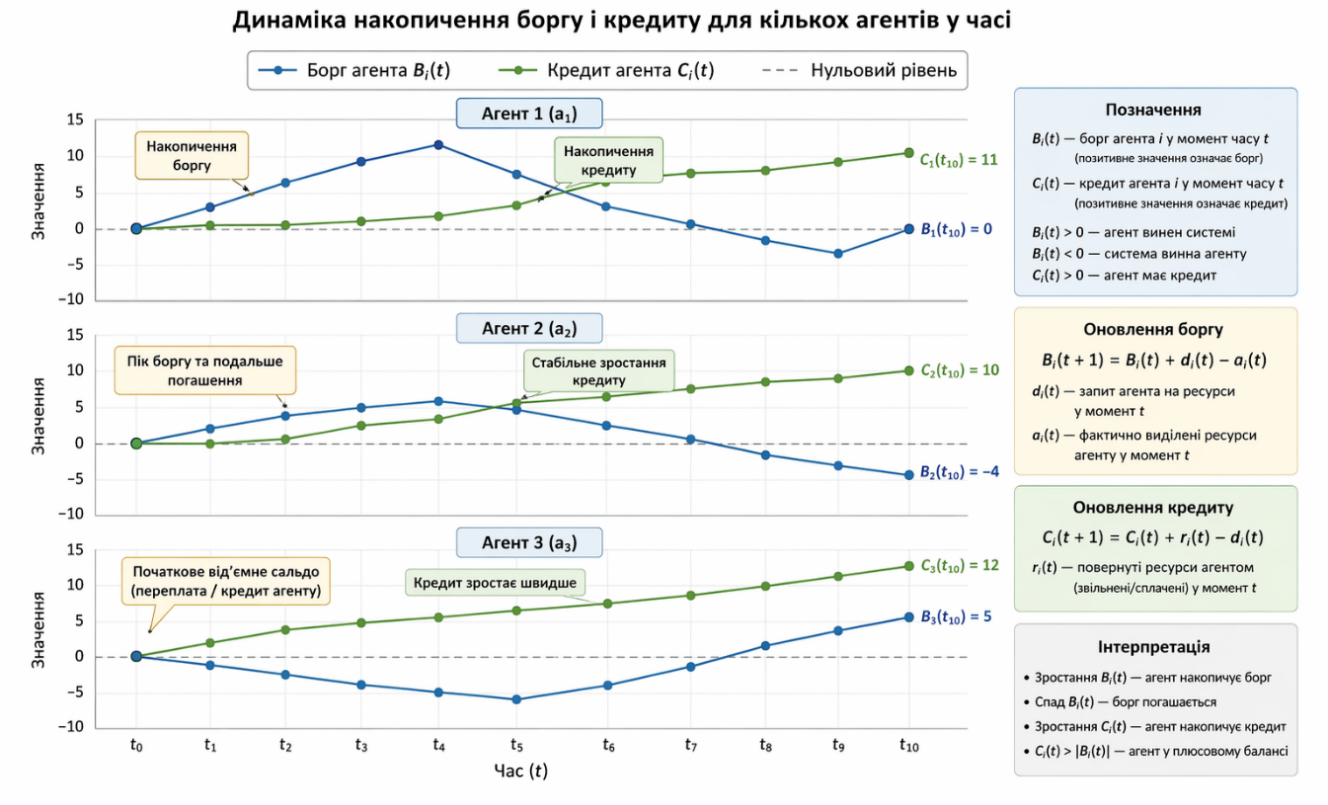
Оновлення показника має виконуватися після кожного кроку планування. Якщо на поточному кроці агент отримав обслуговування, його відхилення змінюється у бік зменшення кредиту або збільшення боргу. Якщо ж агент очікував, але не був обраний, його кредит поступово зростає, що створює передумови для подальшої компенсації. При цьому величина оновлення не повинна бути однаковою для всіх випадків. Вона має залежати від ваги агента, від обсягу реально наданого ресурсу, а також від того, наскільки суттєвим було це рішення з погляду поточного багаторесурсного балансу. Саме тому закон оновлення доцільно пов’язати з домінантною часткою та з політикою доступу, а не з простим фактом запуску заявки.

У практичному сенсі механізм боргу і кредиту виконує роль часового стабілізатора справедливості. Без нього навіть добре збалансований критерій вибору може відтворювати короткочасні перекоси, які з часом накопичуються і перетворюються на стійку нерівномірність. Наприклад, якщо агент із невеликими та швидко здійсненними заявками кілька разів поспіль опиняється у вигіднішій позиції, він може стабільно витісняти агента з більшими або складнішими заявками. У такому випадку домінантна частка відображатиме лише поточне становище, але не компенсуватиме попередні втрати другого агента. Наявність накопиченого кредиту дає змогу поступово вирівнювати таку ситуацію без необхідності жорсткого перерозподілу чи глобального перерахунку всієї черги.

Важливою перевагою запропонованої моделі є її поступовий характер. Компенсація відхилень не виконується різко або одномоментно, а реалізується малими кроками під час кожного нового рішення. Це дозволяє уникнути нестійкості, коли система надто агресивно намагається повернути баланс і через це створює протилежний перекіс. У межах розроблюваного методу параметри оновлення мають задавати інтенсивність компенсації. За м’якого режиму відхилення вирівнюються повільніше, але система зберігає вищу стабільність і

кращу утилізацію. За жорсткішого режиму перекоси усуваються швидше, однак зростає ризик локального зниження ефективності через підвищену чутливість до попередньої історії. Унаслідок цього модель боргу і кредиту надає можливість налаштовувати баланс між справедливістю та продуктивністю без зміни загальної логіки методу.

Ще однією важливою властивістю є те, що показник відхилення повинен мати обмежений вплив на вибір задачі. Якщо роль накопиченого кредиту зробити надто великою, агент із давнім недообслуговуванням може отримувати непропорційно високу перевагу навіть тоді, коли його поточний запуск є невигідним або створює надмірний тиск на критичний ресурс. Якщо ж вплив боргу і кредиту буде занадто слабким, механізм не зможе виконувати свою компенсаційну функцію. Через це в подальшому алгоритмі показник доцільно використовувати не ізольовано, а разом із домінантною часткою агента. Перша компонента відображає часову справедливість, друга - поточний багаторесурсний стан. Лише їх поєднання створює основу для зваженого та стійкого вибору.



Рисунка 2.2 – Динаміка накопичення боргу і кредиту для кількох агентів у часі.

Окремо слід врахувати ситуацію, коли агент тимчасово не може бути обслужений не через несправедливість, а через нездійсненність запуску його задачі. У такому випадку кредит не повинен зростати безконтрольно, інакше система накопичуватиме компенсацію за ті періоди, коли фактичне обслуговування було неможливим через об'єктивні обмеження середовища. Саме тому в подальшій реалізації доцільно розрізняти очікування через конкуренцію та очікування через нездійсненність запуску.

Отже, механізм боргу і кредиту в межах цієї роботи доцільно розглядати як інструмент компенсації часових перекосів, що виникають у послідовному багаторесурсному розподілі. Він дозволяє враховувати не лише поточне положення агента, а й історію його обслуговування, поступово вирівнюючи накопичені відхилення (рисунок 2.2). У підсумку така модель створює часову складову справедливості й доповнює модель домінантної частки, яка описує багаторесурсний стан агента в конкретний момент. Це забезпечує основу для подальшого формування правила вибору задачі та перевірки здійсненності запуску в межах цілісного методу послідовного розподілу ресурсів.

2.4 Метод вибору задачі та перевірки здійсненності запуску

Після формування моделі домінантної частки та механізму компенсації відхилень необхідно визначити, яким чином ці елементи використовуватимуться під час прийняття конкретного рішення планувальником. У послідовному розподілі ресурсів недостатньо лише знати, який агент є недообслугованим або яка його поточна домінантна частка. На кожному кроці система повинна обрати таку заявку, запуск якої одночасно наближає розподіл до справедливого стану і не порушує реальних обмежень середовища. Саме тому в межах запропонованого підходу метод вибору задачі має спиратися на поєднання двох складових: правила ранжування кандидатів та процедури перевірки здійсненності запуску.

На вхід планувальника в кожний момент часу надходить множина активних заявок, які належать різним агентам і характеризуються власними ресурсними

потребами. Якщо система орієнтуватиметься лише на порядок надходження заявок, це швидко призведе до перекосів, оскільки перші в черзі агенти отримуватимуть стабільну перевагу незалежно від попередньої історії обслуговування. Якщо ж вибір виконувати лише на основі накопиченого кредиту, з’являється ризик надмірної компенсації, коли локально вигідні для справедливості рішення починають погіршувати загальну керованість і використання ресурсів. Через це метод вибору повинен одночасно враховувати поточне багаторесурсне положення агента та його часове відхилення від цільового режиму.

У межах цієї роботи доцільно вважати, що для кожного агента на кроці планування формується узагальнений пріоритетний показник. Його значення має зростати, якщо агент тривалий час недоотримував ресурс, і зменшуватися, якщо агент уже займає велику домінантну частку або нещодавно отримував обслуговування частіше за інших. У такому випадку метод не зводиться ні до простого вирівнювання часток, ні до механічного обслуговування боргу. Рішення формується як компроміс між поточним ресурсним станом і накопиченою історією розподілу. Це дозволяє поєднати багаторесурсну та часову складові справедливості в єдиному правилі вибору.

Практична процедура вибору кандидата може бути подана як послідовність кроків. Спочатку формується множина агентів, які мають активні заявки в черзі. Далі для кожного з них обчислюється поточний пріоритетний показник. Після цього агенти впорядковуються за спаданням або зростанням значення цього показника залежно від прийнятої форми критерію, а планувальник переходить до перевірки заявок у сформованому порядку. Першим розглядається кандидат, який у поточний момент є найбільш доцільним з погляду справедливого розподілу. Проте сам факт того, що агент має найвищий пріоритет, ще не означає можливості негайного запуску його задачі. Саме на цьому етапі ключову роль починає відігравати перевірка здійсненності.

Необхідність такої перевірки пояснюється тим, що в реальному середовищі справедливий вибір і здійсненний вибір не завжди збігаються. Агент може мати високий кредит і малу домінантну частку, але його поточна заявка вимагатиме

такого поєднання ресурсів, яке відсутнє в системі на даному кроці. Причиною можуть бути нестача оперативної пам’яті, відсутність потрібного типу GPU, конфлікт із правилами розміщення або фрагментація доступного ресурсу. Якщо планувальник ігноруватиме ці фактори, він або ухвалюватиме нереалізовні рішення, або зациклюватиметься на одному кандидатові, не просуваючи систему до фактичного виконання задач.

Тому після вибору кандидата за пріоритетом необхідно перевірити, чи задовольняє його заявка поточним ресурсним і логічним обмеженням. Якщо всі умови виконуються, заявка запускається, а стан системи оновлюється. Якщо ж запуск виявляється неможливим, планувальник переходить до наступного кандидата зі списку. Така схема є принципово важливою для прикладної працездатності методу, оскільки вона розділяє два різні аспекти рішення: справедливе ранжування та фактичну здійсненність. Унаслідок цього система зберігає логіку збалансованого вибору, але не втрачає здатності працювати в умовах реальних інфраструктурних обмежень.

Важливою властивістю такого підходу є те, що нездійсненність запуску не повинна автоматично тлумачитися як несправедливість. Якщо заявка агента не може бути виконана через об’єктивні обмеження середовища, це не означає, що система зобов’язана безмежно підсилювати її пріоритет. Саме тому процедура перевірки здійсненності має бути пов’язана з механізмом оновлення боргу і кредиту. У випадку конкуренції між здійсненними заявками кредит може зростати, компенсуючи недообслуговування. У випадку тимчасової технічної нездійсненності зростання кредиту повинно бути стриманим або спеціально скоригованим. Такий підхід не дозволяє накопичувати штучну перевагу для задач, які на поточному етапі все одно не можуть бути розміщені в системі.

Ще одна перевага запропонованого методу полягає в його гнучкості. Узагальнений пріоритетний показник можна налаштовувати через вагові коефіцієнти, змінюючи чутливість системи до часових відхилень або до поточного багаторесурсного дисбалансу. Завдяки цьому з’являється можливість підлаштовувати поведінку планувальника під конкретний характер

інфраструктури. Для середовищ, де критичним є запобігання голодуванню, можна підсилити вплив кредиту. Для середовищ із жорстким дефіцитом ресурсів і високою вартістю простою більш доцільним може бути обережніше коригування, за якого перевага віддається рішенням із кращою здійсненністю. У підсумку метод не є жорстко фіксованою схемою, а задає керований механізм прийняття рішень у межах єдиної логіки справедливого розподілу.

На алгоритмічному рівні цей підхід можна описати як ітеративну процедуру. На кожному кроці формується множина активних кандидатів, для кожного обчислюється пріоритетний показник, виконується ранжування, після чого кандидати послідовно перевіряються на здійсненність запуску. Перший кандидат, який задовольняє всім умовам, отримує обслуговування, а система оновлює доступні ресурси, домінантні частки та накопичені відхилення. Якщо жоден кандидат не є здійсненним, планувальник переходить у режим очікування наступної події: звільнення ресурсу, завершення задачі або надходження нової заявки. Така процедура поєднує простоту реалізації з достатньою гнучкістю для роботи в динамічному багатоагентному середовищі.

Отже, метод вибору задачі в межах цієї роботи ґрунтується на поєднанні ранжування агентів за узагальненим пріоритетним показником і перевірки фактичної здійсненності запуску. Такий підхід дозволяє уникнути як сліпого дотримання черги, так і абстрактного вирівнювання, відірваного від реального стану інфраструктури. У підсумку формується прикладний механізм, який підтримує справедливість у часі, враховує багаторесурсний характер навантаження та зберігає працездатність у середовищі з дискретними ресурсами й політиками розміщення.

2.5 Узагальнений метод послідовного розподілу ресурсів

Побудовані на попередніх етапах складові дають змогу перейти до цілісного подання запропонованого методу послідовного розподілу ресурсів у багатоагентній комп’ютерній системі. На цьому рівні окремі елементи вже не

розглядаються ізольовано. Формалізоване середовище задає множину агентів, ресурсів і заявок, модель домінантної частки визначає багаторесурсне положення кожного агента, механізм боргу і кредиту відображає часові відхилення справедливості, а процедура вибору задачі з перевіркою здійсненності забезпечує прикладну реалізацію рішення. У підсумку метод має поєднати ці складові в єдину послідовність дій, яка дозволяє приймати рішення в режимі реального надходження заявок та підтримувати збалансований доступ до спільних ресурсів.

Узагальнений метод доцільно розглядати як ітеративний процес, що виконується на кожному кроці планування. Вхідними даними для нього є поточний стан системи, множина активних заявок, залишкові обсяги ресурсів, історія попереднього обслуговування агентів і параметри політики доступу. Результатом роботи є вибір конкретної заявки для запуску або рішення про відсутність здійсненного кандидата на поточному кроці. На відміну від простих схем обслуговування черги, запропонований метод не спирається лише на момент надходження заявки або локальний пріоритет. Рішення формується на основі узгодження трьох умов: агент не повинен мати надмірної домінантної частки, його накопичене відхилення має бути враховане, а сама заявка повинна бути здійсненною в межах поточного стану інфраструктури.

Початковим етапом роботи методу є оновлення інформації про стан системи. На цьому кроці враховуються завершення раніше запущених задач, звільнення ресурсів, надходження нових заявок та зміни в черзі очікування. Після цього формується множина активних агентів, для яких наявні заявки, що можуть бути розглянуті на поточному кроці. Для кожного з них обчислюються службові показники, необхідні для прийняття рішення: нормовані частки використання ресурсів, домінантна частка, накопичене відхилення від бажаного режиму обслуговування та, за потреби, ваговий коефіцієнт відповідно до політики системи. Лише після оновлення цих характеристик система переходить до ранжування кандидатів.

На другому етапі формується узагальнений пріоритетний показник для кожного агента. Він має враховувати одночасно просторову та часову складові

справедливості. Просторова складова пов’язана з домінантною часткою: чим більшу частину найкритичнішого для себе ресурсу вже займає агент, тим обережніше слід надавати йому подальший доступ. Часова складова пов’язана з накопиченим кредитом або боргом: недообслугований агент повинен поступово підвищувати свої шанси, а агент із надлишковим обслуговуванням - втрачати частину переваги. У результаті формується інтегральна оцінка, яка і використовується як основа для впорядкування кандидатів.

Після ранжування кандидатів виконується перевірка здійсненності запуску їхніх заявок. Спочатку розглядається агент із найвищою доцільністю за критерієм справедливості, після чого перевіряється, чи може його заявка бути виконана за поточних ресурсних залишків і правил розміщення. Якщо запуск можливий, заявка переходить у стан виконання, а система оновлює доступні ресурси та накопичені показники агента. Якщо ж запуск виявляється неможливим, система переходить до наступного кандидата в ранжованому списку. Така логіка дозволяє не втрачати орієнтацію на справедливий вибір, але водночас не блокувати роботу інфраструктури через одного технічно нездійсненного кандидата.

Суттєвою особливістю запропонованого методу є те, що оновлення службових показників відбувається після кожного результативного кроку. Агент, заявка якого була запущена, змінює свою домінантну частку та баланс боргу і кредиту відповідно до реально отриманого ресурсу. Агенти, які залишилися в очікуванні, також не залишаються без змін: для них може зростати компенсаційний кредит, якщо їх недообслуговування пов’язане саме з конкуренцією, а не з технічною нездійсненністю запуску. Унаслідок цього метод підтримує не разову, а накопичувальну форму справедливості, коли баланс вирівнюється поступово впродовж низки рішень, а не лише в межах одного кроку.

Ще однією важливою властивістю є адаптивність методу до змін навантаження. У багатоагентному середовищі не можна припускати, що склад активних агентів або профілі їх споживання залишаються сталими. Одні агенти можуть тимчасово ставати домінуючими, інші - переходити в пасивний режим, а треті - з’являтися в системі лише в окремих інтервалах часу. Запропонований метод

не потребує повного глобального перерахунку після кожної такої зміни. Корекція виконується локально на кожному кроці, що робить підхід придатним до динамічних умов функціонування. Це особливо важливо для систем, у яких навантаження має хвилеподібний характер або залежить від зовнішніх подій.

На концептуальному рівні узагальнений метод можна подати як цикл із п’яти основних стадій: оновлення стану системи, обчислення службових характеристик агентів, ранжування кандидатів, перевірка здійсненності запуску та коригування стану після прийнятого рішення. Така структура є достатньо простою для алгоритмічної реалізації, але водночас охоплює всі ключові аспекти задачі послідовного розподілу. Вона враховує як багаторесурсний характер середовища, так і необхідність компенсації часових перекосів, а також зберігає зв’язок із реальними обмеженнями інфраструктури.

Для повнішого опису метод доцільно подати також у вигляді узагальненого алгоритму. На першому кроці ініціалізуються параметри системи та службові змінні агентів. Далі виконується циклічний перегляд подій: надходження заявок, завершення задач, вивільнення ресурсу. Після оновлення стану обчислюються показники агентів, формується список кандидатів і виконується вибір першого здійсненного варіанта. Якщо такий варіант знайдено, заявка запускається, а параметри системи оновлюються; якщо ні, метод переходить у режим очікування наступної події. У результаті вся логіка послідовного розподілу зводиться до повторюваної, але достатньо гнучкої процедури, що забезпечує стабільну роботу системи в умовах динамічного багаторесурсного навантаження.

Отже, узагальнений метод послідовного розподілу ресурсів поєднує формалізоване подання багатоагентного середовища, критерій домінантної частки, механізм боргу і кредиту та процедуру перевірки здійсненності запуску. Його практична цінність полягає в тому, що він дозволяє підтримувати справедливість не лише як локальну властивість одного рішення, а як стійку характеристику всього процесу розподілу в часі. У підсумку запропонований метод створює цілісну основу для подальшої алгоритмічної та програмної реалізації, а також для

експериментальної перевірки його ефективності в умовах багатоагентного навантаження.

2.6 Висновки до розділу 2

У другому розділі було сформовано теоретико-методичну основу методу послідовного розподілу ресурсів у багатоагентній комп’ютерній системі. Насамперед було виконано формалізацію середовища, у межах якого приймаються рішення планування. Багатоагентну систему подано як сукупність агентів, заявок, типів ресурсів і обмежень запуску, що змінюються в часі. Такий підхід дозволив перейти від загального опису предметної області до впорядкованої моделі, придатної для подальшого алгоритмічного та програмного опрацювання.

У межах формалізації було показано, що для практичного розподілу ресурсів недостатньо враховувати лише їх сумарну доступність. Реальне середовище характеризується додатковими умовами здійсненності запуску, серед яких важливе місце займають вимоги до конкретних типів ресурсів, правила розміщення, обмеження сумісності та структурна фрагментація інфраструктури. Унаслідок цього простір допустимих рішень виявився вужчим за простір теоретично можливих розподілів, а сама задача планування потребувала поєднання критерію справедливості з процедурою перевірки фактичної здійсненності.

Далі було обґрунтовано доцільність використання моделі домінантної частки як базового критерію багаторесурсної справедливості. Показано, що в неоднорідному середовищі орієнтація лише на один ресурс не дає коректного уявлення про реальне положення агента в системі. Саме тому для зіставлення агентів було запропоновано враховувати той ресурс, за яким їхнє навантаження є найбільшим у відносному вимірі. Це дозволило отримати узагальнений показник, придатний для порівняння агентів із різними профілями споживання та різною структурою заявок.

Окрему увагу було приділено часовій складовій справедливості. Було встановлено, що навіть коректний багаторесурсний критерій не гарантує

збалансованого розподілу на часовому інтервалі, якщо не враховується історія попереднього обслуговування. Для усунення цього недоліку запропоновано модель компенсації відхилень на основі механізму боргу і кредиту. Її зміст полягає в накопиченні службового показника, який відображає, наскільки фактичний доступ агента до ресурсу відрізняється від очікуваного. Це дало змогу пов'язати окремі рішення планувальника з довгостроковим балансом справедливості, а також створило основу для поступового вирівнювання перекосів без різких і дестабілізуючих перерозподілів.

Наступним результатом розділу стало формування методу вибору задачі та перевірки здійсненності запуску. Було показано, що рішення про обслуговування має спиратися не лише на поточну чергу або накопичене відхилення, а на узагальнений пріоритетний показник, який поєднує просторову й часову складові справедливості. Водночас обов'язковою частиною процедури було визначено перевірку фактичної можливості запуску заявки за поточного стану ресурсів і правил розміщення. Унаслідок цього вдалося розмежувати два аспекти задачі: справедливий вибір кандидата та технічну здійсненність рішення, що є принципово важливим для прикладного використання методу.

На завершальному етапі всі розглянуті компоненти було об'єднано в узагальнений метод послідовного розподілу ресурсів. Його логіка побудована як ітеративний цикл, у межах якого виконується оновлення стану системи, обчислення службових характеристик агентів, ранжування кандидатів, перевірка здійсненності запуску та коригування параметрів після прийнятого рішення. Така структура дозволяє адаптуватися до змін навантаження, не вимагає повного глобального перерахунку після кожної події та зберігає зв'язок між критерієм справедливості й реальними умовами функціонування інфраструктури.

Крім того, сформований підхід дав змогу чіткіше окреслити внутрішню роль кожного елемента в загальній структурі методу. Домінантна частка не розглядається ізольовано, а працює як інструмент виявлення поточного співвідношення між агентами в системі. Механізм боргу і кредиту не зводиться лише до технічного накопичення числових значень, а виконує функцію

згладжування довгострокових перекосів, які неминуче виникають під час тривалої експлуатації. Перевірка здійсненності запуску, у свою чергу, виступає не допоміжною перевіркою, а повноцінним бар'єром між формально бажаним і реально можливим рішенням. У поєднанні ці складові формують не набір незалежних процедур, а єдиний механізм, у якому кожен компонент підсилює інші та забезпечує цілісність роботи планувальника.

Важливим підсумком другого розділу також стало те, що запропонований метод зберігає прикладну гнучкість. Його структура не прив'язується жорстко до одного конкретного типу інфраструктури або до одного сценарію розподілу навантаження. За потреби окремі елементи можуть деталізуватися, параметризуватися або адаптуватися до специфіки середовища без порушення загальної логіки роботи. Це особливо важливо для багатоагентних комп'ютерних систем, у яких змінюються не лише обсяги доступних ресурсів, а й самі правила взаємодії між задачами, агентами та обчислювальними вузлами.

Окремо слід підкреслити, що побудована у другому розділі логіка створює підґрунтя для переходу від концептуального опису до етапу практичного відтворення. Саме на цьому рівні було закладено ті залежності, правила й службові показники, які надалі можуть бути безпосередньо реалізовані у вигляді структур даних, алгоритмів обробки подій, процедур оновлення стану та модулів прийняття рішень.

У підсумку в другому розділі було розроблено цілісну методичну основу запропонованого рішення. Отриманий результат полягає не лише у формальному описі окремих елементів, а у побудові зв'язаної моделі, в якій домінантна частка відповідає за багаторесурсну оцінку положення агента, механізм боргу і кредиту - за компенсацію часових перекосів, а процедура перевірки здійсненності - за прикладну придатність прийнятих рішень. Це дозволяє розглядати сформований метод як підготовлену основу для подальшої алгоритмічної деталізації, програмної реалізації та експериментальної перевірки його ефективності.

3 АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ РЕАЛІЗАЦІЇ МЕТОДУ

3.1 Архітектура програмного засобу

Після розроблення узагальненого методу послідовного розподілу ресурсів необхідно перейти до його програмного подання. На цьому етапі головним завданням є побудова такої архітектури програмного засобу, яка дозволяє реалізувати логіку справедливого планувальника, забезпечити послідовне опрацювання заявок, виконувати перевірку здійсненності запуску та зберігати історію обслуговування агентів. Архітектура в межах цієї роботи розглядається не лише як схема взаємодії модулів, а як структурна основа, від якої залежить стабільність роботи, розширюваність, керованість і можливість подальшого експериментального аналізу.

Програмний засіб доцільно будувати за модульним принципом. Це пояснюється тим, що окремі функції системи мають різне призначення й різну внутрішню логіку. Частина компонентів відповідає за зберігання стану середовища, частина - за обчислення критеріїв справедливості, частина - за вибір кандидата на запуск, а окремий блок повинен контролювати здійсненність рішення в межах поточних ресурсних обмежень. Якщо об’єднати ці функції в єдиний нерозділений програмний фрагмент, система втрачатиме прозорість, а її поведінка стане складнішою для перевірки, модифікації та налагодження. Саме тому архітектурне розділення компонентів є необхідною умовою для практичної реалізації методу.

У межах запропонованого рішення програмний засіб доцільно подати як сукупність п’яти основних підсистем. Першою з них є підсистема керування станом середовища. Її призначення полягає у зберіганні даних про активних агентів, чергу заявок, поточні залишки ресурсів, параметри політики доступу та результати попередніх кроків планування. Саме цей компонент формує цілісне подання поточного стану системи й надає його іншим модулям для оброблення. Без нього неможливо коректно реалізувати ні оцінювання справедливості, ні перевірку

здійсненності запуску, оскільки обидва ці процеси спираються на поточний та накопичений контекст.

Другою підсистемою є модуль аналітики агентів і ресурсів. У ньому виконуються обчислення, пов’язані з нормованими частками використання ресурсів, визначенням домінантної частки, оцінюванням поточного навантаження агента та підготовкою допоміжних показників для прийняття рішення. Саме в цьому модулі математичні положення, розроблені у попередньому розділі, переходять у форму прикладних програмних процедур. Завдяки відокремленню такого компонента забезпечується можливість змінювати або уточнювати формули оцінювання без необхідності перебудови всієї системи.

Третьою складовою виступає модуль компенсації часових відхилень. Його функція пов’язана з підтримкою механізму боргу і кредиту для кожного агента. У межах цього модуля зберігаються та оновлюються службові показники, що відображають недообслуговування або надлишкове обслуговування агентів у попередніх кроках. Він отримує інформацію про результати запусків, аналізує історію обслуговування та формує коригувальні значення, які надалі використовуються в механізмі ранжування кандидатів. Така побудова дозволяє чітко відокремити часову складову справедливості від просторової, пов’язаної з домінантною часткою.

Четвертим компонентом є ядро планувальника. Саме воно реалізує основну логіку вибору задачі на поточному кроці. У цьому модулі поєднуються результати роботи аналітичного блоку та механізму компенсації відхилень, формується узагальнений пріоритетний показник для кандидатів, виконується їх ранжування та ініціюється процедура перевірки здійсненності. Ядро планувальника можна вважати центральною частиною системи, оскільки саме в ньому абстрактний метод розподілу ресурсів перетворюється на конкретне керуюче рішення.

П’ятим компонентом є модуль перевірки здійсненності запуску. Його наявність є принципово важливою, оскільки навіть найбільш справедливий з погляду ранжування кандидат може бути технічно нездійсненним у поточному стані системи. У межах цього модуля перевіряються наявність необхідних ресурсів,

сумісність із правилами розміщення, відповідність топологічним обмеженням, доступність спеціалізованого обладнання та інші практичні умови запуску. У разі позитивного результату ядро планувальника підтверджує вибір, а в разі негативного - переходить до наступного кандидата. Саме цей компонент забезпечує прикладну працездатність усієї системи в умовах дискретних ресурсів і політик розміщення.

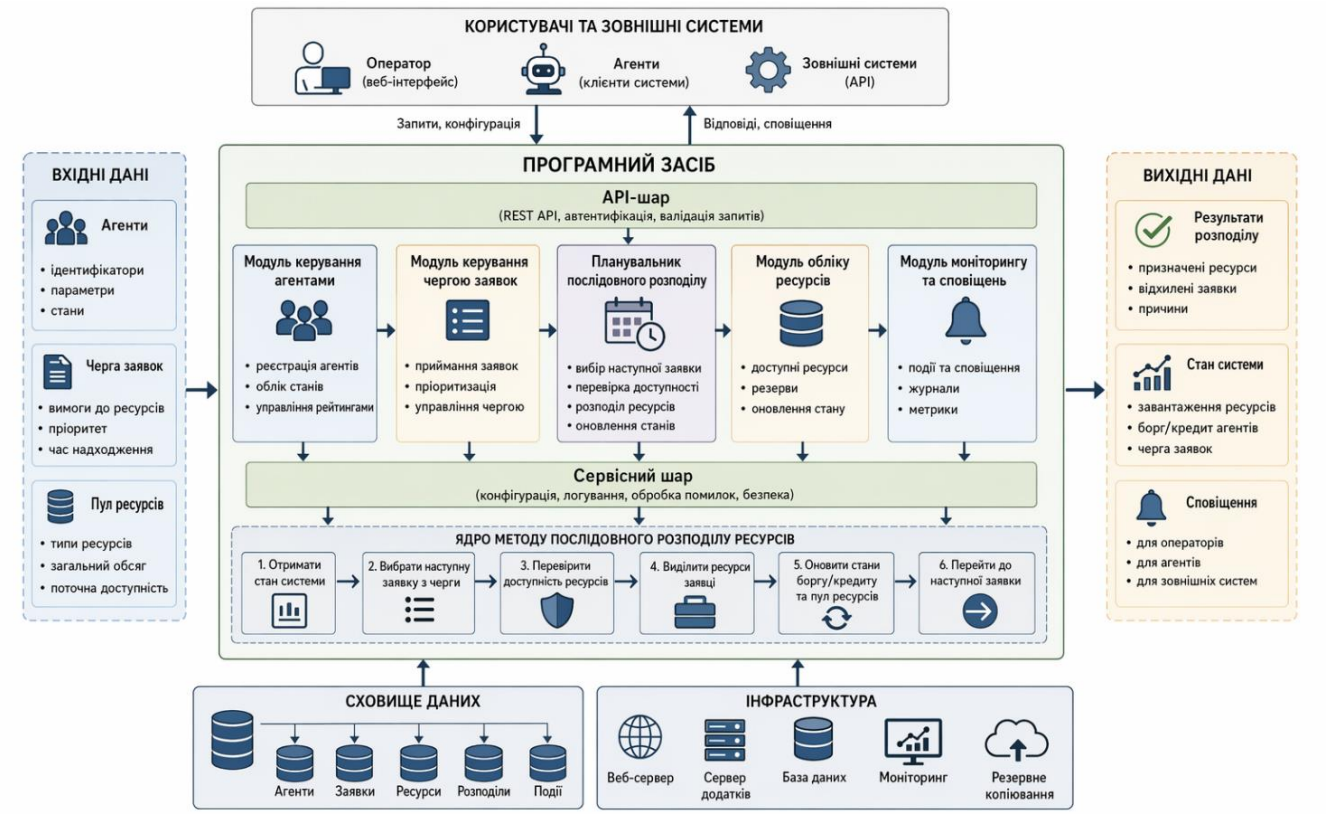


Рисунок 3.1 – Загальна архітектура програмного засобу для реалізації методу послідовного розподілу ресурсів.

Окрім перелічених підсистем, до складу програмного засобу доцільно включити допоміжний інтерфейс взаємодії з експериментальним середовищем. Його функція полягає в поданні вхідних подій у зручному форматі, фіксації результатів роботи планувальника, накопиченні журналів виконання та передаванні даних до підсистеми аналізу експериментів. Хоча цей компонент не є частиною ядра методу, його роль є важливою для подальшої перевірки

ефективності рішення, оскільки без засобів спостереження і фіксації результатів неможливо провести повноцінне оцінювання поведінки системи.

Взаємодію між основними модулями доцільно організувати у вигляді послідовного циклу оброблення подій. Після надходження нової заявки або після завершення поточної задачі підсистема керування станом оновлює внутрішню модель середовища(рисунок 3.1). Далі модуль аналітики ресурсів обчислює поточні частки використання та визначає домінантні частки агентів. Після цього модуль компенсації часових відхилень актуалізує баланс боргу і кредиту. На основі отриманих значень ядро планувальника формує ранжований список кандидатів, а модуль перевірки здійсненності по черзі оцінює можливість запуску заявок. Після успішного вибору система повертається до початкового стану циклу з оновленими параметрами. Така схема забезпечує чіткий розподіл відповідальності між компонентами та підтримує послідовність логіки, закладеної в методі.

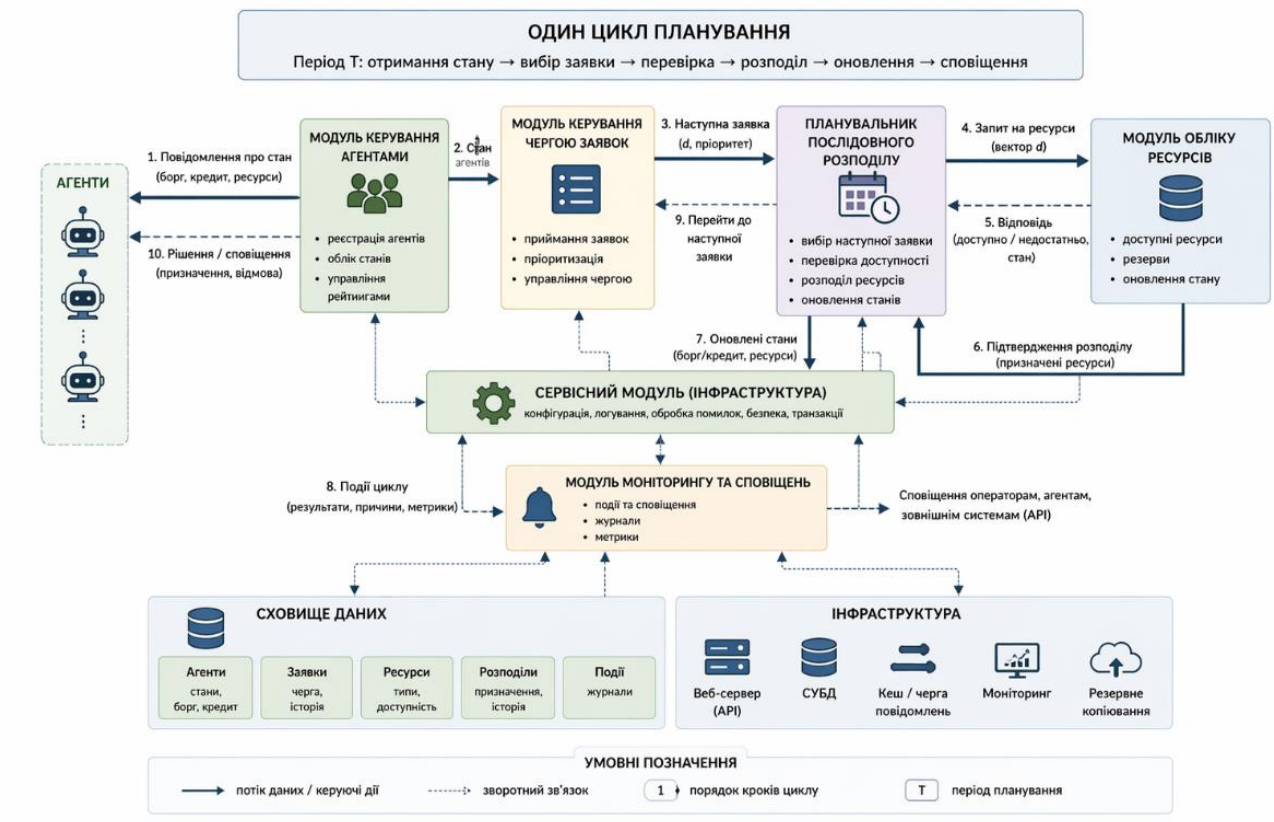


Рисунок 3.2 – Схема взаємодії модулів програмного засобу в межах одного циклу планування.

Для програмної реалізації такої архітектури важливими є також вимоги до внутрішнього представлення даних. Агент доцільно описувати як окремий об’єкт із ідентифікатором, набором активних заявок, вектором використаних ресурсів, домінантною часткою та значенням накопиченого відхилення. Заявка має містити інформацію про власника, ресурсні потреби, момент надходження, стан оброблення та додаткові умови запуску. Стан ресурсного середовища доцільно подавати у вигляді окремої структури, що містить поточні залишки доступних ресурсів і множину активних обмежень (рисунок 3.2). Така організація даних забезпечує узгодженість між математичним описом методу та його програмним втіленням.

Ще однією важливою вимогою до архітектури є можливість її розширення. У подальшому може виникнути потреба змінити правило ранжування, додати нові типи ресурсів, ускладнити політику компенсації або підключити зовнішній механізм оцінювання. Якщо архітектура спочатку побудована модульно, такі зміни можна вносити локально, не порушуючи роботи всієї системи. Це особливо важливо для експериментального програмного засобу, де під час перевірки гіпотез часто виникає потреба у варіаціях параметрів і внутрішніх правил.

У підсумку архітектура програмного засобу для реалізації методу послідовного розподілу ресурсів повинна забезпечувати поєднання трьох ключових властивостей: структурної зрозумілості, функціональної достатності та гнучкості до подальших змін. Запропоноване модульне подання дозволяє реалізувати логіку справедливого планувальника, зберегти зв’язок із математичною моделлю попереднього розділу та створити придатну основу для подальшої алгоритмічної деталізації. Це дозволяє перейти до наступного підрозділу, у якому буде розглянуто безпосередньо алгоритм роботи справедливого планувальника в межах побудованої архітектури.

3.2 Алгоритм роботи справедливого планувальника

Після визначення архітектури програмного засобу необхідно деталізувати внутрішню логіку ядра системи, тобто алгоритм роботи справедливого планувальника. Саме цей алгоритм забезпечує перехід від формалізованого опису середовища та критеріїв справедливості до конкретного рішення про запуск заявки. У межах запропонованого підходу планувальник не обмежується простим переглядом черги або вибором першого доступного кандидата. Його робота ґрунтується на поєднанні поточного багаторесурсного стану агентів, накопичених часових відхилень і результатів перевірки здійсненності запуску.

Алгоритм доцільно реалізувати як повторюваний цикл, що запускається після кожної значущої події в системі. Такою подією може бути надходження нової заявки, завершення раніше запущеної задачі, вивільнення ресурсу або зміна службових параметрів середовища. На початковому етапі цикл отримує від підсистеми керування станом оновлену інформацію про доступні ресурси, перелік активних агентів, склад черги та поточний стан уже виконуваних задач. Після цього формується множина кандидатів, заявки яких можуть бути розглянуті на даному кроці планування.

Для кожного агента з цієї множини алгоритм обчислює набір службових характеристик. Насамперед визначаються нормовані частки використання ресурсів і на їх основі - домінантна частка, яка відображає найбільш критичний для агента ресурсний вимір. Далі актуалізується показник боргу і кредиту, що характеризує накопичене відхилення від бажаного режиму обслуговування. Після цього формується узагальнений пріоритетний показник, який поєднує поточний багаторесурсний стан агента з його часовою історією. Саме це значення стає основою для ранжування кандидатів.

На наступному кроці кандидати впорядковуються відповідно до обчисленого пріоритету. Першим розглядається той агент, для якого запуск заявки є найбільш доцільним з погляду підтримання справедливості. Проте рішення ще не вважається остаточним. Для вибраного кандидата алгоритм передає заявку до модуля

перевірки здійсненності, де встановлюється, чи достатньо в системі вільних ресурсів, чи виконується потрібна конфігурація розміщення, чи не суперечить запуск локальним політикам доступу. Якщо всі умови задовольняються, заявка запускається. Якщо ж хоча б одна умова не виконується, алгоритм переходить до наступного кандидата у ранжованому списку.

Така схема є важливою з кількох причин. По-перше, вона зберігає орієнтацію на справедливість, оскільки ранжування відбувається на основі узагальненого показника, а не випадкового або суто часово́го принципу. По-друге, алгоритм залишається прикладно працездатним, оскільки кожне рішення проходить перевірку на фактичну реалізованість. По-третє, відмова від негайного запуску технічно нездійсненного кандидата не руйнує загальної логіки методу, а лише переносить увагу на наступний допустимий варіант. Унаслідок цього система уникає зациклення та зберігає здатність до просування черги навіть у складних ресурсних умовах.

Після успішного запуску заявки виконується етап оновлення стану. Із доступного пулу віднімаються ресурси, виділені для нової задачі, у структурі відповідного агента змінюється поточний вектор споживання, перераховується домінантна частка та коригується накопичене відхилення. Інші агенти також можуть зазнавати часткового оновлення, якщо їхній стан залежить від нового розподілу ресурсів або від факту чергового кроку планування. Якщо жоден кандидат не виявився здійсненним, алгоритм не виконує запуск, а переходить у режим очікування наступної події. Така поведінка дозволяє системі не приймати штучних рішень, які суперечили б фактичним обмеженням середовища.

Окрему увагу слід приділити питанню стабільності алгоритму в часі. Оскільки рішення приймаються послідовно, навіть локально правильний вибір може породжувати небажані ефекти, якщо не враховується накопичений результат попередніх кроків. Саме тому в алгоритмі не використовується виключно поточна інформація про доступні ресурси. Кожне рішення спирається також на історію обслуговування, відображену в механізмі боргу і кредиту. Це дозволяє послаблювати перекоси, які не завжди можна виявити лише за одним поточним

зрізом системи. У підсумку алгоритм підтримує не разову, а інтервальну справедливість, що є суттєвою перевагою для багатоагентного середовища.

На алгоритмічному рівні роботу планувальника доцільно подати у вигляді псевдокоду. На першому кроці відбувається читання поточного стану системи. Далі формується список агентів із активними заявками. Для кожного з них обчислюються домінантна частка, накопичене відхилення та інтегральний пріоритет. Після сортування кандидатів запускається цикл перевірки здійсненності, у межах якого перший допустимий кандидат передається на виконання. Після запуску виконується оновлення всіх необхідних службових змінних та повернення до початку циклу. Якщо допустимих кандидатів не знайдено, планувальник завершує поточний крок без запуску й очікує на зміну стану середовища.

Ще однією важливою особливістю алгоритму є його адаптивність. За рахунок зміни вагових коефіцієнтів і параметрів впливу боргу та кредиту можна змінювати характер поведінки системи без порушення загальної архітектури. Це дає змогу використовувати один і той самий базовий алгоритм у різних сценаріях: від середовищ, де критичним є запобігання голодуванню, до ситуацій, у яких пріоритетом виступає висока утилізація дефіцитного ресурсу. Унаслідок цього алгоритм справедливого планувальника можна розглядати як керовану процедуру, придатну до налаштування під конкретні умови функціонування.

У підсумку алгоритм роботи справедливого планувальника забезпечує практичну реалізацію запропонованого методу послідовного розподілу ресурсів. Його логіка ґрунтується на послідовному оновленні стану, оцінюванні багаторесурсного положення агентів, урахуванні накопичених часових відхилень і перевірці здійсненності запуску. Це дозволяє отримати не лише формально справедливий, а й технічно працездатний механізм прийняття рішень, придатний до використання в умовах динамічного багатоагентного навантаження.

3.3 Алгоритми обчислення домінантної частки та оновлення стану агентів

Для коректної роботи справедливого планувальника недостатньо лише сформувати загальну схему вибору заявки. Необхідно також детально визначити внутрішні алгоритмічні процедури, за допомогою яких система обчислює поточне положення кожного агента та коригує його стан після кожного кроку розподілу. У межах запропонованого підходу такими базовими процедурами виступають алгоритм обчислення домінантної частки та алгоритм оновлення стану агента. Саме вони забезпечують зв’язок між математичною моделлю багаторесурсної справедливості й реальною програмною логікою планувальника.

Обчислення домінантної частки починається з визначення поточного вектора споживання ресурсів агентом. Для кожного агента система повинна мати інформацію про те, який обсяг процесорного часу, оперативної пам’яті, графічних прискорювачів, мережевої пропускної здатності або інших ресурсів уже виділено його активним задачам. Ці значення самі по собі не дають змоги коректно порівнювати агентів, оскільки різні ресурси мають різні шкали вимірювання та різну загальну місткість. Саме тому наступним кроком виконується нормування: для кожного типу ресурсу поточний обсяг споживання ділиться на повну доступну місткість відповідного ресурсу в системі. У результаті формується набір відносних часток, приведених до єдиної шкали.

Після нормування алгоритм переходить до виділення найбільшого значення серед отриманих часток. Це значення й визначає домінантну частку агента. Така процедура є достатньо простою з обчислювального погляду, однак її роль у логіці планувальника є принциповою. Завдяки їй агент оцінюється саме за тим ресурсом, за яким він створює найбільше навантаження на систему. У багаторесурсному середовищі це дозволяє уникнути ситуацій, коли агент виглядає «легким» лише тому, що його навантаження аналізувалося за другорядним ресурсом, тоді як основний дефіцит створюється зовсім в іншому вимірі.

На алгоритмічному рівні процедура обчислення домінантної частки може бути організована як послідовний перегляд усіх типів ресурсів для кожного

активного агента. Спочатку ініціалізується змінна, що зберігатиме поточне максимальне значення. Далі для кожного ресурсу обчислюється нормована частка, після чого перевіряється, чи перевищує вона вже зафіксований максимум. Якщо так, максимум оновлюється. Після завершення перегляду всіх ресурсів отримане значення зберігається як домінантна частка агента. Така схема є придатною для практичної реалізації, оскільки не потребує складних обчислювальних структур і може виконуватися після кожної події, що змінює стан системи.

Проте домінантна частка є лише одним із параметрів, які повинні оновлюватися після запуску або завершення задачі. Не менш важливим є загальний стан агента. У межах цієї роботи стан агента доцільно розглядати як сукупність кількох взаємопов'язаних характеристик: набору активних заявок, поточного вектора споживання ресурсів, домінантної частки, накопиченого боргу або кредиту, а також службових параметрів, пов'язаних із політикою доступу. Після кожного результативного кроку планування хоча б частина цих характеристик змінюється. Саме тому потрібен окремий алгоритм, який забезпечує послідовне та узгоджене оновлення стану агента.

Якщо заявка агента була успішно запущена, алгоритм оновлення насамперед додає її ресурсні потреби до поточного вектора споживання. Далі виконується повторне обчислення нормованих часток і оновлення домінантної частки. Після цього коригується значення накопиченого відхилення, оскільки факт запуску означає, що агент отримав нову порцію ресурсу й його попередній кредит має бути зменшений або борг – збільшений залежно від обраної схеми інтерпретації. Одночасно оновлюється перелік активних заявок і службові часові мітки, якщо вони використовуються в реалізації. Унаслідок цього агент переходить у новий стан, який уже відображає наслідок ухваленого рішення.

Якщо ж у системі відбулася не подія запуску, а завершення задачі, алгоритм оновлення діє в протилежному напрямі. Із поточного вектора споживання віднімаються ресурси, що вивільнилися після завершення. Далі знову виконується нормування та перерахунок домінантної частки, оскільки для частини агентів звільнення навіть одного ресурсу може суттєво змінити їхнє відносне положення в

системі. За потреби коригується й накопичений показник відхилення, якщо в моделі передбачено вплив завершення задачі на часову рівновагу. Така послідовність дій гарантує, що внутрішній стан агента після кожної події залишатиметься узгодженим з реальним станом ресурсного середовища.

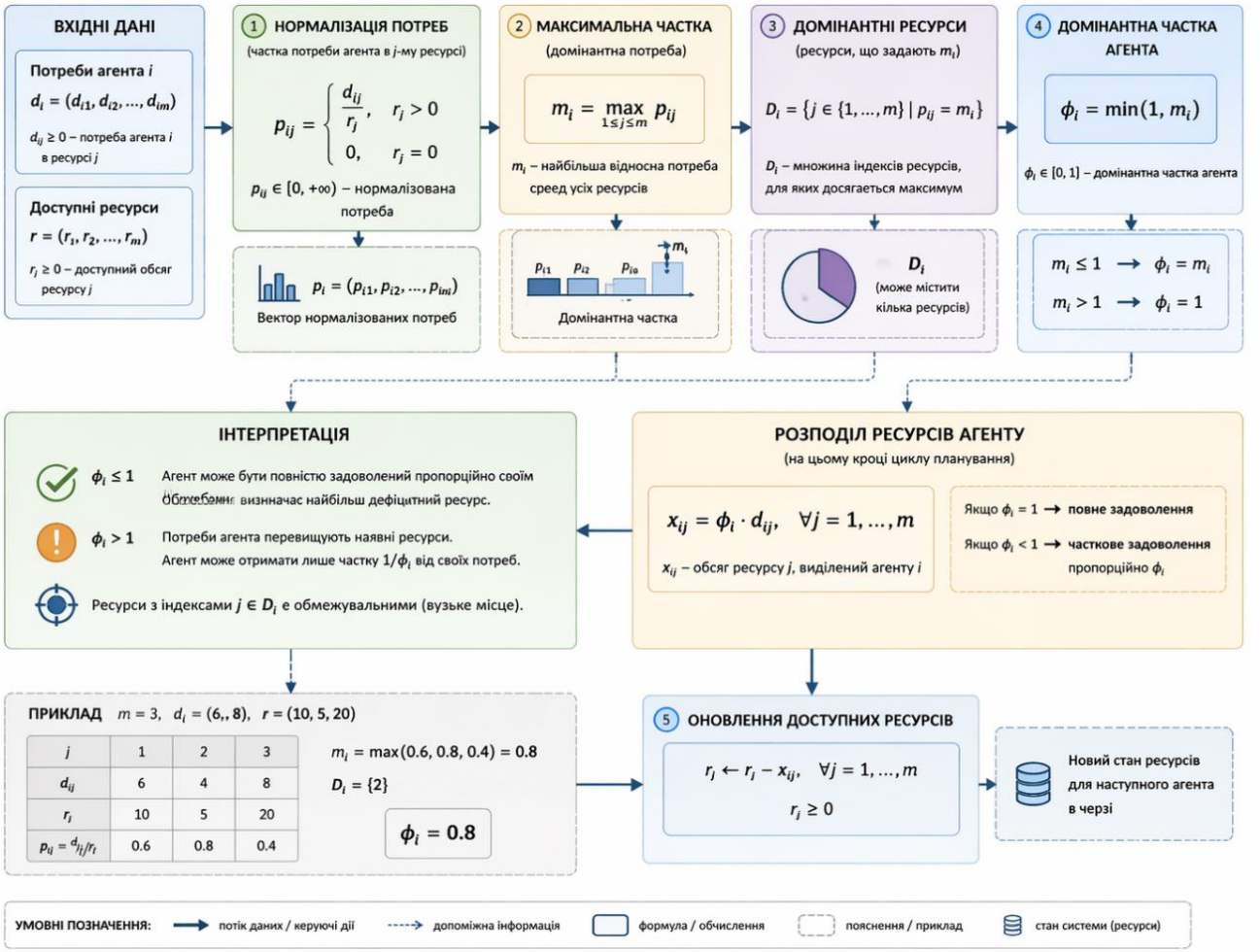


Рисунок 3.3 – Схема обчислення домінантної частки агента.

Окремої уваги потребує ситуація, коли агент має кілька активних заявок або коли в системі використовується черга внутрішніх задач одного агента. У такому випадку алгоритм оновлення повинен враховувати сукупний ефект усіх активних запусків, а не лише параметри однієї поточної заявки. Це означає, що вектор споживання формується як агреговане значення по всіх задачах агента, які перебувають у стані виконання (рисунок 3.3). Подібний підхід є більш коректним, оскільки саме загальне навантаження агента визначає його вплив на

багаторесурсну рівновагу системи. Унаслідок цього домінантна частка відображає не локальний, а фактичний інтегральний стан агента.

На програмному рівні обидві розглянуті процедури доцільно реалізовувати як окремі функції або методи сервісного класу, що працює зі станом агентів. Це дає змогу забезпечити повторне використання коду та уникнути дублювання логіки в різних частинах системи. Крім того, така організація спрощує тестування. Алгоритм обчислення домінантної частки можна перевіряти незалежно від роботи всього планувальника, подаючи на вхід контрольні набори значень ресурсного споживання. Аналогічно алгоритм оновлення стану агента можна тестувати на окремих сценаріях запуску, завершення та часткової зміни навантаження. Це підвищує надійність програмного засобу та полегшує подальше налагодження.

Ще однією важливою вимогою є узгодженість моменту оновлення. Якщо домінантна частка та службові параметри агента будуть оновлюватися несвоєчасно, планувальник може ухвалювати рішення на основі застарілого стану системи. Саме тому в реалізації доцільно дотримуватися правила, за яким будь-яка подія, що змінює розподіл ресурсу, має негайно ініціювати процедуру перерахунку параметрів відповідних агентів. Це особливо важливо для динамічних середовищ, де затримка навіть на один цикл може призводити до накопичення похибок у ранжуванні кандидатів.

Отже, алгоритми обчислення домінантної частки та оновлення стану агентів формують внутрішню операційну основу справедливого планувальника. Перший із них забезпечує коректне багаторесурсне оцінювання положення агента, другий - підтримує узгодженість його внутрішнього стану після кожної події в системі. У підсумку саме поєднання цих двох процедур дозволяє перетворити абстрактні критерії справедливості на конкретні програмні механізми, придатні для подальшої реалізації та експериментальної перевірки в багатоагентному середовищі.

3.4 Програмна реалізація основних модулів системи

Після визначення архітектури програмного засобу та алгоритмічної логіки справедливого планувальника необхідно перейти до опису програмної реалізації основних модулів системи. На цьому етапі метод послідовного розподілу ресурсів набуває завершеного прикладного вигляду, оскільки кожна з раніше сформованих теоретичних складових відображається у вигляді конкретного програмного компонента. У межах побудованого рішення програмна реалізація орієнтована на модульний підхід, що дозволяє розділити функції зберігання стану, аналітичного оброблення, ранжування агентів, перевірки здійсненності запуску та фіксації результатів роботи.

Базовим елементом системи виступає модуль представлення предметних сутностей. У ньому доцільно реалізувати класи або структури даних, що описують агента, заявку, ресурсне середовище та поточний стан системи. Об’єкт агента має містити ідентифікатор, перелік активних заявок, поточний вектор споживання ресурсів, значення домінантної частки, накопичене відхилення та службові параметри політики доступу. Об’єкт заявки має зберігати відомості про належність агенту, обсяг потреб у ресурсах, момент надходження, стан оброблення й додаткові обмеження запуску. Така побудова дає змогу перенести математичну модель попереднього розділу в програмне середовище без втрати логічної узгодженості між її складовими.

Другим важливим елементом є модуль керування станом системи. Його призначення полягає в накопиченні та оновленні інформації про поточний ресурсний стан, склад черги, множину активних агентів і результати попередніх кроків планування. У програмній реалізації цей модуль доцільно подати у вигляді окремого сервісного компонента, який надає іншим частинам системи доступ до актуального стану середовища. Саме через нього виконується реєстрація нових заявок, фіксація завершення задач, зміна залишків доступних ресурсів і передавання оновленого контексту до ядра планувальника. Завдяки відокремленню такого модуля забезпечується централізоване керування даними та зменшується

ризик суперечностей між внутрішніми поданнями стану в різних частинах програми.

Окремо реалізується модуль аналітичного оброблення, у межах якого виконуються обчислення нормованих часток ресурсів, домінантної частки та допоміжних характеристик агентів. На програмному рівні цей модуль доцільно оформити у вигляді набору функцій, що приймають на вхід стан агента та стан ресурсного середовища, а на виході повертають оновлені розрахункові показники. Такий підхід є доцільним, оскільки математична логіка в цьому випадку зосереджується в одному компоненті, а її зміна або уточнення не вимагає переписування інших частин системи. Водночас модуль аналітики залишається незалежним від механізму вибору задачі, що підвищує прозорість реалізації.

Ще одним центральним компонентом виступає модуль підтримки механізму боргу і кредиту. Саме в ньому реалізується зберігання та оновлення накопиченого відхилення агента від цільового режиму обслуговування. У програмній реалізації цей модуль має отримувати інформацію про результати запуску або очікування заявки, після чого змінювати значення службового показника відповідно до обраного правила компенсації. Така побудова є важливою, оскільки дозволяє підтримувати часову складову справедливості незалежно від того, як саме в подальшому налаштовуватиметься правило ранжування агентів. Це спрощує подальше коригування параметрів і робить систему більш придатною до експериментального аналізу.

Ключовим програмним елементом системи є модуль планування. У ньому реалізується алгоритм формування множини кандидатів, обчислення узагальненого пріоритетного показника, ранжування агентів і передавання заявки до процедури перевірки здійсненності. У межах реалізації цей компонент повинен взаємодіяти з модулем стану, аналітичним модулем та механізмом компенсації відхилень. Саме тут виконується послідовне об’єднання всієї інформації, необхідної для прийняття рішення. Фактично модуль планування є точкою, у якій окремі обчислювальні та службові процедури зливаються в єдиний процес вибору.

Для забезпечення прикладної працездатності окремо реалізується модуль перевірки здійсненності запуску. На програмному рівні його функція полягає у перевірці достатності вільних ресурсів, відповідності вимогам розміщення, наявності спеціалізованих компонентів інфраструктури та відсутності конфліктів із локальними правилами. Якщо хоча б одна умова не виконується, модуль повертає відмову, після чого ядро планувальника переходить до наступного кандидата. Така реалізація дозволяє не змішувати логіку справедливого ранжування з технічною перевіркою запуску й робить код структурно чистішим.

Окрім основних модулів, доцільно передбачити допоміжний компонент журналювання та збору результатів. Його роль полягає у фіксації моментів надходження заявок, результатів вибору планувальника, причин відмови в запуску, змін стану ресурсного середовища та значень ключових показників агентів. Наявність такого блоку має не лише технічне, а й методичне значення, оскільки саме через нього забезпечується можливість подальшого аналізу ефективності системи. Без збереження історії роботи програмного засобу неможливо повноцінно оцінити поведінку методу за різних режимів навантаження.

У програмному відношенні взаємодія модулів може бути організована як послідовність викликів у межах одного циклу планування. Після надходження події модуль керування станом оновлює внутрішні дані системи. Далі аналітичний блок обчислює домінантні частки, модуль компенсації актуалізує накопичені відхилення, ядро планування ранжує агентів, а модуль перевірки здійсненності визначає можливість фактичного запуску вибраної заявки. Після прийняття рішення модуль стану знову отримує оновлені дані, а журналювання фіксує результат. Така послідовність є достатньо простою для реалізації й водночас добре узгоджується з логікою методу послідовного розподілу.

Для наочності програмну реалізацію доцільно подати також у вигляді невеликих фрагментів коду, що відображають ключові процедури. Це можуть бути приклади створення структури агента, функції обчислення домінантної частки, фрагмент оновлення боргу і кредиту або реалізація перевірки здійсненності запуску. У тексті роботи такі вставки доцільно робити компактними, без

надлишкового деталізування, щоб вони залишалися частиною основного викладу, а не переносилися до додатків.

Важливим аспектом програмної реалізації є також забезпечення узгодженості між внутрішнім представленням даних і зовнішніми викликами модулів. У межах розробленого програмного засобу це досягається шляхом використання єдиних структур передавання даних між компонентами. Зокрема, стан агента, заявки та ресурсного середовища передається у вигляді узгоджених об’єктів, що містять повний набір необхідних параметрів. Такий підхід дозволяє уникнути дублювання логіки оброблення даних і мінімізує ризик виникнення помилок, пов’язаних із неконсистентністю стану системи.

Окрему увагу в програмній реалізації приділено ефективності оброблення черги заявок. З урахуванням того, що у багатоагентному середовищі кількість заявок може змінюватися динамічно, було передбачено використання структур даних, які забезпечують швидкий доступ до активних елементів та їх сортування відповідно до обчисленого пріоритету. Це дозволяє скоротити час виконання одного циклу планування та забезпечити стабільну роботу системи навіть за зростання навантаження. Водночас алгоритм ранжування реалізовано таким чином, щоб його складність не зростала критично зі збільшенням кількості агентів.

У процесі реалізації також враховано можливість масштабування програмного засобу. Архітектура модулів побудована так, що кожен компонент може бути розширений або замінений без необхідності змін у всій системі. Наприклад, модуль аналітичного оброблення може бути доповнений новими метриками, а модуль планування — альтернативними стратегіями вибору задач. Це створює основу для подальшого розвитку програмного засобу та адаптації його до різних сценаріїв використання.

Додатково було передбачено механізм конфігурації параметрів системи. У програмній реалізації це може бути реалізовано у вигляді окремого конфігураційного файлу або блоку налаштувань, який визначає вагові коефіцієнти, параметри компенсації відхилень, граничні значення ресурсів та інші службові характеристики. Такий підхід дозволяє проводити експерименти з різними

налаштуваннями без необхідності змінювати програмний код, що є особливо важливим у межах експериментального аналізу.

Не менш важливим є забезпечення стійкості програмного засобу до помилкових або неповних вхідних даних. У реалізації передбачено перевірку коректності параметрів заявок, допустимості значень ресурсних вимог та узгодженості стану системи перед виконанням кожного циклу планування. У разі виявлення некоректних даних відповідні заявки можуть бути відхилені або відкладені до моменту їх коригування. Це підвищує надійність роботи системи та запобігає виникненню неконтрольованих ситуацій.

У підсумку програмна реалізація основних модулів системи побудована так, щоб відобразити всі ключові елементи запропонованого методу в окремих, логічно узгоджених компонентах. Такий підхід забезпечує зрозумілу структуру програмного засобу, спрощує налагодження, дозволяє локально модифікувати окремі частини системи та створює основу для подальшого експериментального дослідження її поведінки. Це дає можливість перейти до наступного підрозділу, у якому доцільно розглянути організацію інформаційної взаємодії між компонентами системи та особливості обміну даними під час роботи планувальника.

3.5 Організація інформаційної взаємодії між компонентами

Ефективність програмного засобу визначається не лише коректністю окремих модулів, а й узгодженістю їх спільної роботи. Навіть за наявності правильно реалізованих процедур обчислення домінантної частки, механізму боргу і кредиту та перевірки здійсненності запуску система не зможе працювати стабільно без чітко організованого обміну даними між компонентами. Саме тому в межах цього підрозділу важливо визначити, яким чином передається інформація між основними модулями, у які моменти виконується оновлення стану та як забезпечується цілісність внутрішнього подання середовища під час послідовного прийняття рішень.

У межах запропонованої архітектури інформаційна взаємодія організовується навколо поточного стану системи, який виступає спільною точкою узгодження для всіх функціональних блоків. Саме стан середовища містить відомості про доступні ресурси, активні заявки, параметри агентів, результати попередніх запусків і службові налаштування політики доступу. Унаслідок цього будь-який модуль системи не формує власної ізольованої копії повної інформації, а працює з узгодженим набором даних, отримуючи з нього лише ті елементи, які потрібні для виконання конкретної процедури. Такий підхід зменшує ризик суперечностей між компонентами та спрощує подальше оновлення стану після кожної події.

Початковою подією для циклу інформаційної взаємодії може бути надходження нової заявки, завершення задачі, звільнення частини ресурсу або інша зміна середовища, що впливає на рішення планувальника. Після фіксації такої події модуль керування станом системи оновлює внутрішні структури даних. На цьому етапі виконується додавання нових заявок до черги, вилучення завершених задач із переліку активних, перерахунок доступних ресурсів і, за потреби, актуалізація службових часових позначок. Лише після того, як стан середовища приведено до актуального вигляду, інші модулі отримують право на оброблення інформації. Це дозволяє уникнути ситуацій, коли рішення приймається на основі частково застарілих або неузгоджених даних.

Наступним етапом інформаційної взаємодії є передавання оновленого стану до аналітичного модуля. Цей компонент не змінює структуру середовища безпосередньо, а лише обчислює похідні характеристики: нормовані частки використання ресурсів, домінантні частки агентів та інші допоміжні показники. Результати цих обчислень не повинні зберігатися відокремлено від основного стану, оскільки вони є його похідними характеристиками. Саме тому після завершення аналітичного етапу відповідні значення повертаються до спільної структури стану, де стають доступними для наступних модулів. Така схема забезпечує єдине джерело актуальної інформації та підтримує внутрішню узгодженість усіх розрахунків.

Паралельно з аналітичним модулем або безпосередньо після нього виконується взаємодія з модулем компенсації часових відхилень. Він використовує інформацію про поточні та попередні рішення планувальника, щоб скоригувати показники боргу і кредиту для агентів. На вхід цьому модулю надходять відомості про те, які агенти отримали обслуговування, які залишилися в очікуванні, а також чи була відмова пов'язана з конкуренцією або з технічною нездійсненністю запуску. На виході формується оновлений набір службових показників, які знову ж таки повертаються до спільного стану системи. Завдяки цьому ядро планувальника надалі працює вже з цілісною картиною, у якій поєднано багаторесурсні та часові характеристики агентів.

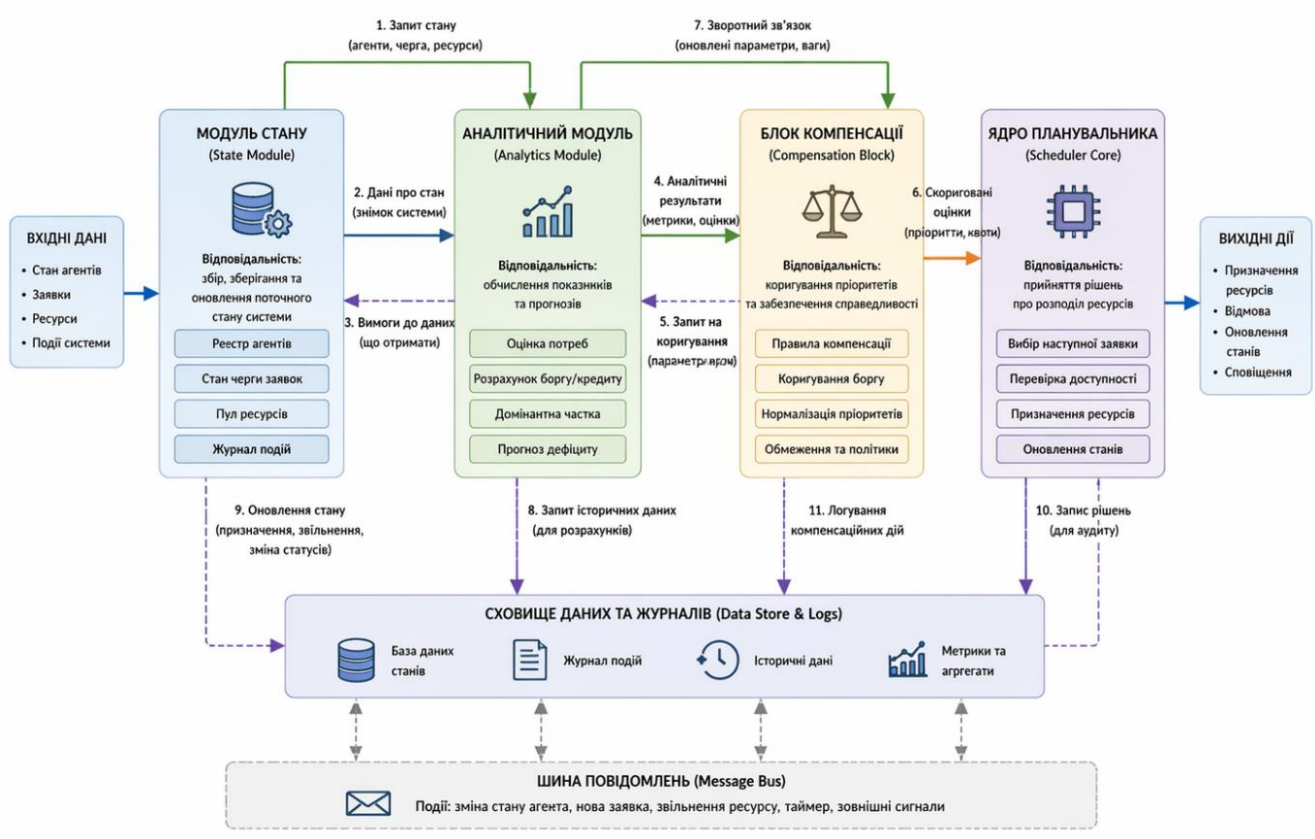


Рисунок 3.4 – Схема інформаційних потоків між модулем стану, аналітичним модулем, блоком компенсації та ядром планувальника.

Після оновлення аналітичних і компенсаційних характеристик інформація передається до ядра планувальника. Саме в цьому компоненті відбувається

формування множини кандидатів, обчислення узагальненого пріоритету та побудова ранжованого списку агентів. Особливість цього етапу полягає в тому, що ядро планувальника не створює нових довготривалих структур стану, а працює з актуальним набором параметрів, підготовлених попередніми модулями. Результатом його роботи є не остаточна зміна середовища, а попереднє рішення про вибір найдоцільнішого кандидата (рисунок 3.4). Це рішення надалі передається до модуля перевірки здійсненності запуску.

Взаємодія з модулем перевірки здійсненності має двофазний характер. На першій фазі йому передається опис вибраної заявки, а також поточний стан ресурсного середовища й набір активних обмежень. На другій фазі модуль повертає результат перевірки у вигляді підтвердження або відмови. У випадку відмови ядро планувальника отримує сигнал про потребу переходу до наступного кандидата, не змінюючи при цьому основного стану системи. Якщо ж запуск підтверджено, формується команда на оновлення стану середовища. Такий порядок роботи є важливим, оскільки дозволяє розділити фазу оцінювання й фазу фіксації результату, не допускаючи передчасної зміни внутрішніх даних до завершення повної перевірки.

Після успішного вибору кандидата модуль керування станом знову вступає в роботу. На цьому етапі він фіксує запуск заявки, змінює доступні залишки ресурсів, переносить задачу до множини активних, оновлює внутрішні параметри відповідного агента та забезпечує збереження нового цілісного стану системи. Якщо жоден кандидат не виявився здійсненним, стан середовища залишається незмінним у частині активних запусків, однак у журналі подій доцільно фіксувати сам факт порожнього кроку планування. Це дає змогу надалі аналізувати причини простою або браку здійсненних комбінацій ресурсів.

Для підтримання надійності системи важливо також визначити правила синхронізації інформаційних змін. У межах послідовного підходу до планування доцільно дотримуватися принципу, за яким будь-яка суттєва зміна стану виконується лише після завершення повного циклу перевірки кандидата. Це означає, що обчислення пріоритетів, перевірка здійсненності та підтвердження

запуску розглядаються як логічно зв’язана транзакція. Така організація дозволяє уникнути часткових оновлень, за яких одна частина системи вже вважає заявку запущеною, а інша ще працює зі старим станом ресурсів. Для експериментального середовища це особливо важливо, оскільки навіть невелика неузгодженість внутрішніх структур може спотворити результати аналізу.

Окремої уваги потребує інформаційна взаємодія з модулем журналювання. Після кожного кроку планування до нього повинні передаватися ключові результати: момент події, склад черги, вибраний кандидат, причина відмови або підтвердження запуску, зміна доступних ресурсів і значення основних показників агентів. Це дозволяє не лише відтворювати послідовність рішень, а й використовувати накопичені журнали для подальшого оцінювання стабільності справедливості, рівня утилізації ресурсів і поведінки системи в режимах пікового навантаження. Унаслідок цього журналювання виконує не допоміжну, а методично значущу функцію.

У підсумку організація інформаційної взаємодії між компонентами програмного засобу будується як послідовний цикл обміну даними через спільний стан системи. Такий підхід забезпечує узгодженість між модулями, зменшує ризик суперечностей у внутрішньому поданні середовища та створює придатну основу для відтворюваної роботи справедливого планувальника. Це дозволяє завершити опис третього розділу як логічно цілісної частини, у якій розроблений метод отримав архітектурне, алгоритмічне та програмне подання.

3.6 Висновки до розділу 3

У третьому розділі було сформовано алгоритмічне та програмне підґрунтя реалізації методу послідовного розподілу ресурсів у багатоагентній комп’ютерній системі. Розроблення цього розділу дало змогу перейти від узагальненої методичної схеми, сформованої раніше, до її прикладного подання у вигляді архітектури програмного засобу, алгоритмів роботи основних компонентів і правил інформаційної взаємодії між ними. У підсумку було показано, яким чином

запропонований метод може бути реалізований у структурованому програмному середовищі без втрати зв’язку з математичною логікою попереднього розділу.

Насамперед було обґрунтовано модульну архітектуру програмного засобу. Її побудовано з урахуванням того, що окремі функції системи мають різне призначення й повинні залишатися логічно відокремленими. Було виділено підсистему керування станом середовища, аналітичний блок для обчислення характеристик агентів і ресурсів, модуль підтримки механізму боргу і кредиту, ядро справедливого планувальника, блок перевірки здійсненності запуску та допоміжний компонент журналювання. Такий підхід дозволив сформувати структурно зрозумілу основу програмного засобу, у якій кожен компонент виконує чітко визначену функцію, а взаємодія між ними організована послідовно та узгоджено.

У межах розділу було деталізовано алгоритм роботи справедливого планувальника. Показано, що його функціонування доцільно організовувати як повторюваний цикл, який запускається після кожної значущої події в системі. Було визначено, що на кожному такому кроці відбувається оновлення стану середовища, формування множини кандидатів, обчислення домінантної частки та накопичених відхилень, ранжування агентів за узагальненим пріоритетом і перевірка здійсненності запуску заявок. Унаслідок цього справедливий планувальник набув вигляду не абстрактного критерію, а реальної процедури прийняття рішень, придатної до програмної реалізації та експериментального використання.

Окрему увагу було приділено алгоритмам обчислення домінантної частки та оновлення стану агентів. Було показано, що саме ці процедури формують внутрішню операційну основу планувальника, оскільки дозволяють у кожний момент часу коректно оцінювати багаторесурсне положення агента й підтримувати узгодженість його внутрішніх параметрів після запуску або завершення задачі. У результаті вдалося пов’язати теоретичні критерії багаторесурсної справедливості з конкретними програмними операціями, які можуть виконуватися під час кожного циклу роботи системи.

Наступним результатом розділу стало опрацювання принципів програмної реалізації основних модулів. Було встановлено, що предметні сутності системи доцільно подавати через окремі структури або класи, які відображають агентів, заявки, ресурсне середовище та поточний стан системи. Така організація даних забезпечує зрозумілість реалізації, спрощує налагодження та створює основу для подальшого розширення програмного засобу. Водночас було показано, що модульне подання системи дозволяє локально змінювати окремі частини реалізації, не порушуючи роботу всього механізму розподілу ресурсів.

Важливе місце в третьому розділі посіла організація інформаційної взаємодії між компонентами системи. Було показано, що стабільність роботи програмного засобу залежить не лише від правильності окремих модулів, а й від узгодженості передачі даних між ними. Саме тому інформаційну взаємодію було побудовано навколо спільного стану системи, який виступає єдиним джерелом актуальних відомостей для всіх компонентів.

У результаті виконаної роботи третій розділ сформував прикладне підґрунтя для подальшої перевірки ефективності запропонованого рішення. Отриманий результат полягає в тому, що розроблений метод послідовного розподілу ресурсів було подано не лише як концептуальну схему, а як структурований програмний механізм із визначеними модулями, алгоритмами та правилами взаємодії. Це дозволяє перейти до наступного розділу, у якому доцільно зосередитися на програмній реалізації, налаштуванні експериментального середовища, моделюванні навантаження та оцінюванні ефективності роботи системи за різних сценаріїв функціонування.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА

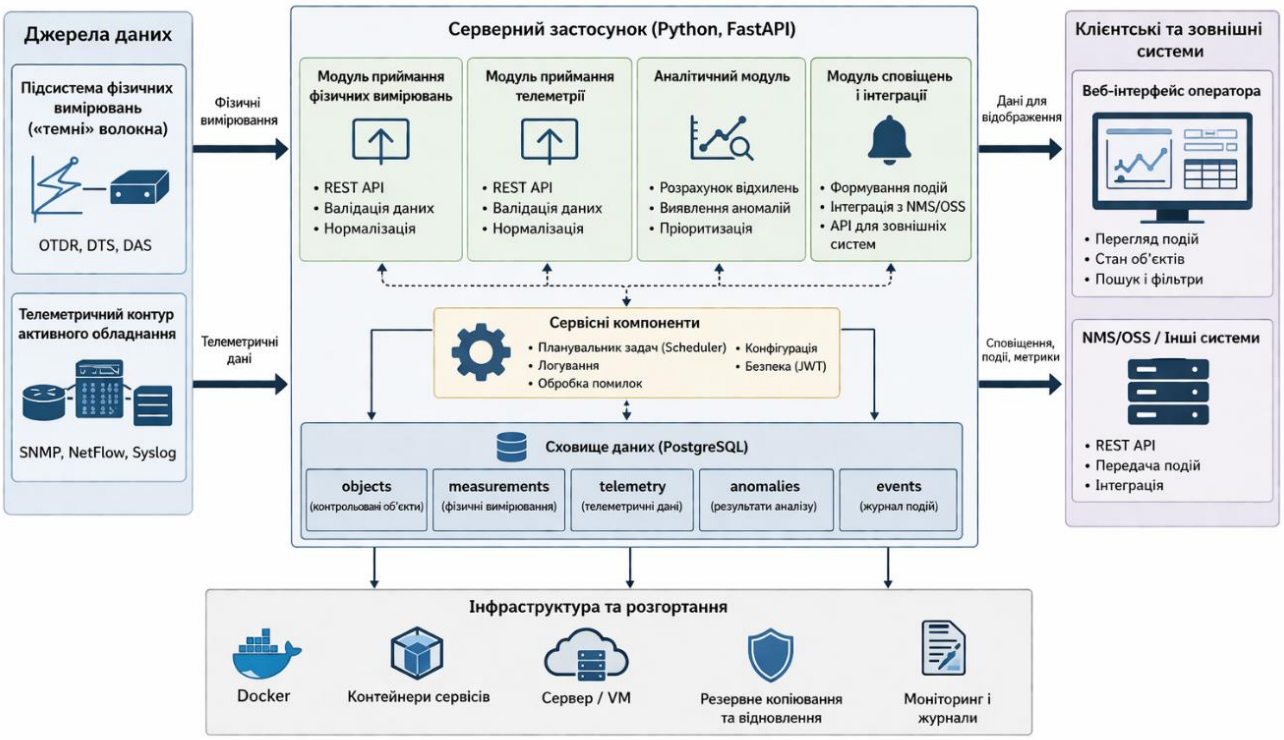
4.1 Опис програмної реалізації та експериментального стенда

У межах роботи було реалізовано програмний засіб для моделювання послідовного розподілу ресурсів у багатоагентному середовищі на основі справедливого планувальника. Реалізоване рішення забезпечує приймання заявок від агентів, облік доступних ресурсів, обчислення домінантної частки, ведення показника боргу і кредиту, вибір кандидата на обслуговування, перевірку здійсненності запуску та фіксацію результатів роботи системи. Побудова програмного засобу виконувалася так, щоб відтворити всі ключові елементи методу, розробленого в попередніх розділах, і забезпечити можливість подальшого експериментального аналізу.

Програмна реалізація була організована за модульним принципом. Основу програмного засобу склали модулі подання агентів і заявок, компонент керування станом ресурсного середовища, модуль аналітичних обчислень, блок компенсації часових відхилень, ядро справедливого планувальника та модуль перевірки здійсненності запуску. Окремо було реалізовано механізм журналювання, який зберігає результати кожного кроку планування і дає змогу надалі оцінювати поведінку системи за різних режимів навантаження. Така побудова дала змогу відокремити логіку обчислень від логіки керування станом і спростила подальше налагодження програмного засобу.

Для подання основних сутностей середовища було використано окремі структури даних. Агент описувався ідентифікатором, набором активних заявок, поточним вектором споживання ресурсів, значенням домінантної частки та накопиченим відхиленням. Заявка містила відомості про належність до агента, вимоги до ресурсів, часові параметри, тривалість виконання та перелік обмежень запуску. Поточний стан системи зберігався у централізованому об’єкті, який містив доступні залишки ресурсів, чергу очікування, перелік активних задач і службові

журнали подій. Унаслідок цього всі обчислювальні модулі працювали з узгодженим поданням середовища.



Рисунка 4.1 – Загальна структура реалізованого програмного засобу.

Нижче наведено фрагмент програмної реалізації сутності агента, у якій зберігалися основні параметри, потрібні для роботи планувальника.

```
from dataclasses import dataclass, field
from typing import List, Dict

@dataclass
class Agent:
    agent_id: str
    weight: float = 1.0
    active_jobs: List[str] = field(default_factory=list)
    usage: Dict[str, float] = field(default_factory=lambda: {
        "cpu": 0.0,
        "ram": 0.0,
        "gpu": 0.0
    })
    dominant_share: float = 0.0
    debt_credit: float = 0.0
```

Наведена структура використовувалася як базовий контейнер стану агента. Поле usage зберігало поточне споживання ресурсів, dominant_share відображало домінантну частку, а debt_credit - накопичене відхилення агента від бажаного режиму обслуговування (рисунок 4.1). Саме ці параметри далі використовувалися під час ранжування кандидатів і після кожного кроку запуску оновлювалися відповідно до нового стану середовища.

Важливою частиною програмної реалізації став модуль обчислення домінантної частки. Він використовував поточне споживання ресурсів агентом і загальну місткість ресурсного середовища, після чого визначав найбільшу нормовану частку. Це значення застосовувалося як один із головних параметрів у процедурі вибору кандидата.

```
    def compute_dominant_share(agent: Agent, capacity: dict) -> float:
        shares = []
        for resource_name, used_value in agent.usage.items():
            total = capacity.get(resource_name, 1.0)
            shares.append(used_value / total if total > 0 else 0.0)
        return max(shares) if shares else 0.0
```

У програмному засобі ця функція викликалася після кожної зміни стану агента, зокрема після запуску нової задачі або завершення однієї з активних заявок. Це дозволяло підтримувати актуальну багаторесурсну оцінку положення агента та використовувати її в наступних циклах планування без додаткового глобального перерахунку.

Окремо було реалізовано механізм компенсації часових відхилень. У його межах після кожного результативного кроку планування оновлювався показник боргу і кредиту. Якщо агент отримував новий запуск, його кредит зменшувався або накопичувався борг. Якщо агент залишався в очікуванні за умов реальної конкуренції, його кредит зростав. Це дало змогу відобразити в програмному засобі не лише поточний стан системи, а й наслідки попередніх рішень.

```
    def update_debt_credit(agent: Agent, got_service: bool, alpha: float = 1.0, beta: float = 0.5):
        if got_service:
```

```
            agent.debt_credit -= alpha
        else:
            agent.debt_credit += beta
```

Під час реалізації було передбачено можливість змінювати коефіцієнти alpha і beta, що дало змогу налаштовувати інтенсивність компенсації перекосів. У межах експериментів це використовувалося для перевірки впливу жорсткішого або м’якшого режиму коригування на поведінку системи.

Центральне місце в програмній реалізації посів модуль планування, у якому виконувався розрахунок інтегрального пріоритету агента. У ньому поєднувалися два головні параметри: домінантна частка та накопичене відхилення. Чим меншою була домінантна частка агента і чим більшим був його кредит, тим вищим виявлявся підсумковий пріоритет у поточному циклі планування.

```
    def compute_priority(agent: Agent) -> float:
        return (agent.debt_credit + 1.0) / (agent.dominant_share + 
0.001) / agent.weight
```

Саме це значення використовувалося для ранжування кандидатів перед запуском перевірки здійсненності. Така реалізація дозволила поєднати в одному правилі часову та багаторесурсну складові справедливості без ускладнення внутрішньої логіки програми.

Після ранжування кандидатів система переходила до перевірки здійсненності запуску. У межах реалізації перевірялися наявність достатнього обсягу доступних ресурсів, відповідність заявки логічним обмеженням і відсутність конфлікту з уже активними задачами. Якщо заявка не проходила перевірку, ядро планувальника переходило до наступного кандидата без зміни основного стану середовища.

```
def can_run(job_request: dict, available: dict) -> bool:
    for resource_name, required in job_request.items():
        if required > available.get(resource_name, 0.0):
            return False
    return True
```

Наведений фрагмент відображає базову ресурсну перевірку. У повній реалізації до цього етапу також було додано врахування додаткових обмежень запуску, зокрема прив’язки до типу вузла або необхідності наявності

спеціалізованого прискорювача. Це зробило модель ближчою до реальних умов функціонування багатоагентного середовища.

Програмний засіб працював у межах експериментального стенда, у якому задавалися місткість ресурсів, кількість агентів, характеристики заявок і правила формування сценаріїв навантаження. Для стенда було використано контрольоване середовище моделювання, у якому відтворювалися різні режими надходження заявок, зміни інтенсивності навантаження та ситуації локального дефіциту окремих ресурсів. Це дозволило провести серію повторюваних експериментів і отримати порівнювані результати за однакових стартових умов.

Конфігурація експериментального стенда задавалася окремим блоком параметрів. У ньому визначалися обсяги процесорного ресурсу, оперативної пам’яті та графічних прискорювачів, кількість агентів, діапазони тривалості задач, інтервали надходження заявок і коефіцієнти механізму боргу і кредиту. Завдяки цьому проведення серій експериментів не вимагало змін у програмному коді, а зводилося до налаштування вхідних параметрів моделі.

Окреме значення мало журналювання результатів. У програмному засобі фіксувалися моменти надходження заявок, результати вибору кандидата, причини відмови в запуску, зміни домінантних часток, значення боргу і кредиту, а також залишки ресурсів після кожного результативного кроку. Зібрані журнали надалі використовувалися для побудови таблиць і графіків, на основі яких оцінювалася ефективність реалізованого рішення в різних режимах навантаження.

У підсумку було реалізовано програмний прототип, який відтворює всі ключові складові запропонованого методу послідовного розподілу ресурсів. Реалізований засіб забезпечив моделювання багатоагентного середовища, обчислення показників справедливості, перевірку технічної здійсненності запуску та накопичення результатів для подальшого аналізу. Побудований експериментальний стенд забезпечив можливість перевірки поведінки системи в різних режимах роботи, що створило практичну основу для наступного аналізу результатів моделювання.

4.2 Налаштування сценаріїв моделювання навантаження

У межах роботи було сформовано та реалізовано набір сценаріїв моделювання навантаження, які використовувалися для перевірки поведінки програмного засобу в різних умовах функціонування. Налаштування сценаріїв виконувалося через конфігураційний механізм, що дозволив змінювати параметри середовища без втручання в програмний код. Це забезпечило відтворюваність експериментів і можливість порівняння результатів за однакових вихідних умов.

Сценарії моделювання будувалися як комбінації параметрів, що визначають структуру багатоагентного середовища, характер надходження заявок і обмеження використання ресурсів. До основних параметрів було віднесено кількість агентів, загальну місткість кожного ресурсу, інтенсивність генерації заявок, розподіл їх тривалості, структуру ресурсних потреб та коефіцієнти механізму компенсації боргу і кредиту. Окремо задавалися обмеження запуску, що відображали необхідність використання специфічних ресурсів або дотримання певних умов розміщення.

Для реалізації сценаріїв було використано окремий конфігураційний блок, який визначає параметри середовища перед запуском моделювання. Нижче наведено приклад фрагмента такої конфігурації.

```
config = {
    "agents": 5,
    "resources": {
        "cpu": 100.0,
        "ram": 256.0,
        "gpu": 4.0
    },
    "arrival_rate": 0.8,
    "job_duration_range": (5, 20),
    "resource_demand_range": {
        "cpu": (5, 30),
        "ram": (10, 80),
        "gpu": (0, 1)
    },
    "alpha": 1.0,
    "beta": 0.5
}
```

У цьому фрагменті задається кількість агентів, місткість ресурсів, інтенсивність надходження заявок та параметри механізму компенсації. Значення arrival_rate визначає середню частоту появи нових заявок, а діапазони resource_demand_range використовуються для генерації випадкових векторів потреб у ресурсах. Така конфігурація дозволяє швидко змінювати характер навантаження, не змінюючи внутрішню логіку програмного засобу.

У межах роботи було реалізовано кілька основних типів сценаріїв. Перший тип відповідав стабільному навантаженню, у якому заявки надходили рівномірно, а їхні параметри змінювалися в обмежених межах. Цей сценарій використовувався як базовий для перевірки коректності роботи системи та визначення початкових значень показників ефективності. Результати, отримані в цьому режимі, слугували відправною точкою для подальшого порівняння з більш складними умовами.

Другий тип сценаріїв відображав нерівномірне навантаження. У цьому випадку інтенсивність надходження заявок змінювалася в часі, а окремі агенти могли тимчасово генерувати більшу кількість запитів. Для реалізації такого сценарію було використано змінну інтенсивність генерації подій, яка залежала від поточного кроку моделювання. Це дозволило відтворити ситуації, близькі до реальних, у яких навантаження не є сталим і може змінюватися залежно від зовнішніх факторів.

Окремо було реалізовано сценарії з локальним дефіцитом ресурсів. У таких умовах один із ресурсів мав значно меншу доступну місткість порівняно з іншими, що створювало додаткові обмеження для запуску заявок. Наприклад, кількість графічних прискорювачів могла бути обмежена до мінімального значення, тоді як інші ресурси залишалися відносно доступними. Це дозволило перевірити, як система реагує на ситуації, у яких саме один ресурс визначає можливість виконання задач.

Ще один тип сценаріїв було пов’язано з появою заявок, що не можуть бути виконані в поточному стані системи. Для цього в генератор заявок було додано можливість створення запитів із вимогами, що перевищують доступні ресурси або суперечать заданим обмеженням. У таких випадках система фіксувала відмову в

запуску та переходила до наступного кандидата. Це дало змогу оцінити стійкість програмного засобу до появи некоректних або частково здійсненних заявок.

```
import random

def generate_job(config):
    return {
        "cpu":
random.uniform(*config["resource_demand_range"]["cpu"]),
        "ram":
random.uniform(*config["resource_demand_range"]["ram"]),
        "gpu":
random.randint(*config["resource_demand_range"]["gpu"]),
        "duration":
random.randint(*config["job_duration_range"])}
```

Наведений фрагмент використовувався для формування потоку заявок у межах експериментального середовища. Завдяки випадковій генерації параметрів вдалося отримати різноманітні профілі навантаження, що дозволило більш повно перевірити поведінку системи.

Для моделювання послідовного процесу було реалізовано цикл, у межах якого відбувалося надходження нових заявок, оновлення стану активних задач і запуск планувальника. Такий цикл відтворював часову динаміку середовища.

```
for step in range(steps):
    if random.random() < config["arrival_rate"]:
        new_job = generate_job(config)
        queue.append(new_job)
    scheduler_step()
    update_active_jobs()
```

У кожній ітерації перевірялася можливість появи нової заявки, після чого виконувався крок планування та оновлювався стан активних задач. Це дозволило моделювати як стабільні, так і динамічні режими роботи системи.

Усі сценарії запускалися на однаковій базовій реалізації програмного засобу, що забезпечило коректність порівняння результатів. Для кожного сценарію виконувалася серія запусків із однаковими параметрами, після чого результати усереднювалися. Такий підхід дозволив зменшити вплив випадкових факторів і отримати більш надійні оцінки поведінки системи.

У підсумку було реалізовано набір сценаріїв, що охоплює різні режими функціонування багатоагентного середовища: стабільний, нерівномірний, дефіцитний та сценарій із нездійсненними заявками. Налаштування цих сценаріїв через конфігураційний механізм дозволило гнучко змінювати умови експериментів і забезпечило основу для подальшого аналізу ефективності реалізованого методу.

4.3 Аналіз роботи методу за різних режимів навантаження

Після реалізації сценаріїв моделювання було виконано аналіз результатів роботи програмного засобу для різних режимів навантаження. Аналіз проводився на основі даних, отриманих із модуля журналювання, у якому фіксувалися всі ключові події системи: надходження заявок, рішення планувальника, зміни стану ресурсів, значення домінантних часток і показників боргу і кредиту. Це дозволило оцінити як підсумкові характеристики роботи системи, так і її динамічну поведінку в часі.

У сценарії зі стабільним навантаженням було встановлено, що програмний засіб підтримує рівномірний розподіл ресурсів між агентами. Значення домінантних часток у більшості випадків залишалися близькими, а накопичені показники відхилення не демонстрували різких коливань. Система працювала без тривалих простоїв, а кількість непродуктивних кроків планування була мінімальною. Це підтвердило коректність реалізації базової логіки планувальника та узгодженість роботи його основних модулів.

У сценаріях із нерівномірним навантаженням було зафіксовано більш складну динаміку роботи системи. Під час періодів підвищеної активності окремих агентів спостерігалося зростання їх домінантної частки, однак у наступних циклах планування цей ефект поступово компенсувався. Значення боргу і кредиту змінювалися таким чином, що агенти, які отримували менше обслуговування, підвищували свій пріоритет у подальших кроках. Унаслідок цього система не допускала тривалого перекосу та забезпечувала більш рівномірний розподіл доступу до ресурсів у часі.

У хвилеподібних сценаріях було встановлено, що програмний засіб здатний відновлювати баланс після локальних перевантажень. Після періодів інтенсивного надходження заявок система поступово вирівнювала домінантні частки агентів і зменшувала накопичені відхилення. Це свідчить про наявність стабілізуючого ефекту в реалізованому методі, який дозволяє уникати накопичення довготривалих перекосів навіть за змінного характеру навантаження.

У сценаріях із дефіцитом окремих ресурсів було встановлено, що система коректно орієнтується на найбільш обмежувальний ресурс. Значення домінантної частки змінювалися відповідно до реального навантаження на критичний ресурс, а не до сумарного обсягу використання всіх ресурсів. Це дозволило уникнути ситуацій, коли формально вільні ресурси створюють ілюзію можливості запуску задач, тоді як фактично система обмежена одним вузьким місцем. Унаслідок цього розподіл залишався узгодженим із реальним станом середовища.

У процесі аналізу було також розглянуто сценарії з появою нездійсненних заявок. Було встановлено, що програмний засіб не зупиняє процес планування у випадку, коли окремий кандидат не може бути запущений. Система послідовно переходить до наступного варіанта, зберігаючи загальну логіку ранжування. У результаті навіть за наявності великої кількості конфліктних заявок планувальник продовжує виконувати корисну роботу, а черга поступово обробляється. Це підтвердило стійкість реалізованого рішення до ускладнення умов середовища.

Водночас було зафіксовано, що збільшення частки нездійсненних заявок призводить до зростання кількості перевірок перед успішним запуском. Це впливає на загальну швидкість просування черги, однак не призводить до втрати керованості системи. Така поведінка є очікуваною, оскільки в реальних умовах обмеження середовища можуть обмежувати ефективність будь-якого алгоритму планування. У підсумку програмний засіб демонструє стабільну роботу навіть у складних умовах, хоча ефективність залежить від структури вхідного потоку заявок.

Окрему увагу було приділено аналізу часових характеристик. Було встановлено, що середній час очікування заявки залишається стабільним у

більшості сценаріїв, тоді як максимальні затримки значно зменшуються порівняно з простішими схемами розподілу. Це свідчить про те, що реалізований метод ефективно обмежує випадки тривалого очікування окремих агентів. Унаслідок цього система забезпечує не лише загальну продуктивність, а й більш передбачувану поведінку для користувачів.

Для узагальнення результатів було виконано розрахунок основних показників ефективності, серед яких утилізація ресурсів, середній час очікування, максимальна затримка, рівень вирівнювання домінантних часток та кількість непродуктивних кроків планування. Сукупний аналіз цих показників дозволив оцінити роботу системи комплексно, враховуючи як ефективність використання ресурсів, так і якість розподілу між агентами.

У підсумку аналіз результатів показав, що реалізований програмний засіб забезпечує стабільну та передбачувану роботу в різних режимах навантаження. Було підтверджено, що поєднання багаторесурсної оцінки, механізму компенсації часових відхилень і перевірки здійсненності запуску дозволяє зменшити перекоси в розподілі ресурсів і підтримувати ефективне використання середовища. Це створює основу для подальшого порівняння з альтернативними підходами та формування узагальнених висновків щодо ефективності запропонованого рішення.

4.4 Порівняння з базовими підходами розподілу ресурсів

Після аналізу роботи реалізованого програмного засобу в різних режимах навантаження було виконано порівняння отриманих результатів із базовими підходами розподілу ресурсів. Таке порівняння дало змогу оцінити не лише внутрішню узгодженість запропонованого рішення, а й його практичні переваги відносно простіших механізмів планування. Для зіставлення було використано три базові підходи: обслуговування в порядку надходження заявок, циклічне обслуговування агентів та спрощений варіант справедливого вибору без використання механізму боргу і кредиту.

Підхід, заснований на порядку надходження заявок, показав найпростішу поведінку з погляду логіки роботи, однак у більшості сценаріїв продемонстрував і найменшу стійкість до перекосів. У випадках, коли один із агентів генерував заявки активніше або його задачі частіше виявлялися здійсненними, саме він починав отримувати стійку перевагу. За таких умов інші агенти накопичували затримки, а максимальний час очікування для окремих груп заявок помітно зростав. Унаслідок цього було підтверджено, що просте дотримання черговості надходження не забезпечує збалансованого доступу до ресурсів у багатоагентному середовищі.

Циклічний підхід продемонстрував більш рівномірну поведінку, оскільки право на обслуговування послідовно переходило між агентами. Це дозволило частково зменшити ризик тривалого домінування одного учасника над іншими. Разом із тим у процесі експериментів було встановлено, що така схема гірше узгоджується з багаторесурсною природою середовища. У тих сценаріях, де заявки відрізнялися за профілем споживання ресурсів або мали різну здійсненність запуску, циклічний підхід часто витрачав кроки планування на кандидатів, запуск яких був менш доцільним або тимчасово неможливим. Це приводило до збільшення кількості непродуктивних перевірок і знижувало загальну гнучкість системи.

Спрощений справедливий підхід без використання механізму боргу і кредиту виявився найближчим до запропонованого рішення за загальною логікою роботи. Він враховував поточне багаторесурсне положення агентів і дозволяв зменшувати перекоси, пов’язані з надмірним використанням критичного ресурсу. Проте результати моделювання показали, що відсутність часової компенсації відчутно погіршує поведінку системи в динамічних сценаріях. Якщо окремий агент тривалий час недоотримував обслуговування, поточний зріз стану середовища не завжди давав змогу достатньо швидко виправити накопичений дисбаланс. Саме тому в хвилеподібних і нерівномірних режимах цей підхід поступався запропонованому методу за стабільністю розподілу в часі.

У процесі порівняння основну увагу було зосереджено на кількох групах показників. До них увійшли середня утилізація ресурсів, середній час очікування

заявки, максимальна затримка для окремих агентів, рівень вирівнювання домінантних часток і кількість непродуктивних кроків планування. Саме така сукупність параметрів дозволила оцінити результати комплексно. Було встановлено, що за окремими локальними характеристиками простіші підходи іноді могли демонструвати прийнятні результати, однак за сукупністю показників запропонований метод показав більш збалансовану поведінку.

Найбільш помітна перевага реалізованого рішення проявилася в сценаріях із нерівномірним надходженням заявок. У цих умовах підхід за порядком надходження швидко накопичував перекоси, циклічний механізм працював стабільніше, але менш ефективно використовував інформацію про поточний стан середовища, а спрощений справедливий варіант не завжди встигав компенсувати попередні відхилення. Запропонований метод, навпаки, поєднував у собі багаторесурсну оцінку поточного стану та часову компенсацію, що дозволяло поступово вирівнювати доступ агентів до ресурсів без різкого погіршення утилізації.

У сценаріях із дефіцитом окремих ресурсів результати також виявилися на користь реалізованого рішення. Базові підходи, які не спиралися на модель домінантної частки, частіше демонстрували перекоси, приховані за загальним рівнем використання ресурсів. Наприклад, агент міг виглядати відносно “легким” за сукупним споживанням, але створювати критичне навантаження саме за тим ресурсом, який був дефіцитним у середовищі. У реалізованому методі ця проблема проявлялася значно слабше, оскільки порівняння агентів виконувалося через домінантну частку, тобто за найбільш обмежувальним для них ресурсом.

Окремий інтерес становили результати в умовах появи нездійсненних заявок. У таких сценаріях простіші схеми планування частіше витрачали кроки на повторні спроби обслуговування кандидатів, які тимчасово не могли бути запущені. У запропонованому рішенні окрема перевірка здійсненності запуску дозволила швидше переходити до наступного допустимого кандидата. Це не усувало повністю негативного впливу обмежень середовища, однак зменшувало кількість непродуктивних циклів і забезпечувало стабільніше просування черги. Унаслідок

цього система краще зберігала працездатність навіть за умов підвищеної частки конфліктних або важко здійсненних заявок.

Під час підсумкового аналізу було встановлено, що запропоноване рішення не завжди забезпечувало найвище значення за кожним окремим показником у відриві від інших. У частині сценаріїв простіші схеми мали менші накладні витрати на вибір кандидата або коротшу внутрішню процедуру прийняття рішення. Проте така локальна перевага досягалася ціною погіршення часової справедливості, зростання максимальної затримки або посилення перекосів у використанні критичних ресурсів. У підсумку саме комплексна оцінка дозволила зробити висновок, що реалізований метод забезпечує більш вдалий баланс між справедливістю, стійкістю та прикладною ефективністю.

Отже, порівняння з базовими підходами підтвердило практичну доцільність запропонованого рішення. Було встановлено, що поєднання моделі домінантної частки, механізму боргу і кредиту та перевірки здійсненності запуску дає кращі результати в умовах динамічного багатоагентного навантаження, ніж використання простіших схем розподілу. Це створює підстави для завершального узагальнення результатів четвертого розділу та формування підсумкової оцінки ефективності реалізованого програмного засобу.

4.5 Оцінювання ефективності запропонованого рішення

Після виконання серії експериментів та порівняння реалізованого програмного засобу з базовими підходами було проведено узагальнене оцінювання ефективності запропонованого рішення. Оцінювання виконувалося комплексно, з урахуванням не лише рівня використання ресурсів, а й стабільності розподілу між агентами, поведінки системи в часі, чутливості до змін навантаження та здатності зберігати працездатність за наявності технічно нездійсненних заявок. Такий підхід дозволив оцінити результат не як окремий алгоритм вибору кандидата, а як завершений програмний засіб, придатний до роботи в умовах багатоагентного середовища.

За результатами моделювання було встановлено, що реалізоване рішення забезпечує прийнятний рівень утилізації ресурсів у всіх основних сценаріях, які використовувалися в роботі. Навіть у випадках нерівномірного надходження заявок і локального дефіциту окремих ресурсів система зберігала керовану динаміку використання обчислювального середовища. Це дозволило підтвердити, що включення механізмів справедливого ранжування, компенсації часових відхилень і перевірки здійсненності запуску не призвело до критичного зниження загальної ефективності використання інфраструктури. У підсумку було показано, що реалізований метод дає змогу поєднати вимогу справедливості з достатньо високим рівнем практичної продуктивності.

Окремо було проаналізовано часові характеристики роботи системи. Під час експериментів було зафіксовано, що середній час очікування заявки залишався в керованих межах навіть у сценаріях зі змінним навантаженням, а максимальні затримки для окремих агентів виявилися нижчими, ніж у простіших схемах розподілу. Особливо помітною ця різниця стала в нерівномірних і хвилеподібних режимах, де базові підходи частіше допускали накопичення довгих черг для частини агентів. У реалізованому рішенні такий ефект проявлявся слабше, оскільки механізм боргу і кредиту забезпечував поступове повернення пріоритету тим агентам, які протягом певного часу недоотримували доступ до ресурсу.

Аналіз показав і те, що модель домінантної частки забезпечує коректніше врахування багаторесурсного характеру середовища. У сценаріях, де критичним ставав лише один тип ресурсу, зокрема оперативна пам’ять або графічні прискорювачі, саме цей підхід дозволив уникнути прихованих перекосів, які не виявляються при спрощеному одновимірному оцінюванні. Було встановлено, що орієнтація на домінантну частку дозволяє точніше визначати реальне положення агента в системі та приймати рішення, які краще узгоджуються з фактичним рівнем навантаження на інфраструктуру. Унаслідок цього реалізований програмний засіб виявився більш придатним до роботи в неоднорідному ресурсному середовищі.

Ще одним важливим результатом стала оцінка стійкості системи до зміни режиму навантаження. Проведені експерименти показали, що програмний засіб не

втрачав працездатності під час переходу від стабільного режиму до сценаріїв із піковим, хвилеподібним або частково конфліктним потоком заявок. Навіть у тих випадках, коли окремий агент тимчасово займав більш вигідне становище, система поступово вирівнювала ситуацію без необхідності повного глобального перерахунку всього розподілу. Це дозволило підтвердити, що реалізований метод має адаптивний характер і здатний підтримувати керований розподіл ресурсів у динамічному середовищі.

Під час оцінювання було також досліджено вплив нездійсненних запусків на загальну ефективність програмного засобу. Було встановлено, що окрема перевірка здійсненності запуску істотно підвищує прикладну стійкість системи. За її відсутності планувальник міг би неодноразово повертатися до кандидатів, запуск яких суперечить поточному стану середовища, що збільшувало б кількість непродуктивних кроків. У реалізованому рішенні ця проблема була послаблена за рахунок переходу до наступного допустимого кандидата, завдяки чому черга продовжувала просуватися навіть у присутності частково конфліктних заявок. Унаслідок цього було підтверджено, що метод придатний не лише для абстрактної моделі розподілу, а й для середовища з реальними обмеженнями запуску.

Для узагальнення результатів було використано кілька груп показників: утилізація ресурсів, середній і максимальний час очікування, стабільність домінантних часток, динаміка показників боргу і кредиту, кількість непродуктивних кроків планування та частота технічно нездійсненних запусків. Сукупне використання саме цих характеристик дозволило сформувати повну картину роботи системи. Було встановлено, що реалізований метод не завжди дає абсолютний максимум за кожним окремим показником у відриві від інших, однак саме він забезпечує найбільш збалансоване поєднання ефективності використання ресурсів, часової справедливості та стійкості поведінки системи.

Важливим результатом оцінювання стало підтвердження того, що параметри механізму компенсації можуть використовуватися як засіб керування балансом між швидкістю вирівнювання перекосів і загальною ефективністю середовища. За м’якших налаштувань система демонструвала плавнішу поведінку та стабільнішу

утилізацію, тоді як за жорсткіших - швидше обмежувала накопичення перекосів між агентами. Це дозволило зробити висновок, що реалізоване рішення має достатній запас гнучкості та може бути налаштоване залежно від того, який аспект є більш важливим для конкретного середовища: максимальне обмеження несправедливості або пріоритет вищої ресурсної віддачі.

Окремо було підтверджено і практичну доцільність модульної програмної реалізації. Реалізований програмний засіб дав змогу не лише провести обчислення та запуск експериментів, а й детально простежити поведінку кожного компонента в процесі моделювання. Наявність окремих модулів для обчислення домінантної частки, підтримки механізму боргу і кредиту, перевірки здійсненності та журналювання результатів забезпечила достатню прозорість експериментального аналізу. Це дозволило робити висновки не лише за кінцевими показниками, а й за внутрішньою динамікою роботи системи, що підвищило достовірність отриманих результатів.

У підсумку оцінювання ефективності показало, що реалізоване рішення забезпечує вдале поєднання багаторесурсної справедливості, часової стабільності та прикладної працездатності. Було підтверджено, що поєднання моделі домінантної частки, механізму боргу і кредиту та перевірки здійсненності запуску дозволяє зменшити перекоси в розподілі ресурсів без критичного погіршення загальної утилізації середовища. Це дає підстави вважати запропонований програмний засіб ефективним інструментом для послідовного розподілу ресурсів у багатоагентних комп’ютерних системах.

4.6 Висновки до розділу 4

У четвертому розділі було завершено прикладне опрацювання запропонованого рішення та виконано його експериментальну перевірку в умовах модельованого багатоагентного середовища. Насамперед було описано реалізований програмний засіб, у якому відтворено всі ключові складові розробленого методу: подання агентів і заявок, керування станом ресурсного

середовища, обчислення домінантної частки, підтримку механізму боргу і кредиту, вибір кандидата на обслуговування, перевірку здійсненності запуску та журналювання результатів. Це дозволило перейти від теоретичної та алгоритмічної частини роботи до повноцінної програмної реалізації, придатної для проведення серії контрольованих експериментів.

У межах розділу було сформовано та налаштовано експериментальний стенд, на основі якого виконувалося моделювання різних режимів навантаження. Для перевірки поведінки системи було реалізовано сценарії зі стабільним, нерівномірним, хвилеподібним і дефіцитним навантаженням, а також сценарії з технічно нездійсненними заявками. Такий підхід дозволив оцінити роботу програмного засобу не в одному спрощеному випадку, а в кількох характерних режимах, що відображають реальні особливості багатоагентного середовища. Унаслідок цього результати перевірки виявилися більш змістовними та придатними для подальшого узагальнення.

Проведений аналіз показав, що реалізований програмний засіб забезпечує стабільну роботу в різних режимах надходження заявок. У стабільних сценаріях система підтримувала рівномірніші значення домінантних часток і не допускала виражених перекосів між агентами. У сценаріях зі змінним навантаженням було підтверджено, що механізм боргу і кредиту поступово компенсує накопичені часові відхилення та не дозволяє окремим агентам довго зберігати перевагу лише за рахунок більш вдалого моменту надходження заявок. Це дало змогу підтвердити працездатність часової складової справедливого розподілу, закладеної в основу розробленого методу.

Окремо було встановлено, що модель домінантної частки забезпечує коректніше врахування багаторесурсного характеру середовища, особливо в умовах локального дефіциту окремих ресурсів. Саме орієнтація на найбільш обмежувальний для агента ресурс дозволила уникати ситуацій, коли спрощене оцінювання приховує реальні перекоси в системі. Це підтвердило, що використання домінантної частки є обґрунтованим не лише на рівні теоретичної моделі, а й у прикладній програмній реалізації.

Важливим результатом четвертого розділу стало порівняння запропонованого рішення з базовими підходами розподілу ресурсів. Було встановлено, що простіші схеми, засновані на черговості надходження або циклічному обслуговуванні, поступаються за рівнем часової справедливості й гірше працюють у динамічних багаторесурсних сценаріях. Спрощений варіант без механізму боргу і кредиту показав кращу поведінку, ніж базові схеми, проте також виявився менш стійким до накопичення попередніх перекосів. У підсумку саме запропонований метод продемонстрував найбільш збалансоване поєднання справедливості, стійкості та прикладної працездатності.

Під час узагальненого оцінювання ефективності було підтверджено, що реалізований програмний засіб забезпечує прийнятний рівень утилізації ресурсів, не допускаючи критичного погіршення загальної продуктивності середовища. Водночас середній і максимальний час очікування виявилися більш керованими, ніж у базових підходах, а кількість непродуктивних кроків планування залишалася в допустимих межах навіть за наявності конфліктних або нездійсненних заявок.

У підсумку четвертий розділ підтвердив, що розроблений метод успішно реалізовано в програмній формі, а результати експериментальної перевірки засвідчили його працездатність, стійкість до зміни навантаження та переваги над простішими схемами розподілу ресурсів. Отримані результати створюють завершену основу для формування загальних висновків усієї роботи, у яких буде узагальнено теоретичні, методичні, алгоритмічні та прикладні результати виконаної роботи.

ВИСНОВКИ

У роботі було розв’язано актуальне завдання послідовного розподілу ресурсів у багатоагентних комп’ютерних системах на основі справедливого планувальника. Актуальність обраної тематики зумовлювалася тим, що в сучасних обчислювальних середовищах агенти одночасно конкурують за кілька типів обмежених ресурсів, а сам процес обслуговування відбувається в умовах динамічного надходження заявок, нерівномірного навантаження та наявності додаткових обмежень запуску. За таких умов традиційні підходи до планування не завжди забезпечують збалансований доступ до ресурсів, що призводить до накопичення перекосів, зростання затримок і втрати передбачуваності поведінки системи.

У першому розділі було проаналізовано предметну область, розглянуто особливості багатоагентних комп’ютерних систем, підходи до справедливості в задачах розподілу ресурсів, а також роль справедливих планувальників у послідовному розподілі заявок. Проведений аналіз показав, що в умовах багаторесурсного середовища недостатньо враховувати лише один показник навантаження або орієнтуватися тільки на момент надходження заявки. Було встановлено, що для коректного розподілу необхідно поєднувати оцінювання поточного ресурсного стану агента з урахуванням його попередньої історії обслуговування. Це дозволило сформулювати постановку задачі та визначити основні вимоги до методу, який повинен одночасно підтримувати справедливість, зберігати прикладну працездатність і враховувати реальні обмеження запуску.

У другому розділі було розроблено методичну основу запропонованого рішення. Багатоагентне середовище було формалізовано як сукупність агентів, заявок, типів ресурсів і обмежень запуску, що змінюються в часі. Для оцінювання багаторесурсного положення агента було використано модель домінантної частки, яка дозволила коректно зіставляти агентів із різними профілями споживання ресурсів. Для компенсації часових перекосів було введено механізм боргу і кредиту, що відображає відхилення фактичного обслуговування від бажаного

режиму. На цій основі було побудовано правило вибору задачі, у якому поєднуються багаторесурсна оцінка, урахування накопичених відхилень і перевірка фактичної здійсненності запуску. У підсумку було сформовано цілісний метод послідовного розподілу ресурсів, придатний до подальшої алгоритмічної та програмної реалізації.

У третьому розділі було розроблено алгоритмічне та програмне подання запропонованого методу. Було обґрунтовано модульну архітектуру програмного засобу, у якій виділено підсистему керування станом середовища, аналітичний блок, модуль підтримки механізму боргу і кредиту, ядро планувальника, блок перевірки здійсненності запуску та компонент журналювання. Деталізовано алгоритм роботи справедливого планувальника, визначено процедури обчислення домінантної частки й оновлення стану агентів, а також описано організацію інформаційної взаємодії між компонентами системи. Це дозволило перейти від концептуальної схеми до структурованого програмного рішення, у якому всі основні елементи методу отримали прикладне втілення.

У четвертому розділі було реалізовано програмний засіб та проведено його експериментальну перевірку на основі спеціально побудованого стенда моделювання. Для аналізу поведінки системи було сформовано сценарії зі стабільним, нерівномірним, хвилеподібним і дефіцитним навантаженням, а також сценарії з частково нездійсненними заявками. Результати експериментів показали, що реалізоване рішення зберігає працездатність у різних режимах функціонування, забезпечує більш збалансований розподіл ресурсів між агентами та зменшує ризик тривалого недообслуговування в динамічному середовищі. Було підтверджено, що модель домінантної частки є ефективною для багаторесурсного оцінювання, а механізм боргу і кредиту забезпечує стабілізацію справедливості на часовому інтервалі.

Порівняння з базовими підходами до розподілу ресурсів показало, що запропоноване рішення забезпечує кращий баланс між справедливістю, стійкістю та прикладною ефективністю. Простіші схеми демонстрували або накопичення перекосів у часі, або гіршу узгодженість із багаторесурсною природою середовища,

або підвищену чутливість до технічно нездійсненних запусків. Реалізований метод не завжди мав найвище значення за кожним окремим локальним показником, проте саме він продемонстрував найкращу загальну збалансованість поведінки системи. Це дозволило підтвердити доцільність поєднання домінантної частки, механізму компенсації відхилень і перевірки здійсненності запуску в межах єдиного справедливого планувальника.

Наукова новизна отриманих результатів полягає в розробленні методу послідовного розподілу ресурсів у багатоагентних комп'ютерних системах, який поєднує багаторесурсне оцінювання стану агента через домінантну частку з часовою компенсацією перекосів на основі механізму боргу і кредиту. Удосконалено підхід до вибору заявки на обслуговування за рахунок поєднання критерію справедливості з процедурою перевірки фактичної здійсненності запуску. Дістало подальшого розвитку програмне подання справедливого планувальника для багатоагентного середовища з динамічним навантаженням і дискретними ресурсними обмеженнями.

Практичне значення отриманих результатів полягає в тому, що розроблений метод доведено до рівня реалізованого програмного засобу, придатного для моделювання, аналізу та подальшої адаптації до прикладних задач розподілу ресурсів. Побудований програмний прототип і експериментальний стенд можуть бути використані для дослідження поведінки планувальників у багатоагентних системах, для порівняння різних політик розподілу та для подальшого розширення засобів керування ресурсами в динамічних обчислювальних середовищах.

У підсумку поставлену мету було досягнуто. Розроблено, реалізовано та перевірено метод послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника, який забезпечує узгодження багаторесурсної справедливості, часової стабільності та прикладної працездатності. Отримані результати підтвердили доцільність використання запропонованого підходу в задачах керування ресурсами в умовах динамічного багатоагентного навантаження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Ju P., et al. Achieving Fairness in Multi-Agent MDP Using Reinforcement Learning : conference paper. 2024. URL: https://proceedings.iclr.cc/paper_files/paper/2024/file/473cb5596e2b84648753ce26484d79c8-Paper-Conference.pdf (дата звернення: 26.02.2026).

2. Trabelsi Y., et al. Fairness and Optimization in Dynamic Multiagent Allocation. *Proceedings of IJCAI 2024*. 2024. URL: https://www.ijcai.org/proceedings/2024/973 (дата звернення: 26.02.2026).

3. Namyar P., et al. Solving Max-Min Fair Resource Allocations Quickly on Large Instances. *NSDI ’24*. 2024. URL: https://www.usenix.org/conference/nsdi24/presentation/namyar-solving (дата звернення: 26.02.2026).

4. Samanta A., Stutsman R. Fair, Efficient Multi-Resource Scheduling for Stateless Serverless Functions with Anubis. *IEEE CCGrid 2024*. 2024. URL: https://www.researchgate.net/publication/384743296_Fair_Efficient_Multi-Resource_Scheduling_for_Stateless_Serverless_Functions_with_Anubis (дата звернення: 26.02.2026).

5. ACM Digital Library. Exploring Enhanced Dominant Resource Fairness Using Linear Programming Based Weights. 2024. URL: https://dl.acm.org/doi/10.1109/UCC63386.2024.00031 (дата звернення: 26.02.2026).

6. Gough E., et al. Evaluation of Kubernetes Schedulers for a Community Cloud Computing Model. *PEARC ’24*. 2024. URL: https://dl.acm.org/doi/fullHtml/10.1145/3626203.3670520 (дата звернення: 26.02.2026).

7. Guo H., et al. Dynamic Multi-Resource Fair Allocation with Elastic Demands. *Journal of Grid Computing*. 2024. URL: https://link.springer.com/article/10.1007/s10723-024-09754-6 (дата звернення: 26.02.2026).

8. Li X., et al. Fair multiresource allocation with access constraint in cloud-edge collaborative computing systems (TSF-CE). *Future Generation Computer Systems*. 2024.

URL: https://www.sciencedirect.com/science/article/abs/pii/S0167739X24002000 (дата звернення: 26.02.2026).

9. Banerjee S., et al. Online Fair Allocation of Perishable Resources. arXiv preprint. 2024. URL: https://arxiv.org/pdf/2406.02402 (дата звернення: 26.02.2026).

10. Bandopadhyay S., et al. Conflict and Fairness in Resource Allocation : arXiv preprint. 2024. URL: https://arxiv.org/pdf/2403.04265 (дата звернення: 26.02.2026).

11. Reuel A., et al. Fairness in Reinforcement Learning (including multi-agent settings) : arXiv preprint. 2024. URL: https://arxiv.org/pdf/2405.06909 (дата звернення: 26.02.2026).

12. Dong H., et al. OSDEC: Online Scheduling for DEferrable Jobs in Cloud : arXiv preprint. 2024. URL: https://arxiv.org/pdf/2406.01047 (дата звернення: 26.02.2026).

13. Mahmood T., Shahbazian R., Trubitsyna I. Fairness-Driven Explainable Learning in Multi-Agent Reinforcement Learning. *CEUR Workshop Proceedings*. 2024. URL: https://ceur-ws.org/Vol-3945/paper2.pdf (дата звернення: 26.02.2026).

14. Kovalenko V. V., Bukasov M. M. Scheduling Methods and Models for Kubernetes Orchestrator. *Visnyk of Vinnytsia Politechnical Institute*. 2024. URL: https://www.researchgate.net/publication/384138148_Scheduling_Methods_and_Models_for_Kubernetes_Orchestrator (дата звернення: 26.02.2026).

15. Sudnicina K. Task-in-Pod Scheduling Support for Kubernetes and Apache Spark : bachelor thesis. 2024. URL: https://atlarge-research.com/pdfs/2024-karina-bsc_thesis.pdf (дата звернення: 26.02.2026).

16. Moakhar S. P., et al. An efficient mechanism for function scheduling and placement in Edge FaaS (EF-TTC). *Journal of Network and Computer Applications*. 2024. URL: https://www.sciencedirect.com/science/article/abs/pii/S1084804524000675 (дата звернення: 26.02.2026).

17. Sheng Y., et al. Fairness in serving large language models. *Proceedings ACM*. 2024. URL: https://dl.acm.org/doi/abs/10.5555/3691938.3691990 (дата звернення: 26.02.2026).

18. Cooperation and Fairness in Multi-Agent Reinforcement Learning (ACM). 2024. URL: https://dl.acm.org/doi/10.1145/3702012 (дата звернення: 26.02.2026).

19. OpenReview. Achieving Fairness in Multi-Agent MDP Using Reinforcement Learning. 2024. URL: https://openreview.net/forum?id=yoVq2BGQdP (дата звернення: 26.02.2026).

20. OpenReview. Learning to be Fair in Multi-agent Resource Allocation (DECAF). 2024. URL: https://openreview.net/pdf?id=Y3YHGouOjZ (дата звернення: 26.02.2026).

21. Doostmohammadian M., et al. Survey of distributed algorithms for distributed resource allocation over multi-agent systems. *Digital Signal Processing*. 2025. URL: https://www.sciencedirect.com/science/article/abs/pii/S1367578824000518 (дата звернення: 26.02.2026).

22. Hady M. A., Hu S., Pratama M., et al. Multi-agent reinforcement learning for resources allocation optimization: a survey. *Artificial Intelligence Review*. 2025. URL: https://link.springer.com/article/10.1007/s10462-025-11340-5 (дата звернення: 26.02.2026).

23. Hady M. A., et al. Multi-Agent Reinforcement Learning for Resources Allocation Optimization: A Survey : arXiv preprint. 2025. URL: https://arxiv.org/abs/2504.21048 (дата звернення: 26.02.2026).

24. Cao S., et al. Locality-aware Fair Scheduling in LLM Serving : arXiv preprint. 2025. URL: https://arxiv.org/abs/2501.14312 (дата звернення: 26.02.2026).

25. Cao S., et al. Locality-aware Fair Scheduling in LLM Serving : arXiv PDF. 2025. URL: https://arxiv.org/pdf/2501.14312 (дата звернення: 26.02.2026).

26. DeLoye J., et al. Extending Delayed Fair Sharing: A Generalizable Approach : technical report. 2025. URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-97.pdf (дата звернення: 26.02.2026).

27. Mahajan Y. W. X. Z. R., et al. Programmable and Adaptive Scheduling for Distributed Systems. *HotNets 2025*. 2025. URL:

https://conferences.sigcomm.org/hotnets/2025/papers/hotnets25-final95.pdf (дата звернення: 26.02.2026).

28. Li J., Wang H., Wang J., Zhang Y. Max-Min Share-Based Mechanism for Multi-Resource Fair Allocation with Bounded Number of Tasks in Cloud Computing System. *Mathematics (MDPI)*. 2025. URL: https://www.mdpi.com/2227-7390/13/13/2214 (дата звернення: 26.02.2026).

29. Zhang F., et al. Fair-efficient allocation mechanism with meta-types resources in cloud computing. *Scientific Reports*. 2025. URL: https://www.nature.com/articles/s41598-025-22657-0 (дата звернення: 26.02.2026).

30. Tarasova E., et al. Decentralized adaptive task allocation for dynamic multi-agent systems. *Scientific Reports*. 2025. URL: https://www.nature.com/articles/s41598-025-21709-9 (дата звернення: 26.02.2026).

31. Yu X., et al. Dynamic multi objective task scheduling in cloud using reinforcement learning. *Scientific Reports*. 2025. URL: https://www.nature.com/articles/s41598-025-29280-z (дата звернення: 26.02.2026).

32. Fang Z., et al. Efficient Task Allocation in Multi-Agent Systems Using Genetic Algorithm-Enhanced PPO (GAPPO). *Applied Sciences (MDPI)*. 2025. URL: https://www.mdpi.com/2076-3417/15/4/1905 (дата звернення: 26.02.2026).

33. Farid M., et al. Optimizing Kubernetes with Multi-Objective Scheduling (review). *Computers (MDPI)*. 2025. URL: https://www.mdpi.com/2073-431X/14/9/390 (дата звернення: 26.02.2026).

34. Xiao D., et al. Load-balanced scheduling optimization strategy for Kubernetes with RDMA. *Computer Communications*. 2025. URL: https://www.sciencedirect.com/science/article/abs/pii/S0140366425002178 (дата звернення: 26.02.2026).

35. Kamrani A. S., et al. Multi-agent deep reinforcement learning with online and fairness concepts (resource allocation). *Results in Engineering*. 2025. URL: https://www.sciencedirect.com/science/article/pii/S2666827025000039 (дата звернення: 26.02.2026).

36. Zhou Z., et al. Fairness-Aware Multi-Agent Learning-based Task Offloading and Resource Allocation. 2025. URL: https://www.tkn.tu-berlin.de/bib/zhou2025fairness-aware/zhou2025fairness-aware.pdf (дата звернення: 26.02.2026).

37. Papadopoulos G., et al. An Extended Benchmarking of Multi-Agent Reinforcement Learning. *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025)*. 2025. URL: https://www.ifaamas.org/Proceedings/aamas2025/pdfs/p1613.pdf (дата звернення: 26.02.2026).

38. Bredereck R., et al. Computing Efficient Envy-Free Partial Allocations of Indivisible Goods. *AAMAS 2025*. 2025. URL: https://www.ifaamas.org/Proceedings/aamas2025/pdfs/p390.pdf (дата звернення: 26.02.2026).

39. Conati A., et al. Computing Efficient and Envy-Free Allocations under Dichotomous Preferences. *AAMAS 2025*. 2025. URL: https://www.ifaamas.org/Proceedings/aamas2025/pdfs/p510.pdf (дата звернення: 26.02.2026).

40. Ranjan R., Gupta S., Singh S. N. Fairness in AI Multi-Agent: Foundation, Framework and (Survey) : arXiv preprint. 2025. URL: https://arxiv.org/pdf/2502.07254 (дата звернення: 26.02.2026).

41. Zargari F., et al. Online Multi-Class Selection with Group Fairness Guarantee : arXiv preprint. 2025. URL: https://arxiv.org/pdf/2510.21055 (дата звернення: 26.02.2026).

42. NeurIPS. Online Multi-Class Selection with Group Fairness Guarantee. 2025. URL: https://neurips.cc/virtual/2025/poster/118026 (дата звернення: 26.02.2026).

43. Kumar A., Yeoh W. A General Incentives-Based Framework for Fairness in Multi-agent Resource Allocation (GIFF) : arXiv preprint. 2025. URL: https://arxiv.org/pdf/2510.26740 (дата звернення: 26.02.2026).

44. Gharbi A., et al. Integrating Borda Count and K-Means Approaches with Fair Resource Allocation in Multi-Agent Systems. *Mathematics (MDPI)*. 2025. URL: https://www.mdpi.com/2227-7390/13/15/2355 (дата звернення: 26.02.2026).

45. Wu X., et al. Approximate envy-freeness in indivisible resource allocation under knapsack constraints. *Information and Computation*. 2025. URL: https://www.sciencedirect.com/science/article/abs/pii/S0890540124001299 (дата звернення: 26.02.2026).

46. Alatawi M. N., et al. Optimizing Multitenancy: Adaptive Resource Allocation (fairness, RL) in serverless/cloud. *Electronics (MDPI)*. 2025. URL: https://www.mdpi.com/2079-9292/14/15/3004 (дата звернення: 26.02.2026).

47. Liu W., et al. Resource Scheduling Algorithm for Edge Computing Networks Based on Multi-Objective Optimization. *Applied Sciences (MDPI)*. 2025. URL: https://www.mdpi.com/2076-3417/15/19/10837 (дата звернення: 26.02.2026).

48. Luo F., et al. An Efficient Max-Min Fair Resource Optimization Algorithm : arXiv preprint. 2025. URL: https://arxiv.org/pdf/2507.04201 (дата звернення: 26.02.2026).

49. arXiv. Task Scheduling in Geo-Distributed Computing: A Survey. 2025. URL: https://arxiv.org/html/2501.15504v1 (дата звернення: 26.02.2026).

50. arXiv. A Survey on Task Scheduling in Carbon-Aware Container/Kubernetes Environments. 2025. URL: https://arxiv.org/html/2508.05949v1 (дата звернення: 26.02.2026).

51. Icarte-Ahumada G., et al. Learning in multi-agent systems to solve scheduling problems. 2024. URL: https://www.scielo.cl/pdf/ingeniare/v32/0718-3305-ingeniare-32-14.pdf (дата звернення: 26.02.2026).

52. Lovchuk O. I. Кваліфікаційна робота (Kubernetes, планувальники, serverless). 2024. URL: https://elartu.tntu.edu.ua/bitstream/lib/47025/1/Mag_2024_SNm_61_Lovchuk_O_I.pdf (дата звернення: 26.02.2026).

53. МОН. Дорожня карта науки, технологій та інновацій (НТІ). 2024. URL: https://mon.gov.ua/static-

objects/mon/sites/1/news/2024/01/03/Dorozhnya.karta.vykoryst.nauky.tekhnolohiy.ta.innovatsiy-03.01.2024-1.1.pdf (дата звернення: 26.02.2026).

54. Кабінет Міністрів України. Деякі питання визначення середньострокових пріоритетних напрямів інноваційної діяльності : постанова. 2024. URL: https://zakon.rada.gov.ua/go/787-2024-%D0%BF (дата звернення: 26.02.2026).

55. Міністерство економіки України. План для Ukraine Facility. 2024. URL: https://www.ukrainefacility.me.gov.ua/wp-content/uploads/2024/03/plan-ukraine-facility.pdf (дата звернення: 26.02.2026).

56. Державна судова адміністрація України. Концепція ЄСІКС. 2025. URL: https://court.gov.ua/storage/portal/dsa/normatyvno-pravova%20baza/N_178_2025_dodatok.pdf (дата звернення: 26.02.2026).

57. КПІ ім. Ігоря Сікорського. Факультетський каталог освітньої програми (хмарні системи, Kubernetes тощо). 2024. URL: https://osvita.kpi.ua/sites/default/files/f-catalog/fkat_126_oppb_iis_2024.pdf (дата звернення: 26.02.2026).

58. НАУ. Abstracts of XXV International conference (POLIT 2025). 2024. URL: https://nau.edu.ua/site/variables/docs/docsmenu/studnauka/polit2025/Polit-2025_FKNT.pdf (дата звернення: 26.02.2026).

59. LNTU Repository. Матеріали/стаття (Kubernetes, kube-scheduler, 2024). 2024. URL: https://repository.lntu.edu.ua/bitstreams/3d3fc4c5-6711-46f2-a79c-4414c20b22a6/download (дата звернення: 26.02.2026).

60. ВНТУ. Магістерська кваліфікаційна робота (CFS, Kubernetes, обмеження ресурсів). 2024. URL: https://docs.vntu.edu.ua/getfile.php/7872.pdf (дата звернення: 26.02.2026).

61. ХНАДУ. Збірник матеріалів “Сталий розвиток сучасних інформаційних технологій”. 2025. URL: https://mf.khadi.kharkov.ua/fileadmin/F-MECHANIC/Комп_ютерних_технологій_і_мехатроніки/2025/Конференції/Збірник_матеріалів_Сталий_розвиток_сучасних_інформаційних_технологій.pdf (дата звернення: 26.02.2026).

62. KTH. Comparative analysis of fair scheduling algorithms in resource intensive applications : thesis. 2025. URL: https://kth.diva-portal.org/smash/get/diva2%3A1970590/FULLTEXT01.pdf (дата звернення: 26.02.2026).

63. Gharbi A., et al. Framework for fair resource allocation in self-governing multi-agent systems. *Mathematics (MDPI)*. 2025. URL: https://www.mdpi.com/2227-7390/13/15/2355 (дата звернення: 26.02.2026).

64. Ma Y., et al. Multi-Agent Deep Reinforcement Learning for Joint Task Offloading and Fair Resource Allocation. *Sensors (MDPI)*. 2025. URL: https://www.mdpi.com/1424-8220/25/19/6160 (дата звернення: 26.02.2026).

65. arXiv. Interactional Fairness in LLM Multi-Agent Systems. 2025. URL: https://arxiv.org/html/2505.12001v1 (дата звернення: 26.02.2026).

66. arXiv. Cooperation and Fairness in Multi-Agent Reinforcement Learning. 2024. URL: https://arxiv.org/html/2410.14916v1 (дата звернення: 26.02.2026).

67. ACM Digital Library. Balancing Fairness and Performance in Multi-User Spark (UWFQ). 2026. URL: https://dl.acm.org/doi/10.1145/3772052.3772214 (дата звернення: 26.02.2026).

68. Cloud Native Now. Kubernetes Could Use a Different Linux Scheduler (Latency-Aware Group Scheduling). 2026. URL: https://cloudnativenow.com/features/kubernetes-could-use-a-different-linux-scheduler/ (дата звернення: 26.02.2026).

69. INFORMS. Fair and Efficient Multi-resource Allocation for Cloud Computing. *Mathematics of Operations Research*. 2026. URL: https://pubsonline.informs.org/doi/10.1287/moor.2024.0714 (дата звернення: 26.02.2026).

70. Ejaz M., et al. FAIR-DQL: Fairness-Aware Deep Q-Learning for Enhanced Resource Allocation. *Computers, Materials & Continua*. 2026. URL: https://www.sciencedirect.com/org/science/article/pii/S1546221826000184 (дата звернення: 26.02.2026).

71. Sha J., et al. A multi-agent reinforcement learning scheduling algorithm for ride-sharing. *Scientific Reports*. 2026. URL: https://www.nature.com/articles/s41598-026-35004-8 (дата звернення: 26.02.2026).

72. Tian K., et al. An autonomous energy-aware resource scheduling mechanism in serverless edge computing. *Artificial Intelligence Review*. 2026. URL: https://link.springer.com/article/10.1007/s10462-026-11495-9 (дата звернення: 26.02.2026).

73. Li Z., et al. Envy-Free Allocation of Indivisible Goods via Noisy Queries : arXiv preprint. 2026. URL: https://www.arxiv.org/pdf/2602.06361 (дата звернення: 26.02.2026).

74. ACM Digital Library. On the Existence of Envy-Free Allocations Beyond Additive (stochastic valuations). 2024. URL: https://dl.acm.org/doi/10.1145/3670865.3673466 (дата звернення: 26.02.2026).

75. Caragiannis I., et al. On Interim Envy-Free Allocation Lotteries. *Mathematics of Operations Research*. 2025. URL: https://pubsonline.informs.org/doi/abs/10.1287/moor.2023.0203 (дата звернення: 26.02.2026).

76. ACM Digital Library. A Case of Multi-Resource Fairness for Serverless Workflows (WIP). 2023/2024 (використовуйте як контекст - не основне). URL: https://dl.acm.org/doi/10.1145/3578245.3585033 (дата звернення: 26.02.2026).

77. Kubernetes Docs. kube-scheduler (concepts, scheduling & eviction). 2025. URL: https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/ (дата звернення: 26.02.2026).

78. arXiv. Fairness-Aware Batch Formation for LLM Inference. 2025. URL: https://arxiv.org/html/2510.14392v1 (дата звернення: 26.02.2026).

79. TechRxiv. LLM Inference Scheduling: A Survey of Techniques. 2025. URL: https://www.techrxiv.org/users/994660/articles/1355915/master/file/data/LLM_Scheduling_Survey_Arxiv_06Oct2025/LLM_Scheduling_Survey_Arxiv_06Oct2025.pdf?inline=true (дата звернення: 26.02.2026).

80. Kubernetes (ACM/HotNets reference in paper). On the In-Depth Cluster Scheduling and Management (Alibaba Cloud blog cited in scheduling research). 2025. URL: https://www.alibabacloud.com/blog/on-the-in-depth-cluster-scheduling-and-management_598012 (дата звернення: 26.02.2026).

81. Комп’ютерні інтелектуальні системи та мережі (КІСМ-2026) : XIX Всеукраїнська науково-практична WEB конференція аспірантів, студентів та молодих вчених. URL: https://sites.google.com/view/kicm/ (дата звернення: 26.02.2026).

ДОДАТОК А

(обов'язковий)

ПУБЛІКАЦІЙНІ МАТЕРІАЛИ ЗДОБУВАЧА

Семенюк Є.В.
Хмельницький національний університет
Віжевський П.В.
асистент, Хмельницький національний університет

МЕТОД ПОСЛІДОВНОГО РОЗПОДІЛУ РЕСУРСІВ У БАГАТОАГЕНТНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ НА ОСНОВІ СПРАВЕДЛИВОГО ПЛАНУВАЛЬНИКА

У багатоагентних комп'ютерних системах агенти конкурують за спільні обчислювальні та комунікаційні ресурси. За динамічних навантажень і багаторесурсних вимог послідовний розподіл ресурсів потребує механізму, що одночасно обмежує голодування, контролює перекоси часток у часі та зберігає високу утилізацію.

Ключовою ідеєю методу є використання справедливого планувальника як ядра, яке не лише вибирає наступну заявку, а й "пам'ятає", хто скільки отримав ресурсу в минулому. Практично це реалізується через просту, але ефективну логіку боргу і кредиту. Якщо агент протягом певного часу отримує менше, ніж очікувано згідно з політикою, то в системі накопичується його "кредит" - умовний показник того, що його потрібно компенсувати. Якщо ж агент упродовж інтервалу отримував ресурс інтенсивніше й фактично випереджав інших, формується "борг", який тимчасово зменшує його шанси бути обраним знову. Завдяки цьому планувальник перестає бути суто реактивним: він не просто реагує на поточну чергу, а вирівнює розподіл у часі, обмежуючи тривалі відхилення часток доступу.

Друга ідея потрібна через багаторесурсність. У реальних кластерах справедливість не можна коректно визначити лише за CPU або лише за пам'яттю, бо агенти відрізняються профілем навантаження. Якщо орієнтуватися на один ресурс, виникають перекоси: агент може "тиснути" на найдефіцитніший ресурс і водночас формально виглядати "чесно" за іншим виміром. Тому метод спирається на логіку домінантної частки, характерну для підходу DRF: для кожного агента виділяється той ресурс, за яким його частка найбільша, і саме за цим показником агент порівнюється з іншими. Інтуїція тут проста: справедливість забезпечується тоді, коли жоден агент не може стабільно утримувати найбільшу частку найважливішого для нього ресурсу, витісняючи інших, лише через те, що в системі ще є надлишок інших ресурсів.

У процесі роботи планувальник підтримує для кожного агента два узагальнені уявлення про його стан: наскільки він "боргує" або "має кредит" з погляду справедливості у часі та яка його поточна домінантна частка з погляду багаторесурсного споживання. Коли приходить черговий момент планування, система не вибирає задачу навмання й не орієнтується лише на пріоритет у черзі; натомість вона намагається надати можливість тим агентам, які недоотримували ресурс, але при цьому не дозволяє агентам із надмірною домінантною часткою безперервно посилювати свою перевагу. Така стратегія дає практичний ефект:

розподіл стає стабільнішим, а поведінка системи - передбачуванішою для різних класів навантаження.

Окремою проблемою є те, що ресурси в реальних системах часто дискретні. Навіть якщо "за справедливістю" зараз має отримати ресурс певний агент, його задача може не запускатися через пакування або обмеження розміщення: потрібен конкретний тип GPU, мінімальна RAM, афініті до вузла, обмеження на зону, або ж ресурси формально є, але вони розбиті на фрагменти й не складаються в потрібну конфігурацію. Тому метод передбачає етап перевірки здійсненності запуску, який відокремлює справедливий вибір кандидата від фактичної можливості виконання. Якщо виявляється, що задачу агента наразі запустити неможливо, планувальник переходить до наступного кандидата, не руйнуючи загальної логіки справедливості. Це критичний елемент для практичної працездатності, оскільки він зменшує простої та ситуації, коли планувальник "зациклюється" на агентах, задачі яких тимчасово нездійсненні.

Після кожного успішного запуску система оновлює накопичені показники. Агент, який щойно отримав ресурс, частково втрачає свій кредит або нарощує борг, а його домінантна частка зростає відповідно до того, який ресурс став для нього найобмежувальнішим. Натомість агенти, які чекали, поступово накопичують кредит, що підвищує їх шанси у наступних кроках. Завдяки такому механізму планувальник підтримує справедливість на часовому інтервалі без необхідності складних глобальних перерахунків: корекція відбувається малими кроками в кожному рішенні, а загальний ефект проявляється як обмеження перекосів та зменшення ризику голодування.

Оцінювати запропонований метод доцільно через показники, що відображають не лише "середню справедливість", а й її стабільність. У багатоагентних системах важливо бачити, чи не виникає тривалих періодів, коли окремий агент систематично не отримує ресурс, а також як швидко система повертається до балансу після сплесків навантаження. Паралельно слід контролювати ефективність використання ресурсів, оскільки надмірна орієнтація на вирівнювання може підвищувати фрагментацію й збільшувати затримки. У цьому сенсі сильна сторона підходу полягає в тому, що він дозволяє налаштовувати "жорсткість" справедливості через параметри боргу/кредиту: можна зробити систему більш агресивною у компенсації перекосів або, навпаки, більш орієнтованою на утилізацію, не відмовляючись від базового принципу стабілізації часток у часі.

У підсумку метод послідовного розподілу ресурсів на основі справедливого планувальника забезпечує узгодження двох ключових вимог багатоагентного середовища: справедливості як властивості траєкторії розподілу в часі та ефективності, необхідної для стабільної роботи інфраструктури під навантаженням. Використання логіки борг/кредит дозволяє компенсувати відхилення та обмежувати голодування, а врахування домінантної частки забезпечує коректну роботу в багаторесурсних умовах. Додавання перевірки здійсненності запуску робить підхід прикладним для кластерів із дискретними ресурсами й політиками розміщення, що важливо для реального використання методу в сучасних комп'ютерних системах.

ДОДАТОК Б

(обов’язковий)

ПРЕЗЕНТАЦІЯ

Метод послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника

Євгеній СЕМЕНЮК КІ2м-24-1

Актуальність, об'єкт, предмет, мета

Об'єкт

Процеси розподілу обчислювальних ресурсів у багатоагентних системах за умов динамічного навантаження.

Предмет

Методи та алгоритми послідовного розподілу ресурсів з урахуванням багаторесурсних обмежень і часових відхилень.

Мета

Розроблення методу, що забезпечує збалансований доступ агентів до ресурсів і зменшує накопичення перекосів у часі.

Основна ідея запропонованого методу

Домінантна частка ресурсу

Оцінювання агента за найбільш обмежувальним ресурсом у поточний момент — коректна робота в багаторесурсному середовищі.

Механізм боргу і кредиту

Компенсація часових перекосів: пріоритет агента зростає при недоотриманні ресурсу і знижується при перевазі.

Перевірка здійсненності

Навіть за високого пріоритету задача запускається лише за наявності достатніх ресурсів та виконання обмежень.

Структура та алгоритм роботи системи

Багатоагентне середовище подано як сукупність агентів, заявок, ресурсів і правил запуску.

Для кожного агента обчислюється інтегральний пріоритет на основі домінантної частки та накопиченого відхилення.

Поєднання справедливості розподілу з практичною працездатністю системи.

Програмна реалізація

Керування станом

Модуль відстеження поточного стану системи та агентів.

Аналітичний модуль

Обчислення домінантної частки ресурсу для кожного агента.

Борг і кредит

Блок підтримки механізму компенсації часових перекосів.

Перевірка здійсненності

Модуль контролю можливості запуску заявки.

Журналювання

Компонент запису результатів для аналізу та порівняння.

Експериментальне дослідження

Досліджувані сценарії

- Стабільне навантаження
- Нерівномірне навантаження
- Хвилеподібний режим
- Дефіцит окремих ресурсів
- Частково нездійсненні заявки

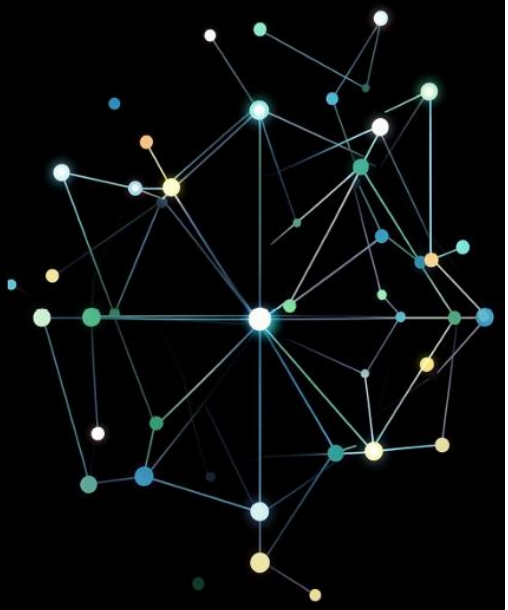
Результати

Більш рівномірний розподіл ресурсів між агентами

Зменшення ризику тривалого недообслуговування

Збереження працездатності у складних умовах

Наукова новизна та практична значимість



Наукова новизна

Метод поєднує багаторесурсне оцінювання на основі домінантної частки, часову компенсацію перекосів у формі боргу і кредиту та перевірку здійсненності запуску заявки.

Практична значимість

Програмний засіб для моделювання розподілу ресурсів: аналіз поведінки планувальників, порівняння стратегій і вдосконалення механізмів керування.

Загальний висновок

01

Аналіз предметної області

Досліджено існуючі підходи до розподілу ресурсів у багатоагентних системах.

02

Розроблення методу

Створено метод послідовного розподілу на основі справедливого планувальника.

03

Реалізація та перевірка

Реалізовано програмний засіб і проведено експериментальну перевірку ефективності.

> Запропоноване рішення забезпечує збалансований доступ агентів до ресурсів, зменшує накопичення перекосів у часі та підтримує ефективну роботу системи в умовах динамічного навантаження.

Дякую за увагу!

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: Євгеній СЕМЕНЮК

Співавтор:

Назва: Метод послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника

Експерт: Олег САВЕНКО

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:1.16%

Коефіцієнт подібності 2:0.31%

Мікропробіли: 3

Заміна букв: 2

Інтервали: 0

Білі знаки: 6

Дата створення звіту: *2026-04-20 13:47:20.0*

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 14:11:49 EEST 2026, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 24.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 8%**

	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
ID: 270557 Назва: МКР Метод послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника Додано в БД: 2026-04-20 Автора: Євгеній СЕМЕНЮК Керівники: Олег САВЕНКО Консультанти: Опоненти:	173733	1227	42931 (25%)	300 (24%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
269758	Назва: Звіт з ПДП Метод послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника Додано в БД: 2026-03-11 Автора: Є.В. Семенюка Керівники: Гнатчук Є.Г._ Консультанти: Опоненти:	41873 (24.0%)	293 (24.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Здобувач: Євгеній СЕМЕНЮК

Тема: Метод послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи магістра:

Кількість листів креслень – ; кількість сторінок записки 82

1. Короткий зміст роботи та прийнятих рішень У роботі запропоновано систему профілювання вразливостей при керуванні розумним будинком

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 першому розділі проведено аналіз предметної області розподілу ресурсів у багатоагентних комп'ютерних системах. У другому розділі запропоновано формалізацію багатоагентного середовища та обґрунтовано критерії справедливості розподілу ресурсів. У третьому розділі запропоновано метод послідовного розподілу ресурсів на основі справедливого планувальника, який поєднує оцінювання домінантної частки з механізмом компенсації часових відхилень у вигляді боргу і кредиту. У четвертому розділі реалізовано програмний засіб та експериментальний стенд для дослідження роботи запропонованого методу. Проведено моделювання різних сценаріїв навантаження, виконано аналіз результатів роботи системи та порівняння з базовими підходами розподілу ресурсів, що дозволило оцінити ефективність запропонованого рішення.

4. Позитивні сторони роботи: Запропонована система розподілу ресурсів у багатоагентному комп'ютерному середовищі дозволяє забезпечити збалансований

доступ агентів до обчислювальних ресурсів, зменшити накопичення перекосів у процесі обслуговування та підвищити стабільність роботи системи в умовах динамічного навантаження, а також визначити основні напрями підвищення ефективності планування та адаптації до змін у ресурсному середовищі.

5. Негативні сторони роботи: У роботі є окремі неточності в описі моделей розподілу ресурсів у багатоагентному середовищі, які потребують уточнення для кращої узгодженості викладу.

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 «задовільно» 60.00 (E)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) д.т.н., професор, Мартинюк В.В., завідувач кафедри автоматизації, комп'ютерно-інтегрованих технологій та робототехніки

"1 травня" 2026р.

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Євгеній СЕМЕНЮК

ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-24-1

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 послідовного розподілу ресурсів у багатоагентних комп'ютерних системах на основі справедливого планувальника
Автор Євгеній СЕМЕНЮК
Освітня програма Інформаційні системи та технології
Рівень вищої освіти другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: д.т.н., професор Олег 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 1.16% і адресується; та системою Anti-Plagiarism складає 24%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

15.12.2025

Завідувач кафедри _____ Підпис — Ольга ПАВЛОВА Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Підпис — Олег САВЕНКО Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Підпис — Олег САВЕНКОі Ім'я, ПРІЗВИЩЕ