

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра технічної кібернетики
(повна назва кафедри)

«На правах рукопису»

УДК _____

«До захисту допущено»

Завідувач кафедри

_____ Ткач М. М.
(підпис) (ініціали, прізвище)

“ ” _____ 20__ р.

Магістерська дисертація

з напрямку підготовки 8.05020102 «Комп'ютеризовані та робототехнічні системи»
(код та назва напрямку підготовки або спеціальності)

на тему: _____
Дослідження оптимальної моделі розподіленої обробки
великих об'ємів даних

Виконав: студент VI курсу, групи ІК-31м
(шифр групи)

_____ Кобилинський Богдан Андрійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник _____ к.т.н., доцент Тимошин Ю.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант охорона праці _____ к.н., доцент Праховнік Н. А. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній
роботі немає запозичень з праць
інших авторів без відповідних
посилань.

Студент _____
(підпис)

Київ – 2015 року

Національний технічний університет України
«Київський політехнічний інститут»
Факультет інформатики та обчислювальної техніки
Кафедра технічної кібернетики

Освітньо-кваліфікаційний рівень «Магістр»
(назва ОКР)
Напрямок підготовки 6.050201 «Системна інженерія»
Спеціальність 8.05020102 “Комп’ютеризовані та робототехнічні системи”

ЗАТВЕРДЖУЮ
Завідувач кафедри
Ткач М. М.
(підпис) (ініціали, прізвище)
«__» _____ 20__ р.

ЗАВДАННЯ
на магістерську дисертацію студента
Кобилинського Богдана Андрійовича
(прізвище, ім’я, по батькові)

1. Тема роботи Дослідження оптимальної моделі розподіленої обробки
великих об’ємів даних
- Науковий керівник дисертації к.т.н., доцент Тимошин Ю.А.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
- затверджені наказом по університету від « 10 » березня 2015 р. № 292/2С
2. Строк подання студентом дисертації 02.06.2015 р.
3. Об’єкт дослідження Розподілене обчислювальне середовище
4. Предмет дослідження Алгоритми балансування навантаження
5. Перелік завдань які необхідно вирішити: Аналіз особливостей
розподіленої обробки даних; огляд критеріїв оптимальності системи
балансування навантаження; аналіз сучасних алгоритмів планування задач;
виділення варіантів покращення та усунення недоліків алгоритмів
розподілення задач; розробка емулятора Грід системи; розробка адаптивного
методу балансування навантаження; оцінка ефективності запропонованого
алгоритму.

6. Орієнтовний перелік ілюстративного матеріалу: Структурні схеми розподіленого середовища обробки даних; блок-схема алгоритму роботи Грід емулятора; графіки експериментальних досліджень; схема виробничого приміщення.

7. Орієнтовний перелік публікацій: Аналіз методів балансування навантаження у розподілених обчислювальних середовищах / Молода наука України. Перспективи та пріоритети розвитку. — Київ:ГО «ІОМП», 2015.

8. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1.Розділ охорони праці	Праховнік Н.А., доцент		
2.Нормоконтроль	Пасько В.П., доцент		

9. Дата видачі завдання 26.09.2014

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Вибір теми магістерської дисертації	22-24 вересня 2014 р.	
2.	Постановка задачі та розробка плану роботи	25 вересня – 17 жовтня 2014 р.	
3.	Дослідження предметної області системи	2–12 листопада 2014 р.	
4.	Розробка емулятора грід системи	20 листопада – 18 грудня 2014 р.	
5.	Аналіз існуючих методів балансування навантаження	21 січня – 10 лютого 2015р.	
6.	Розробка нового методу балансування навантаження	15 лютого -25 березня 2015р.	
7.	Оформлення пояснювальної записки до роботи	2 квітня – 15 травня 2015 р.	
8.	Оформлення розділу з охорони праці	16-30 травня 2015 р.	

Студент

(підпис)

(ініціали, прізвище)

Керівник роботи

(підпис)

(ініціали, прізвище)

ABSTRACT

The structure and the scope of work. Explanatory note of the degree project consists of 4 chapters, including 26 figures, 5 tables, 8 appendices and 15 sources.

Thesis is devoted to the research of optimal model of distributed processing of large volumes of data and the development of an effective method of load balancing between resources of distributed system.

Research should show an increase of efficiency in the distributed grid-systems by improving the load-balancing algorithm, the development of a number of requirements and allocation of the optimal model that will best accomplish the specified quality criteria.

This work will be conducted to:

1. Analysis of distributed computing.
2. Review of optimal criteria for load balancing system.
3. Analysis of modern planning algorithms.
4. Review of choices in improving deficiencies of distributed algorithms.
5. Development of grid-system emulator.
6. Development of adaptive load balancing algorithm.
7. Evaluating the effectiveness of the proposed algorithm.

Keywords: distributed computing, distributed processing, Grid, load balancing, tasks scheduling.

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломної роботи складається з 4 розділів, містить 26 рисунків, 5 таблиць, 8 додатків та 15 джерел.

Дипломна робота присвячена дослідженню оптимальної моделі розподіленої обробки великих об'ємів даних та розробці ефективного методу балансування навантаження між ресурсами розподіленої системи.

Дослідження має показати підвищення ефективності розподілення задач у grid-системах шляхом покращення алгоритму балансування навантаження на обчислювальні ресурси, а також розробка ряду вимог та виділення оптимальної моделі що буде максимально відповідати заданим критеріям якості.

У ході роботи буде проведено:

1. Аналіз особливостей розподіленої обробки даних.
2. Огляд критеріїв оптимальності системи балансування навантаження.
3. Аналіз сучасних алгоритмів планування задач.
4. Виділення варіантів покращення та усунення недоліків алгоритмів розподілення задач.
5. Розробка емулятора grid-системи.
6. Розробка адаптивного методу балансування навантаження.
7. Оцінка ефективності запропонованого алгоритму.

Ключові слова: розподілені обчислення, розподілена обробка даних, грід, балансування навантаження, планування задач.

**Пояснювальна записка
до магістерської дисертації**

на тему: «Дослідження оптимальної моделі розподіленої обробки
великих об'ємів даних»

Київ – 2015 року

ЗМІСТ

1. ОГЛЯД ТЕХНОЛОГІЇ РОЗПОДІЛЕНОЇ ОБРОБКИ ДАНИХ.....	12
1.1. Поняття розподілених систем та розподілених обчислень	12
1.2. Цілі побудови розподілених систем	16
1.3. Модель розподіленої обробки даних.....	18
1.4. Розподілена обробка даних у grid-системах.....	21
1.4.1. Grid-сервіси	23
1.4.2. Архітектура grid-системи	24
Висновки до розділу.....	28
2. АНАЛІЗ ПІДСИСТЕМИ ПЛАНУВАННЯ ЗАДАЧ ТА РОЗРОБКА ЕМУЛЯТОРА GRID-СИСТЕМИ	29
2.1. Процес планування задач.....	29
2.2. Підхід до аналізу алгоритмів розподілення задач	32
2.3. Оцінка ефективності планування	36
2.4. Симуляція grid-системи.....	37
2.5. Розробка системи для верифікації підсистеми планування задач.....	38
2.5.1. Вимоги до засобу моделювання роботи grid-системи.....	38
2.5.2. Опис розробленого емулятора grid-системи.....	40
2.6. Існуючі алгоритми планування задач у grid-системах	45
2.6.1. Стохастичний алгоритм балансування навантаження	45
2.6.2. Алгоритм балансування навантаження Round-Robin.....	46
2.6.3. Алгоритм мінімального часу виконання.....	47
2.6.4. Алгоритм найменшої черги.....	48
2.6.5. Алгоритм K -means.....	49
2.7. Перевірка роботи алгоритмів планування задач	50
Висновки до розділу.....	57
3. МОДЕЛЮВАННЯ ОПТИМАЛЬНОГО АЛГОРИТМУ ПЛАНУВАННЯ ЗАДАЧ ...	58
3.1. Варіанти покращення та виявлення впускених деталей у існуючих алгоритмах...	58
3.2. Розробка адаптивного алгоритму планування задач	60
3.3. Визначення оптимальних коефіцієнтів моделі	62
3.4. Порівняння запропонованого та існуючих алгоритмів на віртуальній системі.....	69
Висновки до розділу.....	72

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ	73
4.1. Характеристики приміщення	73
4.2. Аналіз шкідливих виробничих факторів.....	75
4.2.1. Мікrokлімат.....	75
4.2.2. Освітлення.....	77
4.2.3. Рівень шуму.....	78
4.2.4. Аналіз випромінювання.....	78
4.3. Електробезпека.....	79
4.4. Пожежна безпека.....	79
4.4.1. Категорія приміщення за вибухопожежною небезпекою.....	80
4.4.2. Пожежна сигналізація.....	81
4.4.3. Засоби гасіння пожежі.....	82
4.5. Техніка безпеки	82
4.6. План ліквідації надзвичайної ситуації	83
4.6.1. Можливі аварійні ситуації.....	83
4.6.2. Оцінка радіаційної обстановки в зоні радіаційного забруднення.....	85
4.7. Техніка безпеки після закінчення роботи.....	87
Висновки до розділу.....	87
ЗАГАЛЬНІ ВИСНОВКИ	88
ПЕРЕЛІК ПОСИЛАНЬ	90
ДОДАТОК А. Відомість магістерської дисертації	
ДОДАТОК Б. Лістинг коду розробленого емулятора grid-системи	
ДОДАТОК В. Архітектура обчислювального grid-середовища	
ДОДАТОК Г. Візуалізація алгоритмів балансування навантаження	
ДОДАТОК Д. Блок-схема алгоритму роботи емулятора grid-системи	
ДОДАТОК Е. Архітектура системи планування задач у grid-системах	
ДОДАТОК Ж. Графіки порівняння результатів роботи різних алгоритмів планування	
ДОДАТОК К. Діаграма класів розробленого емулятора grid-системи	

ВСТУП

Актуальність дослідження. Все більша кількість обчислювальних задач в різних областях, таких як наука, інженерія та бізнес важко піддаються обробці навіть на високопродуктивних компютерах. Справді, через їх розмір та складність вони повинні працювати з розподіленими моделями додатків. Тому у багатьох національних проектах при проектування архітектури був використаний концепт grid-обчислень, що дає можливість вирішувати широкий спектр задач. grid-системи дозволяють спільно використовувати обчислювальні ресурси які належать різним організаціям і які можуть бути розташовані в різних адміністративних регіонах. У таких системах можуть об'єднуватися різноманітні обчислювальні ресурси, такі як персональні комп'ютери, робочі станції, суперкомп'ютери та навіть цілі кластери. Такий підхід дає цілий ряд переваг як економічної точки зору, так з точки зору продуктивності і надійності.

Переваги з економічної точки зору обумовлюються тим, що вартість суперкомп'ютера або кластера, що здійснює централізовану обробку, на кілька порядків вище ніж вартість декількох десятків типових комп'ютерів.

Переваги з точки зору надійності організації обчислень впливають з того, що у разі відмови каналів зв'язку управління відповідною частиною об'єкта може здійснюватися локально.

Проте перехід до розподілених обчислень створює ряд нових організаційних проблем, відсутніх при централізованій обробці даних. Зокрема, до таких належать:

1. Постійне відстеження доступних обчислювальних ресурсів.
2. Організація обміну даними між обчислювальними ресурсами.
3. Забезпечення можливості асинхронного обміну даними.
4. Віддалений запуск паралельних процесів прикладної задачі.

5. Вибір обчислювальних ресурсів, на яких будуть запущені задачі.
6. Обробка помилкових ситуацій (відмов одного з комп'ютерів, що беруть участь в обчисленнях або каналу зв'язку з ним).
7. Збір інформації про хід виконання паралельних процесів завдання.
8. Облік ненадійності та пропускну здатності каналів зв'язку та обчислювальних ресурсів.

Мета дослідження. Метою даної дисертаційної роботи є підвищення ефективності обчислювальних систем для збереження та обробки великих об'ємів даних шляхом покращення алгоритму балансування навантаження на обчислювальні ресурси; а також розробка ряду вимог та виділення оптимальної моделі що буде максимально відповідати заданим критеріям якості. Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати особливості розподіленої обробки даних.
2. Виокремити критерії оптимальності системи балансування навантаження: швидкість обробки даних, час очікування задач у черзі, простій обладнання, масштабованість архітектури.
3. Проаналізувати сучасні алгоритми планування задач.
4. Виділити варіанти покращення та усунення недоліків алгоритмів розподілення задач.
5. Розробити емулятор grid-системи та оцінити ефективність запропонованого алгоритму.

Новизна дослідження. Наукова новизна даного дослідження полягає у новому підході до балансування навантаження на обчислювальні ресурси grid-систем. В результаті проведених досліджень було запропоновано новий алгоритм розподілення задач, що базується на існуючих алгоритмах та враховує гетерогенність, віддаленість ресурсів та їх поточну завантаженість.

Ефективність запропонованого методу доведена та перевірена практично на розробленому емуляторі grid-системи.

Об'єкт дослідження. Об'єктом дослідження є територіально-розподілена обчислювальне середовище – програмно-апаратний комплекс, що складається з безлічі різнорідних (гетерогенних) комп'ютерів, об'єднаних мережею передачі даних, та програмного забезпечення, яке робить можливим їх спільне використання для паралельного вирішення прикладних задач.

Предмет дослідження. Предметом дослідження є алгоритми балансування навантаження обчислювальних ресурсів у розподілених grid-системах, визначення особливостей їх застосування та ефективності їх застосування методів декомпозиції завдань. В якості основних критеріїв алгоритмів планування задач та оцінки їх ефективності використовуються загальний час вирішення задач, середній час очікування задач у черзі на виконання та простій обчислювальних ресурсів.

1. ОГЛЯД ТЕХНОЛОГІЇ РОЗПОДІЛЕНОЇ ОБРОБКИ ДАНИХ

В інформаційних системах однією з найважливіших технологій є розподілена обробка даних. Розташування персональних комп'ютерів на робочих місцях, тобто на місцях виникнення та використання інформації, дало можливість розподілити їх ресурси по окремих функціональних сферах діяльності та змінити технологію обробки даних у напрямку децентралізації. Розподілена обробка даних дозволяє підвищити швидкість вирішення масштабних задач та забезпечити гнучкість прийняття рішень. Перевагами розподіленої обробки даних є:

- збільшення числа віддалених взаємодіючих користувачів, що виконують функції збору, обробки, зберігання, передачі інформації;
- знятті пікових навантажень з централізованої бази шляхом розподілу обробки та зберігання локальних баз даних на різних ЕОМ;
- забезпечення доступу працівника до обчислювальних ресурсів мережі ЕОМ;
- забезпечення обміну даними між віддаленими користувачами [1].

1.1. Поняття розподілених систем та розподілених обчислень

Область розподілених обчислень (distributed computing) представляє собою розділ теорії обчислювальних систем, що вивчають теоретичні питання організації розподілених систем.

Також розподілені обчислення та обробку даних іноді визначають в більшу вузькому сенсі як застосування розподілених систем для вирішення великих ресурсомістких задач. В такому контексті розподілені обчислення є частковим випадком паралельних обчислень, тобто одночасного вирішення різних частин однієї задачі декількома обчислювальними пристроями. Також варто відмітити що при вивченні паралельних обчислень більше уваги варто

приділяти методам розподілення задачі на підзадачі, котрі можуть обчислюватися одночасно для максимізації швидкості обчислень. Основна особливість в організації паралельних обчислень з використанням розподілених систем буде полягати в необхідності враховувати відмінність характеристик доступних обчислювальних пристроїв в наявності існуючої часової затримки при обміні даними між ними.

В подальшому буде розглянуто розподілені обчислення в широкому сенсі, як теоретичну основу для побудови розподілених систем обробки даних.

Розглядаючи розподілену систему з апаратної точки зору в вигляді сукупності взаємопов'язаних автономних комп'ютерів або процесорів, з програмної точки зору – в вигляді сукупності незалежних процесів (виконувані програмні компоненти розподіленої системи), взаємодіючі шляхом передачі повідомлень для обміну даними та координації своїх дій. Комп'ютери, процесори або процеси можна вважати вузлами розподіленої системи. Щоб процесори змогли рахуватися автономними, вони повинні по меншій мірі володіти власним незалежним управлінням. По цій причині паралельний комп'ютер, архітектура якого побудована по схемі «одна команда для великої кількості даних», не може рахуватися розподіленою системою. Під незалежністю процесів розуміється той факт що кожен процес має свій власний стан, що представляється набором даних включаючий в себе поточні значення рахівника команд, регістрів та змінних, до яких процес може звертатися та які може змінювати. Стан кожного процесу є повністю закритим для інших процесів: інші процес не мають до нього прямого доступу та не можуть змінювати його.

Швидкості виконання операцій різних процесів в розподіленій системі різні та заздалегіть невідомі, а доставка відправлених повідомлень може займати непередбачуваний час.

Оскільки в якості вузлів системи можуть виступати процеси, під наведене визначення також підпадають програмні системи, що представляють собою сукупність взаємодіючих процесів, що виконуються на одному й тому ж самому обчислювальному пристрої. В цій ситуації канали взаємодії, що здійснюють передачу повідомлень між процесами, реалізуються за допомогою колективної пам'яті замість мережі зв'язку. Однак в більшості випадків в розподіленій системі все ж міститься декілька процесорів, пов'язаних між собою за допомогою різних способів комунікації. Типова розподілена система представлена на рис. 1.1.

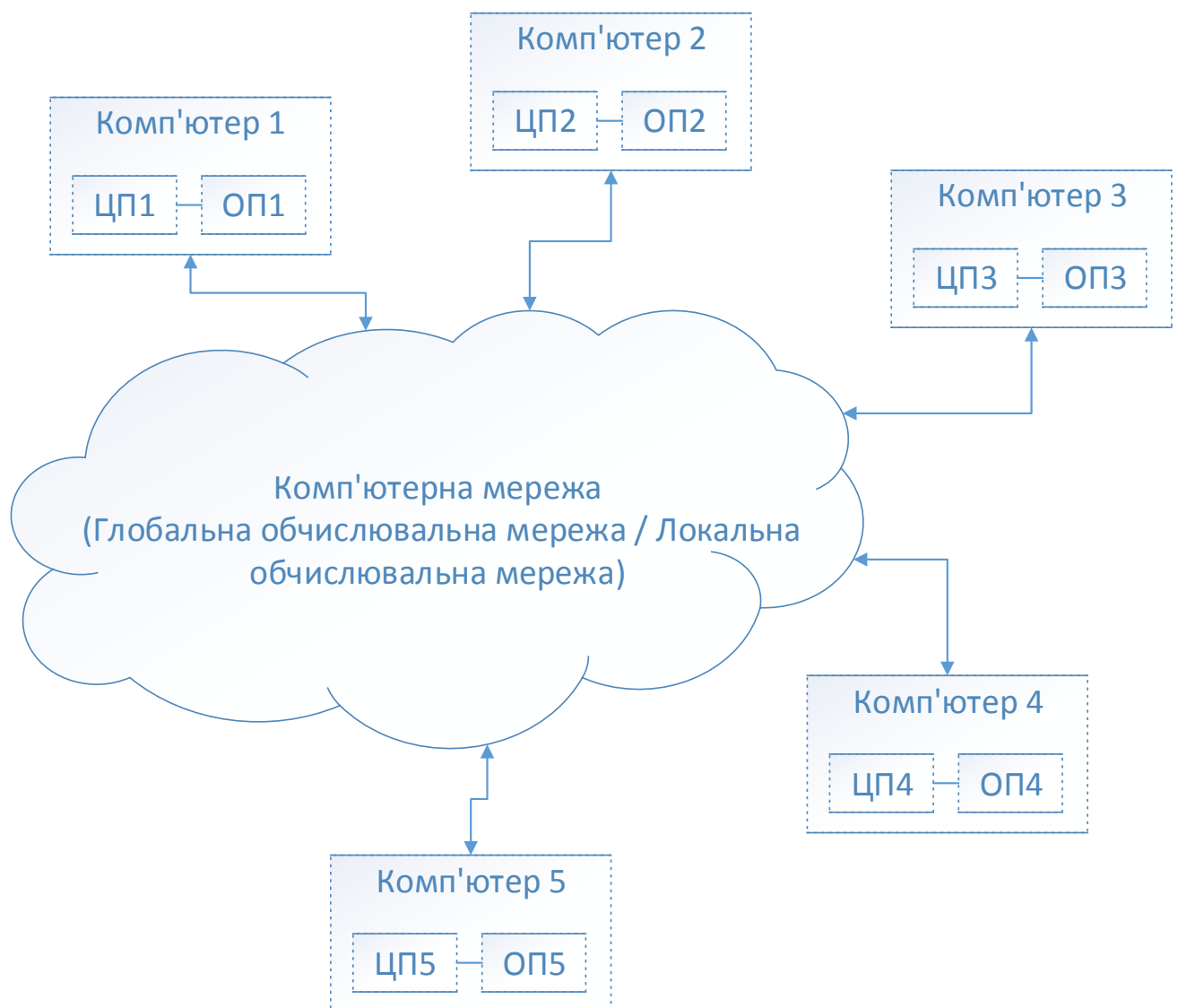


Рис. 1.1. Розподілена система об'єднує незалежні комп'ютери у мережу

Варто звернути увагу на основні відмінні ознаки, що характеризують розподілені системи. До таких ознак зазвичай відносять наступні.

- **Відсутність єдиного часу** для компонентів розподіленої системи. Це важливе припущення для вирішення задач проектування та побудови розподілених систем. Воно характеризує територіальне розподілення компонентів системи, а саме процесорів, що входять до її складу, але, що не менш важливе, з цього слідує відсутність синхронності в роботі.
- **Відсутність спільної пам'яті.** Це ключова характеристика, з якої слідує необхідність обміну повідомленнями між програмними компонентами розподіленої системи для їх взаємодії та синхронізації. Крім того, ця характеристика має на увазі відсутність єдиного для усіх процесорів фізичного часу. Варто відмітити що деякі розподілені системи можуть надавати своїм користувачам абстракцію єдиного адресного простору для усіх процесорів за допомогою механізмів розподіленої роздільної пам'яті. В цьому випадку, якщо не розглядати складність конкурентного доступу декількох процесорів до одного сегменту пам'яті, для кожного процесору розподілену роздільну пам'ять можна представити як цілком нормальну організацію віртуальної пам'яті, де в якості тимчасового сховища інформації використовується не власний диск, а оперативна пам'ять віддаленого комп'ютера. В зв'язку з цим, зазвичай посеред іншого розглядаються аспекти організації загальної пам'яті в багатопроцесорних системах.
- **Географічне розподілення.** Цілком природньо що чим більш віддалені процесори один від одного територіально, тим зрозуміліше що система буде розглядатися як розподілена. Однак зовсім не обов'язково щоб комп'ютери були об'єднані в глобальну обчислювальну мережу. Кластер зі звичайних робочих станцій, об'єднаних за допомогою

локальної обчислювальної мережі, також розглядається як невелика розподілена система. При цьому все обладнання такої розподіленої системи може знаходитися в одному або декількох сусідніх будівлі. Подібні кластери становляться все більш популярні через відносно низьку вартість вхідних компонентів з однієї сторони та непоганої продуктивності з іншої. Наприклад, ядро пошукової системи Google побудовано саме за цією архітектурою.

- **Незалежність та гетерогенність.** Комп'ютери, що входять до складу розподіленої системи слабо зв'язані в тому сенсі що вони можуть мати різний склад та різну продуктивність та, отже, забезпечувати різний час виконання ідентичних задач. Зазвичай вони не є частинами однієї спеціалізованої системи, але функціонують сумісно, надаючи свої служби одне одному для виконання спільної задачі. Більш того, в загальному випадку на комп'ютерах, що складають розподілену систему, можуть бути встановлені різні операційні системи.

1.2. Цілі побудови розподілених систем

За останнє десятиріччя розподілені системи ставали все більш популярними та їх роль тільки зростала. Серед основних причин зростання їх значимості можна виділити наступні.

- **Географічно розподілене обчислювальне середовище.** Сьогодні в більшості випадків обчислювальне середовище по своїй природі являє собою територіально розподілену систему. В якості прикладу можна провести банківську мережу. Кожен банк обслуговує рахунки своїх клієнтів та обробляє операції з ними. У випадку переводу коштів з одного банку в інший необхідно здійснити міжбанківську транзакцію та взаємодія систем банків одне з одним. Іншим прикладом географічно

розподіленим обчислювальним середовищем є усім відома мережа Інтернет.

- **Вимоги покращення продуктивності обчислень.** Швидкодія традиційних однопроцесорних систем стрімко наближається до своєї межі. Різноманітні архітектури (такі як: суперскалярна архітектура, матричні та векторні процесори, однокристальні багатопроцесорні системи) покликані покращити показники продуктивності обчислювальних систем за рахунок різноманітних механізмів паралельного виконання команд. Однак усі ці прийоми здатні покращити продуктивність максимум в десятки разів у порівнянні з класичними послідовними рішеннями. Окрім того, масштабованість подібних підходів залишає бажати кращого. Щоб покращити продуктивність в сотні або тисячі разів, та при цьому забезпечувати гідну масштабованість рішення, необхідно звести воедино процесори та забезпечити їх ефективну взаємодію. Цей процес реалізується у вигляді великих багатопроцесорних систем та багатомашинних комплексів.
- **Сумісне використання ресурсів.** Важливою метою створення та використання розподілених систем є надання користувачам (та додаткам) доступу до віддалених ресурсів та забезпечення їх самостійного використання. Наприклад, користувач Комп'ютера 1 може використовувати дисковий простір Комп'ютера 2 для зберігання своїх файлів. Або додаток А може використовувати вільну обчислювальну потужність деяких комп'ютерів для прискорення власних розрахунків. Розподілені бази даних та розподілені системи об'єктів можуть бути відмінним прикладом сумісного використання програмних компонентів, коли відповідні програмні абстракції розподілені по декільком

комп'ютерам та узгоджено обслуговуються декількома процесорами, що утворюють розподілену систему.

- **Відмовостійкість.** В традиційних «нерозподілених» обчислювальних системах, побудованих на базі одиничного комп'ютера (можливо високопродуктивного), вихід із ладу одного з його компонентів зазвичай приводить до непрацездатності усієї системи. Такий збій в одному або декількох компонентах системи називають частковою відмовою, якщо він не зачіпає інші компоненти. Характерною рисою розподілених систем, котра відрізняє їх від одиничних комп'ютерів, є стійкість до часткових відмов, тобто система повинна продовжувати функціонувати після часткових відмов, правда, незначним чином зменшуючи при цьому загальну продуктивність. Подібна можливість досягається за рахунок збитковості, коли у систему додається додаткове обладнання (апаратна збитковість) або процеси (програмна збитковість), котрі роблять можливим правильне функціонування системи при непрацездатності або некоректної роботи деяких з її компонентів. В цьому випадку розподілена система намагається сховати факти відмов або помилок у одних процесах від інших процесів [2].

1.3. Модель розподіленої обробки даних

Отже, суть розподіленої обробки даних полягає в тому, що користувач отримує можливість працювати з мережевими службами та прикладними процесами, розташованими в декількох віддалених взаємопов'язаних абонентських системах. При цьому можливі декілька видів задач, які він може вирішувати:

- віддалений запит, наприклад, команда, що дозволяє посилати одиночну заявку на виконання обробки даних;
- віддалена транзакція, що здійснює напрям групи запитів прикладному процесу;
- розподілена транзакція, що дає можливість використання декількох серверів і прикладних процесів, які виконуються в групі абонентських систем.

Для розподіленої обробки здійснюється сегментація прикладних програм, тобто поділ складної прикладної програми на частини, які можуть бути розподілені по системах локальної мережі. Сегментація здійснюється за допомогою спеціального інструментального програмного забезпечення, яке автоматизує даний процес. В результаті виконання зазначеного процесу прикладна програма ділиться на самостійні частини, що завантажуються на різні віддалені ресурси системи. Завдяки цьому, створюється можливість переміщення програм з однієї системи в іншу і розподіленої обробки даних. В результаті сегментації кожна виділена частина програми включає управління даними, алгоритм і блок презентації. Завдяки цьому, вона може бути оптимальним чином виконана на основі платформ, що використовуються в мережі. Передача даних для розподіленої обробки відбувається за допомогою віддаленого виклику процедур або електронної пошти. Перша технологія характеризується високою швидкістю, а друга – низькою вартістю. Віддалений виклик процедур працює аналогічно місцевим викликом процедур і забезпечує організацію обробки даних. Цій меті служить механізм навігації в мережі, пошуку інформації, запуску процесу в декількох системах, передачі отриманих результатів користувачам, які посилали запити. Цей процес характеризується прозорістю, завдяки якій об'єкти мережі, розташовані між користувачами і програмами не помітні обом партнерам. Виконання

віддаленого виклику процедур є дорогою операцією бо, на весь час її виконання, усі системи що беруть участь в роботі повинні передавати дані один одному по каналах зв'язку. Альтернативною віддаленого виклику є застосування інтелектуальних агентів або виконання розподіленої обробки даних з використанням електронної пошти. Цей метод не вимагає великих витрат, але працює значно повільніше.

При плануванні обробки даних можуть розглядатися три моделі:

- обробка у одноранговій локальній мережі;
- централізована обробка;
- обробка в моделі клієнт/сервер.

При будь-якій обробці є три основних рівня маніпулювання даними:

- зберігання даних;
- виконання додатків, тобто вибірка та обробка даних для потреб прикладної задачі;
- представлення даних і результатів обробки кінцевому користувачу.

При обробці в одноранговій мережі всі три рівні, як правило, виконуються на одному (персональному) робочому місці. У сучасних технологіях застосування обчислювальної техніки персональна обробка інформації, коли всі дані і засоби їх обробки зосереджені в межах одного робочого місця, і обмін даними між робочими місцями не відбувається або виконується епізодично (наприклад, засобами електронної пошти), поступово відходить у минуле. Сучасні інформаційні, управлінські та офісні системи більшою чи меншою мірою орієнтуються на багатокористувацьку обробку, при якій дані доступні багатьом користувачам з різних робочих місць. Поняття ефективності та надійності вимагають централізації процесів зберігання і обробки даних.

І централізована обробка, і модель клієнт / сервер в рівній мірі використовують переваги централізації. Різниця між цими двома моделями полягає в тому, що при централізованій обробці подання інформації кінцевому користувачу також виконується засобами центральної обчислювальної системи – на її терміналах, підключених до обчислювальної системи через канали введення-виведення. У моделі клієнт / сервер навпаки: термінали, що представляють інформацію, є інтелектуальними – самостійними обчислювальними системами (зазвичай персональними комп'ютерами) і пов'язані з сервером через мережеві засоби.

Хоча централізована обробка забезпечує більшу ефективність у супроводі системи і в швидкості обміну, кращою все ж видається модель клієнт / сервер, до переваг якої слід віднести насамперед гнучкість – можливість будувати клієнтські робочі місця на різних платформах та на різних операційних середовищах, таким чином, гнучко пристосовуючи можливості інтелектуального терміналу, що стоять перед даним робочим місцем [3].

1.4. Розподілена обробка даних у grid-системах

Наявність потужних комп'ютерів і високошвидкісних мережевих технологій призвели до вирішення більш масштабних великих проблем. Це рішення відоме нам як grid-обчислення. grid-системи дозволяють зберігати, оброблювати та збирати інформацію на різноманітних розподілених ресурсах, таких як суперкомп'ютери, ресурси зберігання, джерела даних та інші спеціалізовані пристрої. Ресурси grid-систем можуть бути географічно розподілені, належати різним організаціям, і використовуватися для вирішення масштабних проблем в галузі науки, техніки, торгівлі та інших різноманітних областях. Концепція grid-обчислень виникла як проект що географічно зв'язує суперкомп'ютери, але тепер вона виросла далеко за межі

своїї первісної мети. grid-інфраструктура може принести користь багатьом додаткам, у тому числі для спільного проектування, дослідження даних та високопродуктивних обчислень.

Компоненти найвищого рівня grid-інфраструктури пов'язані в єдиному інтегрованому обчислювальному grid-середовищі, що показано на рис. 1.2. В системі grid завдання являють собою атомарну одиницю що запланована та призначена на конкретний ресурс, в той час як додаток складається з безлічі таких атомарних задач, які виконуються на безлічі ресурсів. Кінцеві користувачі взаємодіють з брокером ресурсів, який виконує виявлення ресурсів, планування і моніторинг задач по розподіленим grid-ресурсам. Для того щоб надати користувачам єдине обчислювальне середовище, необхідно вирішити кілька проблем. Однією з основних проблем є різноманітність в grid-середовищі, що призводить від безлічі різноманітних ресурсів та широкого спектру охоплених технологій. Ще одна проблема торкається питання автономності у зв'язку з географічно розподіленими grid-ресурсами по декількох адміністративних доменах, що належать різним організаціям. Інші проблеми включають в себе масштабованість (зазвичай спостерігається потенційна деградація продуктивності, оскільки розмір сітки збільшується) і динамічність / адаптація ресурсів. Програмне забезпечення grid-системи повинне динамічно адаптуватися та ефективно використовувати усі наявні ресурси та послуги.

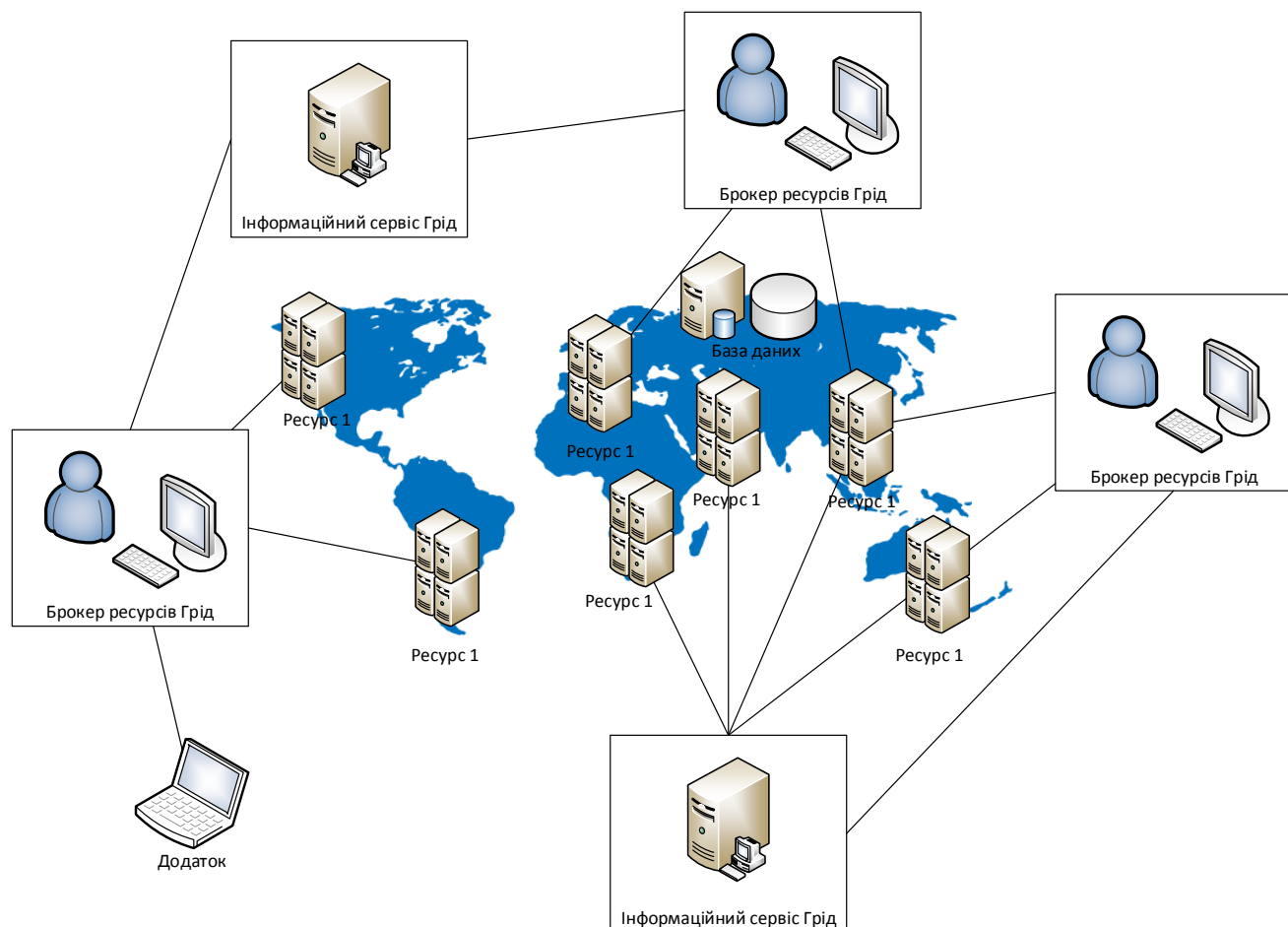


Рис. 1.2. Обчислювальне середовище grid

1.4.1. Grid-сервіси

З точки зору кінцевого користувача, grid-системи можуть використовуватися для забезпечення різних видів послуг. Найбільш популярними сервісами є наведені нижче.

- **Обчислювальні сервіси (Computational services):** надавання (часто захищене) послуг для виконання великомасштабних та ресурсномістких завдань на розподілених обчислювальних ресурсах (індивідуально або колективно). grid-систему, що забезпечує обчислювальні послуги часто називають обчислювальною grid-системою.

- **Сервіси обробки та передачі даних (Data services):** гарантують безпечне зберігання та управління доступом до розподілених масивів даних. Для забезпечення масштабованого зберігання і доступу, набори даних можуть бути відтворені, каталогізовані та навіть збережені в різних місцях. Обробка наборів даних здійснюється з використанням послуг обчислювальної grid-системи (Computational Grid), в результаті чого в поєднанні ці системи зазвичай називають grid-системою даних (Data Grid) [5].

1.4.2. Архітектура grid-системи

На рис. 1.3 зображено розподілену архітектуру grid-системи. Ця архітектура є достатньо повною та підходить під опис різних моделей, що використовуються для розподілення та управління ресурсами. Основними компонентами grid-системи є:

- користувач grid-системи та додатки (дослідницькі, інженерні та ін.);
- інтерфейсне програмне забезпечення рівня користувача (високорівневі послуги та інструменти;
- програмні оточення;
- брокери ресурсів;
- інтерфейсне програмне забезпечення ядра grid-системи (розподілення ресурсів та об'єднання розподілених ресурсів);
- постачальники ресурсів та послуг grid.

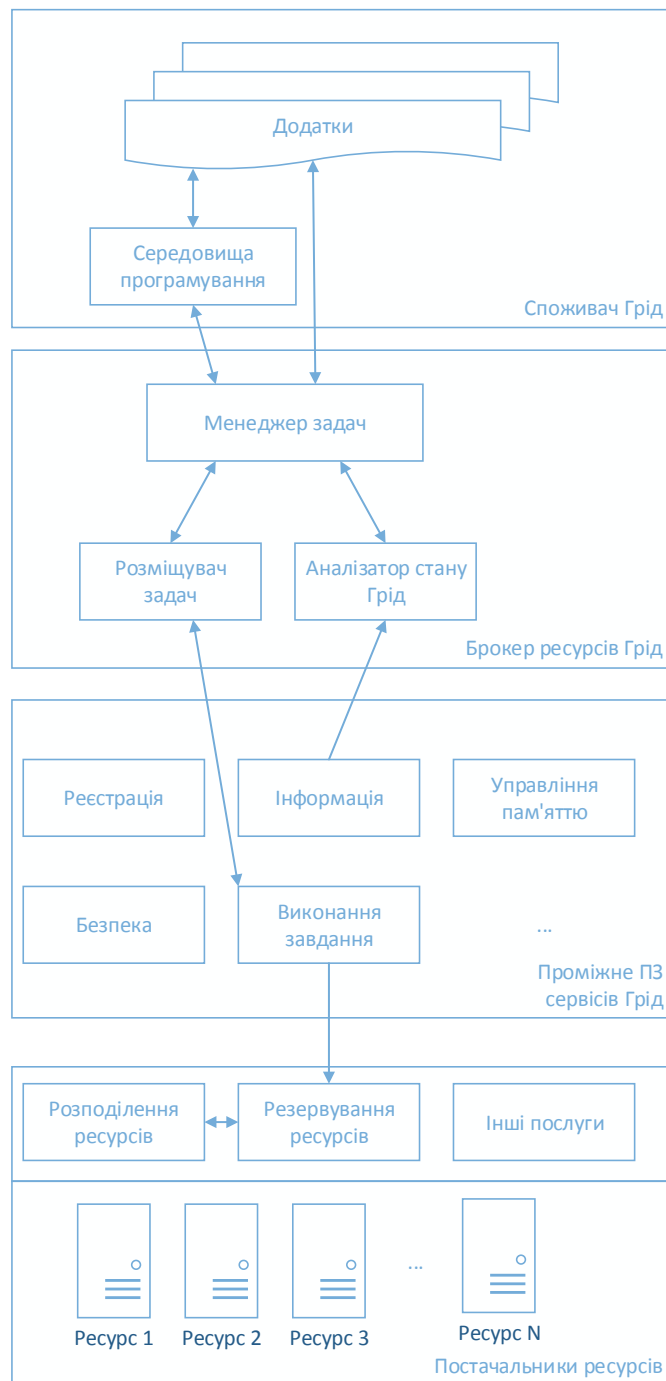


Рис. 1.3. Архітектура grid-системи

Двома головними суб'єктами в системі grid є споживачі та постачальники ресурсів. Обидва суб'єкти мають свої власні стратегії. Споживачі ресурсів застосовують стратегії вирішення своїх прикладних задач в залежності від необхідного часу та наявного бюджету. Постачальники ресурсів використовують стратегію отримання найбільшої вигоди від ресурсів

на які було вкладено кошти. Власники таких ресурсів намагаються максимізувати використання своїх ресурсів [4].

Користувачі grid-системи мають можливість вибору саме тих постачальників, які більшою мірою задовольняють їх потребам та вимогам. Якщо постачальники ресурсів мають локальних користувачів, то вони будуть намагатися обслуговувати їх в першу чергу, надаючи більший пріоритет їх завданням. Для цього grid-системи повинні використовувати утиліти та механізми, які дозволяють постачальникам і споживачам задавати свої вимоги.

Споживачі ресурсів grid взаємодіють з брокерами для задання своїх вимог. Це може бути бюджет, або сума, яку вони можуть витратити на вирішення даного завдання та межа часу протягом якого вони хочуть отримати результат. Вони також повинні мати можливість вибирати між цими двома групами вимог. Постачальники ресурсів потребують утиліти для задання цінових політик та механізмів, які допоможуть їм максимізувати прибуток та утилізацію ресурсів.

Брокер ресурсів grid виступає в ролі проміжної ланки між користувачами і постачальниками ресурсів grid, що сполучає їх використовуючи інтерфейсне ПЗ ядра для забезпечення послуг. Він відповідає за пошук ресурсів, їх вибір, прив'язку програмного забезпечення і даних до обраних ресурсів, ініціалізацію обчислень, адаптацію до змін в системі. Брокер виступає для користувачів grid як єдиний ресурс та складається з наступних компонентів.

- **Система управління завданнями.** Відповідає за проходження завдання користувача. Взаємодіє з планувальником для генерації розкладу, проводить процес створення завдання, публікацію завдання менеджеру ресурсів, підтримку статусу завдання, взаємодію з клієнтами та користувачами, планувальником та розподільником.
- **Система складання розкладів.** Відповідає за пошук ресурсів (використовуючи систему дослідження grid), вибір ресурсу та розподіл завдання (генерація розкладу), щоб переконатися в тому, що вимоги користувача задоволені.
- **Система дослідження grid.** Відповідає за виявлення ресурсів, взаємодіючи з інформаційним сервером grid.
- **Система встановлення завдання.** Відповідає за активацію виконання завдання на вибраному ресурсі та періодичне оновлення статусу виконуваного завдання.

Базовий рівень складається з розподілених ресурсів, таких як комп'ютери, мережеві пристрої, пристрої зберігання даних та інших спеціалізованих пристроїв та інструментів. Обчислювальні ресурси в сукупності представляють собою різні архітектури, такі як: кластери, суперкомп'ютери, сервери та звичайні ПК. Вони можуть працювати з різними операційними системами (наприклад, UNIX або Windows). Різноманітні наукові інструменти, такі як сенсори, датчики, детектори надають дані в режимі реального часу, що можуть бути передані безпосередньо на обчислювальні ресурси або зберігатися в базі даних. Основний проміжний рівень grid-системи надає різноманітні послуги, такі як дистанційне управління процесами, розподіл ресурсів, доступ до сховищ даних, реєстрації, доступу, безпеки інформації. Основні сервіси grid-системи абстрагують

складність та неоднорідність базового рівня, забезпечуючи послідовний метод доступу для розподілених ресурсів [5].

Висновки до розділу

1. В даному розділі визначено поняття розподілених систем, їх необхідність у нашому житті, цілі їх побудови та актуальність проблеми.
2. Розглянуто модель розподіленої обробки даних у таких системах, розглянуто види задач що можуть вирішуватися за допомогою grid-систем та було наведено загальну архітектуру grid-систем, описано основні функціональні компоненти.
3. Показано що слабким місцем систем розподілених обчислень є насамперед планувальники задач що можуть більш оптимально балансувати навантаження на доступні ресурси.
4. Визначено що необхідно більш детально проаналізувати алгоритми балансування навантаження на ресурси для визначення шляхів покращення ефективності розподілених систем.

2. АНАЛІЗ ПІДСИСТЕМИ ПЛАНУВАННЯ ЗАДАЧ ТА РОЗРОБКА ЕМУЛЯТОРА GRID-СИСТЕМИ

Інновації в обчислювальних та мережевих технологіях призвели до появи обчислювальних grid-систем. Планування завдань є невід'ємною частиною розподіленої таких обчислювальних систем та їх алгоритми повинні адекватно створювати відповідність поставленого завдання доступному ресурсу. При плануванні враховується декілька критеріїв декілька критеріїв, роблячи дану задачу многочленною в загальному її формулюванні. Планування в grid-системах є NP-повною задачею оптимізації, що використовує різноманітні типи та параметри системної інформації для створення розумного рішення планування.

Не існує єдиного алгоритму планування задач в grid-системах, який би задовольняв усі потреби та був ефективним на всіх grid-системах з різними додатками, інструментарієм та функціями. У цьому розділі вирішується проблема планування в grid-системах, в яких ресурси широко розповсюджені, в результаті чого необхідно враховувати також значну вартість зв'язку між доступними ресурсами [6].

2.1. Процес планування задач

Як правило, процес планування задач або процесів можна розділити на три етапи:

1. пошук ресурсів та їх фільтрація;
2. відбір ресурсів та планування відповідно до певних заданих цілей;
3. запуск процесу.

Рис. 2.1 показує модель планування задач, в якій функціональні компоненти з'єднані двома типами потоків даних: потік інформації про ресурс або додаток (ламані лінії) та потік задачі (суцільні лінії).

Процес планування задач у grid-системах включає в себе наступні кроки: отримання певної задачі від користувачів, проведення відбору можливих та доступних ресурсів для виконання цих завдань відповідно до отриманої інформації з модуля Інформаційної Служби Grid (Grid Information Service), та, нарешті, генерації співставлень завдання-ресурс на основі певних об'єктивних функцій та передбачень продуктивності ресурсів. Рис. 2.1 показує один планувальник задач, в той час як на практиці може бути розгорнуто таку архітектуру, яка може включати в себе декілька таких планувальників, які можуть організовувати різні структури (централізовані, ієрархічні ті децентралізовані) у відповідності з різними проблемами, такими, як продуктивність та/або масштабованість [7].

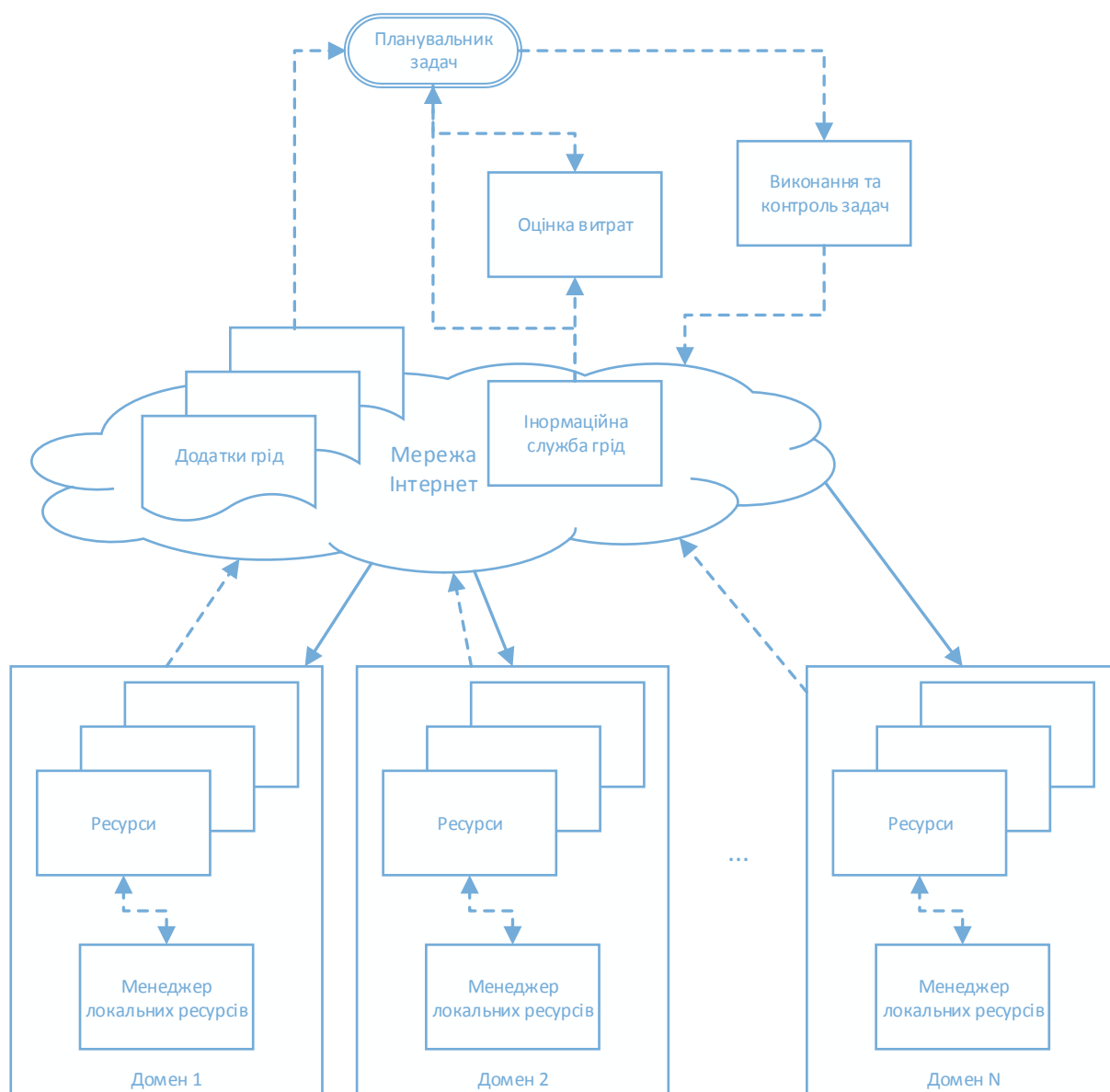


Рис. 2.1. Логічна архітектура планування задач у grid-системах

Інформація про статуси доступних обчислювальних ресурсів є важливим фактором для прийняття рішень щодо планування задач. Зокрема, дана інформація повинна враховуватися коли grid-система носить неоднорідний та динамічний характер. Інформаційна Служба grid несе відповідальність за надання такої інформації для grid-планувальників. Вона збирає та прогнозує інформацію про стан ресурсів, наприклад, потужності процесора, обсяг пам'яті, пропускну здатність мережі, готовність програмного забезпечення та навантаження ресурсів. Для прийняття рішення на додаток до

інформації про стан ресурсів з ІСГ, враховуються властивості задачі (наприклад: вимоги, приблизна кількість команд, необхідна кількість пам'яті та місця на жорсткому диску, залежність від інших задач та ін.). Модуль оцінки вартості обчислює вартість всіх ресурсів-кандидатів та формує оптимальний список ресурсів відповідно до поточної задачі. Розраховані витрати використовуються планувальником для вибору ресурсу що задовольняє бажані показники продуктивності. Модуль контролю та запуску задач реалізує рішення планування шляхом запуску завдання на обраному ресурсі, постановку вхідних даних, та контроль виконання завдання. Менеджер Локальних Ресурсів (Local Resource Manager) в основному відповідає за два завдання: локальне планування задач, що надійшли як ззовні, так і з локальної мережі, на ресурсах всередині домену. В домені один або декілька локальних планувальників повинні працювати згідно локальної стратегії управління ресурсами. МЛР також збирає інформацію з місцевих ресурсів та надсилає результати до ІСГ [6].

2.2. Підхід до аналізу алгоритмів розподілення задач

Алгоритми розподілення навантаження можна розділити на три типи: статичні, динамічна та адаптивні.

Динамічні алгоритми розподілення навантаження використовують інформацію про стан системи, наприклад, про завантаження вузлів, для прийняття рішень про розподілення навантаження в процесі роботи системи. Статичні алгоритми таку інформацію не використовують. В статичних алгоритмах рішення про розподілення навантаження реалізовані заздалегідь, до початку роботи системи, та не змінюються в процесі роботи. Прикладом статичного алгоритму може слугувати алгоритм Round Robin (описаний у розділі 2), що розподіляє задачі між вузлами по черзі. Динамічні алгоритми

завдяки використанню інформації про стан системи мають можливість покращити ефективність статичних алгоритмів. З іншого боку, внаслідок того що динамічні алгоритми повинні збирати, зберігати та аналізувати інформацію про стан системи, їм властиві великі накладні витрати на балансування в порівнянні зі статичними алгоритмами. Адаптивні алгоритми походять від динамічних алгоритмів та в залежності від умов та апіорного знання про властивості алгоритмів балансування, вибирають той з них, що найбільш підходить до поставленого завдання. Інформація про стан системи є важливим елементом роботи динамічного алгоритму. Існують різноманітні методи збору даних про завантаженість системи та побудови характеристик завантаженості. Завантаженість вузла може визначатися, наприклад, як частина процесорного часу, витраченого на вирішення задачі, або ж як ймовірність виявити вузол зайнятим в деякий момент часу.

По відношенню до початку передачі задач визначають витісняючі та невитісняючі алгоритми балансування. Витісняючі алгоритми можуть передавати задачі, вирішені частково, в той час як невитісняючі можуть розподіляти тільки задачі, виконання яких ще не почато. Прикладом витісняючого алгоритму може слугувати алгоритм планування процесі у системі Unix. Динамічні алгоритми балансування поділяються ступенем централізованості. Алгоритми можуть бути повністю централізованими, ієрархічними, повністю розподіленими або поєднувати вказані підходи. Централізовані алгоритми потенційно менш надійні, ніж повністю розподілені, так як збій центрального компоненту приведе до неполадок у всій системі. Вирішення даної проблеми може слугувати підтримка надлишкової кількості компонентів що дублюють один одного. Інша слабка сторона централізованих алгоритмів полягає в тому що центральний компонент представляє собою можливе слабе місце у всій системі. В той час як

ієрархічні алгоритми дозволяють частково вирішити дві проблеми, повне вирішення досягається тільки розподіленими алгоритмами. З іншого боку, централізовані алгоритми легше контролювати та вони є більш легкими у підтримці [8].

Зазвичай динамічні алгоритми балансування містять в собі 4 елементи:

- політика (правила) балансування;
- алгоритм вибору партнера;
- алгоритм вибору задачі для передачі;
- механізм збору необхідної інформації про стан системи.

Політика балансування визначає, чи є вузол об'єктом балансування. Різноманітні види політик балансування використовують порогові значення завантаженості вузлів, відхиленні значення навантаження на вузол від середнього значення по системі та інші методи.

Алгоритм вибору задачі визначає, яку задачу необхідно передати. При виборі задачі для відправки алгоритм може враховувати що накладні витрати, пов'язані з пересилкою задачі, повинні бути мінімальними, складність виконання задачі повинна бути великою, кількість зв'язків в задачі з локальними ресурсами повинно також бути мінімальним.

Алгоритм вибору партнеру відповідає за вибір підходящого вузла для операції балансування. Розповсюдженою технікою в розподілених алгоритмах є опитування вузлів. Опитування може бути послідовним або паралельним, використовувати результати попередніх опитувань. В централізованих алгоритмах вузол звертається до спеціалізованого координатора для визначення підходящого партнера для балансування. Координатор збирає та підтримує в актуальному становищу інформацію про завантаженість вузлів системи, а алгоритм пошуку партнеру використовує ці дані.

Механізм збору інформації про завантаженість системи визначає, коли збираються дані про завантаженості, що є джерелом інформації та де зберігається інформація. Виділяють декілька класів механізмів збору інформації, наведених нижче.

1. **Збір даних по необхідності.** Розподілені алгоритми даного класу збирають інформацію про завантаженість коли вузол потребує балансування навантаження. Вони підрозділяються на ініційовані відправником, отримувачем та симетрично ініційовані. В алгоритмах, ініційованих відправником, вузол що передає задачі, шукає користувачів яким він може передати частину задач. В алгоритмах, ініційованих отримувачем, вузол що отримує задачі, запозичує задачі у отримувача. В симетричних алгоритмах використовується комбінація вказаних підходів.
2. **Періодичний збір даних.** Алгоритми даного класу можуть бути як централізованими, так і розподіленими. В залежності від зібраних задач, алгоритм ініціює балансування навантаження.
3. **Збір даних по стану системи.** В системах що реалізують алгоритми даного класу, вузли самі розповсюджують інформацію про зміну завантаження при зміні внутрішнього стану. У випадку розподілених алгоритмів дані направляються сусіднім вузлам, а у випадку централізованих алгоритмів дані направляються координатору.

Під стабільністю алгоритму балансування або системи загалом зазвичай розуміють одну з двох характеристик:

- **системна стійкість**, що має на меті недопустимість ситуації коли окремі вузли системи перевантажені, в той час як інші простоюють або недовантажені;
- **алгоритмічна стабільність**, що виражається в тому що алгоритм не здійснює безкорисних дій, наприклад, не передає задачу по колу між вузлами так, що задача ніколи не виконується.

2.3. Оцінка ефективності планування

Розглянемо типові показники продуктивності grid-системи і методи та інструментарії що використовуються для оцінки алгоритмів планування задач.

Існує ряд ключових показників ефективності, значення яких залежить як від самої grid-системи, так і додатків що виконуються. Виокремимо наступні показники, де перші два використовуються для алгоритмів обчислювальних grid-систем, а grid-системи обробки даних оцінюються з використанням всіх трьох показників:

- максимальний час завершення всіх задач;
- середній час завершення всіх доступних задач;
- доступність даних, що вимірюється часом доступу ресурсу до даних, необхідних для даної задачі. Ця оцінка показує наскільки близько обчислювальний ресурс знаходиться до ресурсу, який зберігає необхідні дані. Позначимо $a_{l,j}$ доступністю даних для задачі l , коли його обчислення ресурс є j . Отже, цей показник обчислюється як:

$$a_{l,j} = \frac{t_{l,j}}{d_l},$$

де d_l – кількість даних, необхідних для задачі l (наприклад, у МБайтах), $t_{l,j}$ – час, який ресурс потребує для того щоб витягнути усю необхідну інформацію для задачі l з відповідного віддаленого ресурсу-сховища.

Якість алгоритму планувальника в виборі ресурсу для запуску ресурсномістких задач може бути оцінена на основі середньої доступності усіх ресурсів з-поміж усіх задач:

$$a = \frac{\sum_{j=1}^M \sum_{l=1}^{n^t} a_{l,j}}{n^t},$$

де n^t – загальна кількість запущених задач,

M – загальна кількість ресурсів у системі.

2.4. Симуляція grid-системи

Повномасштабна оцінка тестової grid-системи може потребувати значних навантажень та великої кількості ресурсів та обладнання. Крім того, важко оцінити стратегії реплікації і алгоритми планування. Таким чином, необхідно запровадити інший метод для тестування алгоритмів планування у grid-системі з навантаженням, близьким до реального.

Симуляція grid-системи, мабуть, є потенційним способом проаналізувати алгоритми планування у великих масштабах. На відміну від використання реальної системи в режимі реального часу, моделювання дозволяє уникнути накладних витрат ресурсів. Тому це не додає непотрібну складність до механізму аналізу. Моделювання є також ефективним у роботі з дуже великими задачами, які вимагають участі великої кількості активних користувачів та ресурсів. Ці та інші вимоги призвели до необхідності розробки grid-симуляторів.

Одним з найбільш поширених в даний час програмних засобів для реалізації grid-системи є пакет Globus Toolkit. Пакет Globus Toolkit розробляється, підтримується та поширюється міжнародним альянсом розробників з університетів США та Великобританії, а також наукових лабораторій та обчислювальних центрів. Globus Toolkit є вільно поширюваним з відкритим вихідним кодом програмним пакетом та пропонує базові засоби

для створення grid-інфраструктури: засоби забезпечення безпеки в розподіленому середовищі, засоби надійної передачі великих обсягів даних, засоби запуску та отримання результатів виконання завдань на віддалених обчислювальних ресурсах. На базі пакету Globus Toolkit створюються промислові версії реалізацій grid-інфраструктури, наприклад, такі як Univa та Platform Globus Toolkit.

Однак, незважаючи на те, що вже зараз пропонуються засоби створення grid-інфраструктури, що стали "де-факто" стандартними, існує ряд важливих наукових завдань, у тому числі і теоретичних, без вирішення яких повномасштабне використання можливостей Grid-технологій в промисловості неможливо. Оптимізація алгоритмів керування розподіленим середовищем на безпосередньо вже існуючої Grid-інфраструктурі є нетривіальною задачею та пов'язано зі значними витратами та простоями ресурсних центрів, а часто в силу масштабності розподіленого середовища взагалі не можливо. У зв'язку з цим, актуальним завданням є створення системи моделювання grid-інфраструктури, яка дозволить адекватно оцінювати її поведінку при змінних умовах і, на основі цього, оптимізувати стратегії управління потоками задач.

2.5. Розробка системи для верифікації підсистеми планування задач

2.5.1. Вимоги до засобу моделювання роботи grid-системи

В першу чергу слід відмітити необхідність розробки емулятора grid-системи що був б максимально подібний до реальної grid-системи, був гнучким у конфігурації та враховувала би усі деталі та можливості актуальних grid-систем. Тому наведемо ряд вимог до емулятора grid-системи що буде використовуватися для верифікації алгоритмів планування задач.

1. Симулятор grid-системи повинен забезпечувати гнучкий та легкий у застосуванні механізм контролю ресурсів grid-середовища.
2. Необхідно контролювати кількість задач та обчислювальних ресурсів.
3. У зв'язку з гетерогенністю (різнорідністю) справжніх grid-систем, потрібно врахувати наявність обчислювальних ресурсів з різною продуктивністю. Тому необхідно запровадити можливість задання коефіцієнту продуктивності кожного ресурсу, що буде явно впливати на час задач.
4. Через можливу високий рівень розподіленості реальної grid-системи, існує проблема з часом доступу як до обчислювального ресурсу, так і до ресурсу де зберігаються дані, необхідні для задачі. Отже потрібно врахувати можливість конфігурування параметрів розподіленості ресурсів, що включає в себе час доступу до обчислювального ресурсу та час доступу до сховища даних (що буде унікальним значенням для кожної пари ресурс-задача).
5. Для верифікації оптимального розподілу задач по ресурсам, потрібно врахувати ряд параметрів, що були зазначені вище. Серед них важливим параметром є простій ресурсу, тобто час, який ресурс знаходиться в стані очікування нових задач. Даним параметром не слід нехтувати, тому він також потрібен бути врахований при моделюванні grid-системи.
6. Найбільш важливим питанням є необхідність запровадження можливості конфігурування потоку задач у симуляторі grid-системи. Тому потрібно розробити просту можливість формування потоку обчислювальних задач, що характеризується швидкістю їх подачі у загальну чергу.

7. Кожна задача має унікальний час виконання на кожному ресурсі, тому необхідно врахувати відхилення цього часу від первісного. Це може бути реалізовано шляхом зазначення коефіцієнту продуктивності ресурсу (зазначеного у п.3.).
8. Для перевірки часу очікування кожної задачі у черзі необхідно обчислювати час з моменту надходження задачі у чергу до моменту початку її виконання.
9. Для порівняння різних алгоритмів планування задач у grid-системі повинна існувати можливість задання правил (алгоритмів) розподілення задач по ресурсам та конфігурування параметрів кожного алгоритму.

2.5.2. Опис розробленого емулятора grid-системи

Отже, у рамках даної дисертації був розроблений емулятор grid-системи, що відповідає усім вищенаведеним вимогам.

Програмний продукт було розроблено на мові програмування Java у середовищі IntelliJ IDEA. Вихідний код наведений у Додатку Б. Розглянемо детальніше блок-схему алгоритму роботи емулятора. Загальний алгоритм роботи емулятора grid-системи наведений на рис. 2.2.

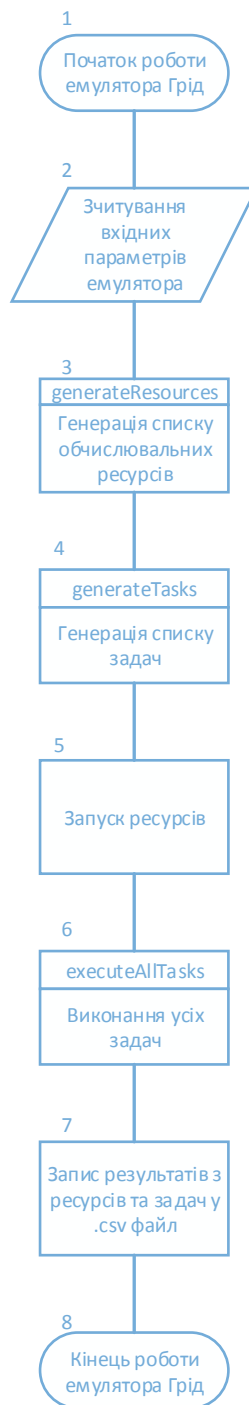


Рис. 2.2. Загальний алгоритм роботи емулятора grid-системи

В першу чергу відбувається зчитування вхідних параметрів grid-системи, що включають в себе:

- кількість задач;
- кількість обчислювальних ресурсів;

- середній час виконання задачі та допустиме відхилення цього часу (емулятор згенерує випадкове значення в заданих межах);
- максимально можливий час доступу від планувальника задач до обчислювальних ресурсів (буде генеруватися для кожного ресурсу окремо);
- максимальний час доступу від кожного ресурсу до сховища даних;
- інтервал надходження задач у загальну чергу.

Після зчитування даних параметрів відбувається генерація списку обчислювальних ресурсів та задач, алгоритми яких зображені на рис. 2.3. Вхідними параметрами для генерації списку ресурсів є: кількість ресурсів, рівень флуктуації продуктивності ресурсів та максимально можливий час доступу до обчислювального ресурсу. Функція генерації задач приймає кількість необхідних задач, мінімальний та максимальний час виконання задачі, список ресурсів (для генерації масиву значень часу доступу до даних від кожного ресурсу) та максимальний час доступу до даних.

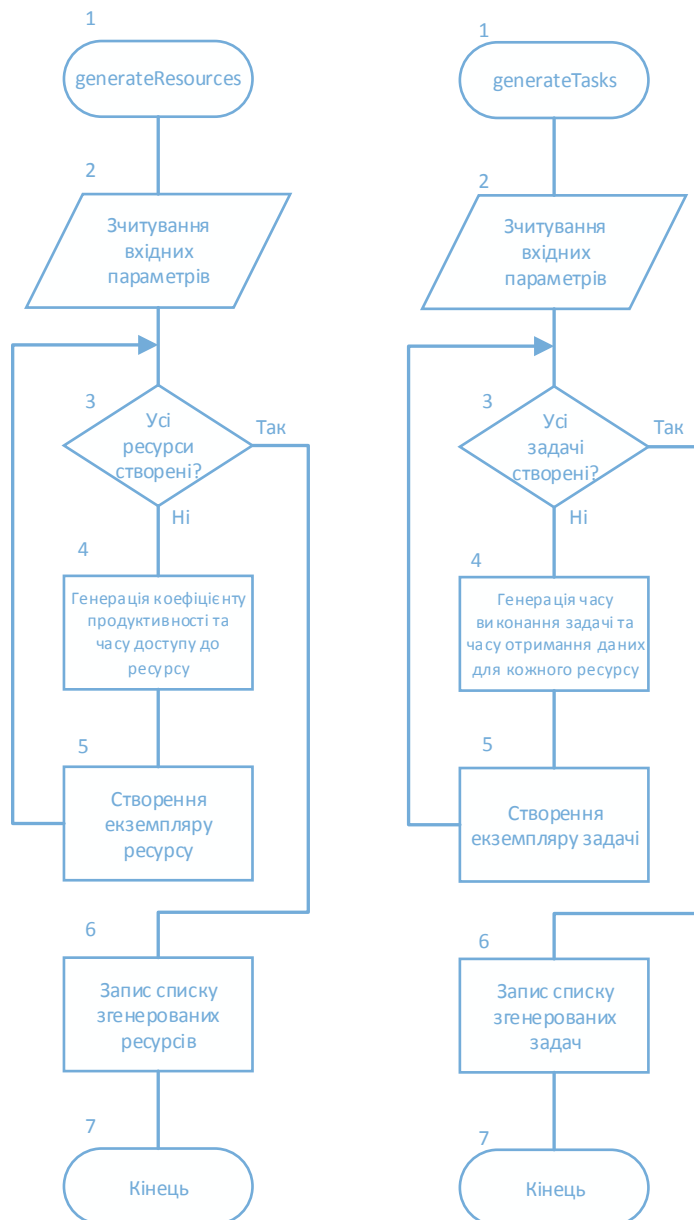


Рис. 2.3. Алгоритми генерації обчислювальних ресурсів та задач в grid-емуляторі

Після успішної генерації списків ресурсів та задач які необхідно виконати, емулятор grid-системи запускає кожен ресурс у новому потоці та очікує готовності ресурсів приймати задачі. Далі йде ітерація по списку задач та кожна задача надходить до планувальника задач, що приймає на вхід правило розподілення, задачу та список доступних ресурсів. Алгоритм запуску задач показаний на рис. 2.4. Даний етап завершується тоді, коли усі

задачі будуть завершені та усі ресурси будуть звільнені. Варто відмітити що тут кожний ресурс спостерігається менеджером, що зчитує інформацію про їх стан, а саме поточне завантаження та час простою ресурсу.

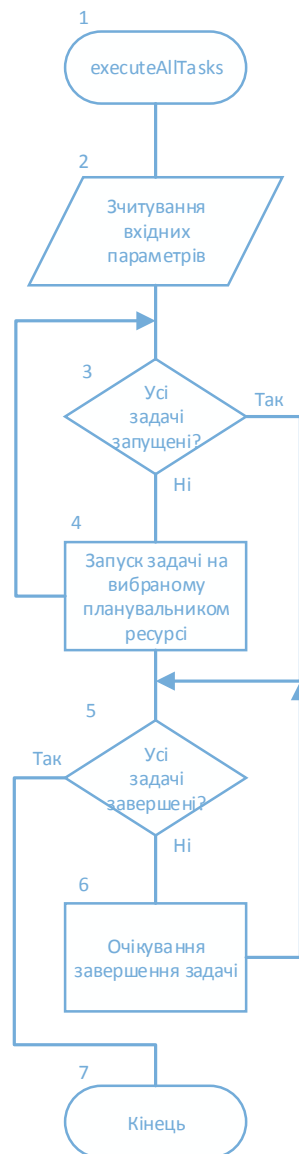


Рис. 2.4. Загальний алгоритм виконання задач на обчислювальних ресурсах

Після завершення усіх задач проводиться витягування результатів роботи grid-системи для подальшого аналізу, а саме:

- загальний час виконання усіх задач;
- середній час очікування задач у черзі;
- максимальний час очікування задач у черзі;

- середній час простою ресурсів;
- максимальний час простою ресурсів.

Після завершення роботи усіх ресурсів, їх потоки автоматично припиняються, на чому програма завершує свою роботу. Наведені вище результати записуються у файл формату .csv (текстовий формат, призначений для представлення табличних даних).

У програмі також реалізована можливість запуску одразу декількох режимів планування, що дозволяє порівнювати вихідні дані. Приклад результату роботи програми наведений у Таблиці 2.1.

Таблиця 2.1. Результати роботи емулятора grid-системи

Алгоритм	Загальний час (мс)	Сер. час очік-ння (мс)	Макс. час очік-ння (мс)	Сер. час простою (мс)	Макс. час простою (мс)
K-means	1856	64	369	88	286
Мінім. час	1837	9	91	94	221
Round-Robin	1871	51	320	78	186
Мінім. черга	1860	14	134	95	226

2.6. Існуючі алгоритми планування задач у grid-системах

Розглянемо алгоритми планування задач що найчастіше застосовуються на практиці в grid-системах.

2.6.1. Стохастичний алгоритм балансування навантаження

Цей алгоритм намагається зберегти кожен вузол зайнятим. Він призначає кожну наступну задачу в випадковому порядку до наступного доступного ресурсу без урахування його поточного навантаження. Даний алгоритм відправляє наявні на даний час невиконані завдання на вузли в

довільному порядку, незалежно від поточного навантаження вузла. Кожне завдання присвоюється вузлу в довільному порядку. Це забезпечує баланс розкладу навантаження, але це призводить до дуже поганих показників часу виконання задач.

Основною перевагою такого підходу є його простота, але його продуктивність часто далека від оптимальної.

2.6.2. Алгоритм балансування навантаження Round-Robin

Цей алгоритм використовує циклічну кругову схему: він вибирає перший ресурс випадковим чином, а потім призначає задачі для всіх інших ресурсів круговим чином. Задачі призначаються кожному ресурсу в круговому порядку без будь-яких пріоритетів, а отже, довготривалий простій у такій системі відсутній. Схема алгоритму наведена на рис. 2.5.

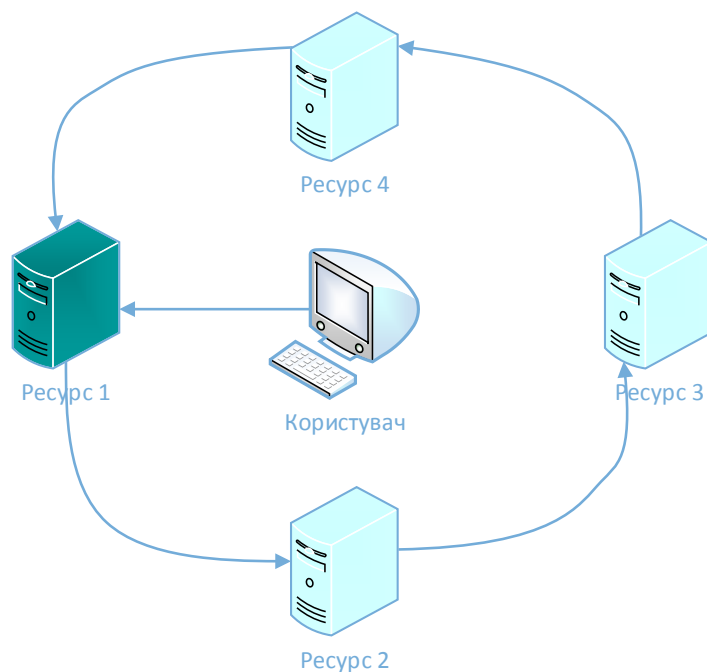


Рис. 2.5. Алгоритм балансування навантаження Round-Robin

По принципу Round Robin поточний трафік на серверах нехтується. Тому, щоб поліпшити критичні випадки у звичайній круговій системі був розроблений так званий «зважений» алгоритм Round Robin. У зваженого алгоритму перебору кожному ресурсу (серверу) призначена вага, а отже і задачі розподіляються на ресурси у відповідно до значення ваги. Ресурсам з великими потужностями присвоюється більш високе значення ваги та, відповідно, вони будуть отримувати більше задач. У ситуації, коли усі ресурси стають рівноправними, тобто мають однакові значення ваги, сервери будуть отримувати збалансований трафік. Але даний алгоритм є також статичним, тому не є кращим.

2.6.3. Алгоритм мінімального часу виконання

Метод мінімального часу виконання направляє кожну задачу на ресурс, який швидше за інші виконає її. Очікуваний час виконання для поточної задачі обчислюється на кожному ресурсі та відправляється назад до планувальника. Планувальник задач вибирає ресурс з мінімальним очікуваним часом завершення задачі. Даному алгоритму необхідні такі параметри:

1. приблизний об'єм (складність) завдання;
2. поточна приблизна швидкість виконання завдання (з урахування флуктуації);
3. поточна доступна пропускна здатність між ресурсами;
4. поточне навантаження на кожний ресурс.

Цей алгоритм потребує повної інформації про стан ресурсу (останні три параметри обчислюються на кожному сервері), а тому може погіршувати загальну ефективність системи витрачаючи ресурси на обчислювання приблизного часу виконання. Загальна схема даного алгоритму наведена на рис. 2.6.

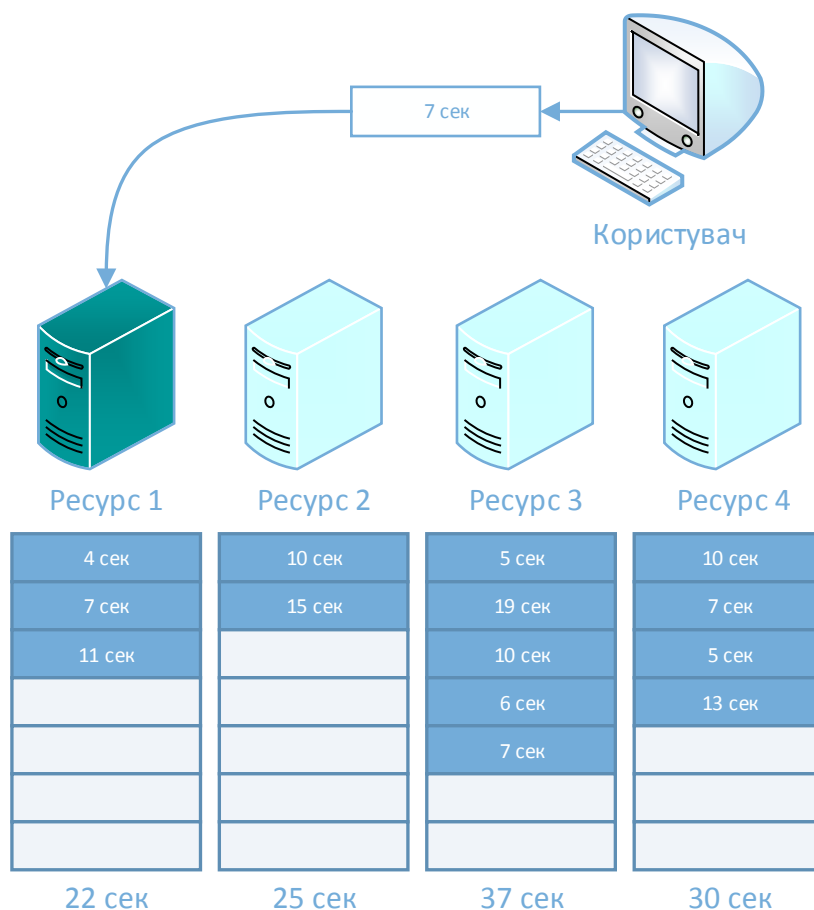


Рис. 2.6. Алгоритм балансування навантаження за мінімальним часом виконання

2.6.4. Алгоритм найменшої черги

Метод найменшої черги призначає поточну задачу на ресурс з мінімальною кількістю задач в черзі. Перевагою такого підходу є те що він не потребує обчислення складності завдання, а також детальної інформації про стан ресурсу. Він повинен знати лише один параметр: поточна кількість задач в черзі кожного ресурсу. Це значення збирається кожного разу при виконанні планувального рішення. Загальна схема даного алгоритму наведена на рис. 2.7.

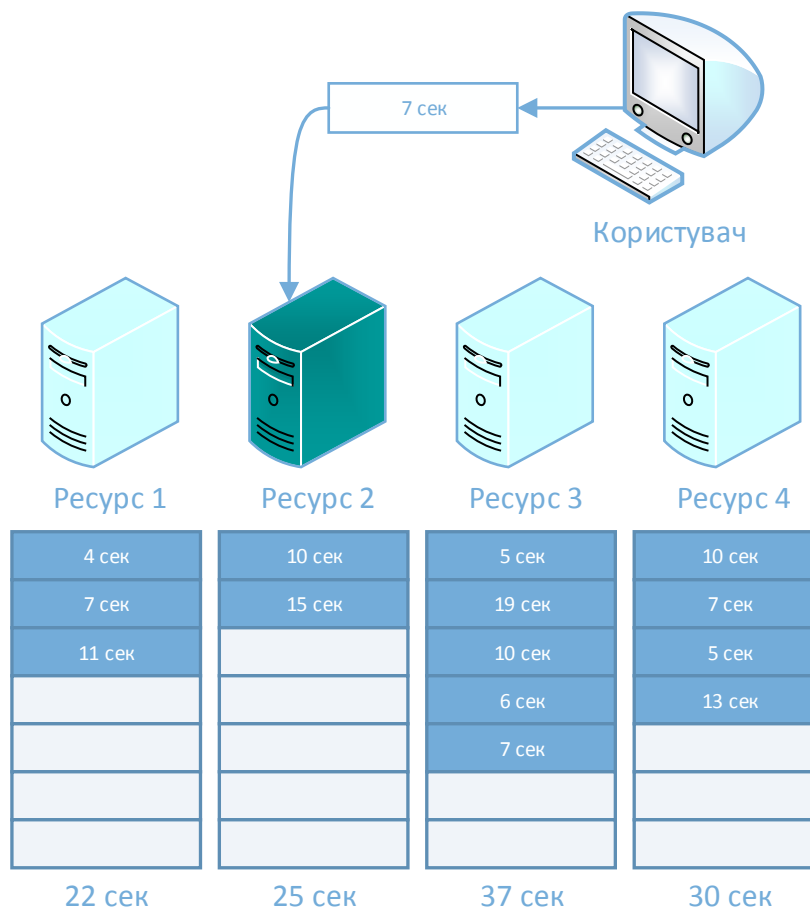


Рис. 2.7. Алгоритм балансування навантаження за мінімальною чергою

2.6.5. Алгоритм K -means

Алгоритм K -means (K -середніх) використовує аналогічний до алгоритму мінімального часу виконання (представленого у розділі 2) підхід. Єдиною відмінністю є те що замість обчислювання часу виконання на усіх ресурсах, він проводить дослідження лише деякого масиву ресурсів, що зменшує витрати на надмірні обчислення та зв'язок. Не зважаючи на те, що метод K -групових середніх може знизити надлишкові обчислення, він може спричинити погіршення загальної продуктивності системи якщо коефіцієнт K буде вибраний неналежним чином. Тому для більше успішного застосування даного алгоритму необхідно проводити попередній стохастичний

експеримент по підбору цього коефіцієнту, що також потребує достатньо ресурсів. Загальна схема даного алгоритму наведена на рис. 2.8 [8].

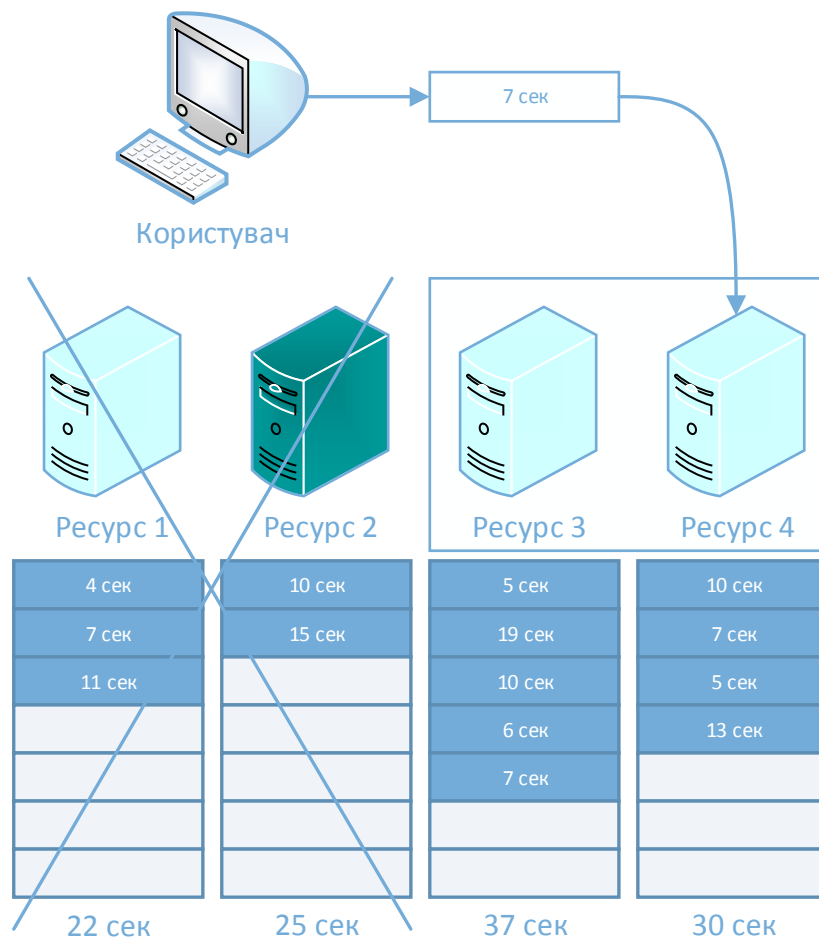


Рис. 2.8. Алгоритм балансування навантаження за мінімальною чергою

2.7. Перевірка роботи алгоритмів планування задач

Розглянемо детальніше ефективність наведених вище алгоритмів планування задач в розробленому емуляторі grid-системи. Для порівняння роботи різних алгоритмів планування задач буде здійснено з наступними параметрами емулятора:

- 10 обчислювальних ресурсів;
- 500 задач;

- коефіцієнт продуктивності кожного ресурсу в межах від 0.8 до 1.2 (тобто ресурс з коефіцієнтом 0.8 буде в 1.5 рази повільніше виконувати задачу ніж ресурс з коефіцієнтом 1.2);
- середній час виконання однієї задачі становить 1 секунда з флуктуацією 200 мс;
- період додавання задачі в чергу ресурсу становить 100 мс;
- час доступу до ресурсу дорівнює 5мс з флуктуацією 5мс;
- час доступу до сховища даних дорівнює також 5мс з флуктуацією 5мс.

Емуляція роботи grid-системи була проведена на робочій станції з 64-розрядною операційною системою Windows 7 з процесором Intel Core i7-4770 з тактовою частотою 3.4 ГГц та 16 ГБ оперативної пам'яті.

Для більш об'єктивних результатів було проведено 10 незалежних експериментів та в якості результатів було підраховано середнє арифметичне усіх значень. Результати роботи планувальника задач з різними алгоритмами балансування навантаження наведені у табл. 2.2, 2.3 та рис. 2.9, 2.10.

Спочатку необхідно дослідити наскільки ефективно будуть себе проявляти алгоритми в ідеальних умовах, тобто без затримок між планувальником задач та обчислювальними ресурсами та від обчислювальних ресурсів до сховищ даних. Результати емуляції такого варіанту наведені у таблиці 2.2.

Таблиця 2.2. Результати роботи алгоритмів планування задач з урахуванням миттєвого доступу від планувальника задач до обчислювального ресурсу та від обчислювального ресурсу до сховища даних

Алгоритм	Загальний час (мс)	Середній час очікування (мс)	Макс. час очікування (мс)	Середній час простою (мс)	Макс. час простою (мс)
Стохастичний	64899,5	3475,6	16005,4	5209,3	13423,6
Round-Robin	60013,4	1254,5	9178	2573	8094,7
K-means	52260,3	645,6	1849,8	1580,3	2692,2
Мінім. час	52182,2	476	1575,5	1423,1	2508,2
Мінім. черга	52124,5	526,9	1774,2	1492,6	3773,2

Як бачимо, найбільш швидкими в даному випадку показали себе динамічні алгоритми, а саме мінімального часу, мінімальної черги та K-means. Очевидно, що вони є найбільш ефективними у даному випадку, оскільки обчислення найбільш оптимального ресурсу для виконання задачі відбувається миттєво, в той час як навантаження на ресурси є збалансованим.

Оскільки алгоритми мінімальної черги та мінімального часу виконання при кожному запиті на визначення найкращого ресурсу опитують кожен доступний ресурс, це може погіршити їх ефективність у разі великого рівня розподіленості системи. Тому необхідно провести дослідження, що є більш наближеним до реального випадку, а саме з урахуванням часу доступу від планувальника задач до обчислювальних ресурсів та від обчислювальних ресурсів до сховища даних, де будуть зберігатися дані, що необхідні для задач. Результати проведеного дослідження наведені у Таблиці 2.3.

Таблиця 2.3. Результати роботи алгоритмів планування задач з урахуванням доступу від планувальника задач до обчислювального ресурсу та від обчислювального ресурсу до сховища даних з випадковою затримкою до 10 мс.

Алгоритм	Загальний час (мс)	Середній час очікування (мс)	Макс. час очікування (мс)	Середній час простою (мс)	Макс. час простою (мс)
Стохастичний	72409,20	3413,90	19992,20	6215,90	16817,70
Round-Robin	60294,70	986,20	7110,80	3622,20	9767,70
K-means	74890,50	77,20	952,30	21725,00	27460,40
Мінім. час	383187,33	645,00	1345,11	68043,89	264086,44
Мінім. черга	384269,00	647,33	1206,89	68446,56	266066,78

З наведених вище результатів видно що динамічні алгоритми балансування стають значно неефективними у роботі при високій розподіленості системи у зв'язку з тим що кожен запит до обчислювального ресурсу та до сховища даних займає деякий час. Не дивлячись на те що цей час становив у середньому всього 5мс, це призвело до значного росту загального часу виконання 1000 задач паралельно на 10 ресурсах. Тобто тут ми маємо приріст загального часу виконання в середньому на величину, що вираховується за наступною формулою:

$$t_{\text{заг}}^{\text{з затр.}} - t_{\text{заг}}^{\text{без затр.}} \approx N_{\text{задач}} * N_{\text{ресурсів}} * t_{\text{затримки}},$$

де $t_{\text{заг}}^{\text{з затр.}}$ – загальний час виконання усіх задач з урахуванням часу затримки зв'язку між обчислювальними ресурсами;

$t_{\text{заг}}^{\text{без затр.}}$ – загальний час виконання усіх задач без урахування часу затримки зв'язку між обчислювальними ресурсами;

$N_{\text{задач}}$ – кількість задач;

$N_{\text{ресурсів}}$ – кількість ресурсів;

$t_{\text{затримки}}$ – середній час затримки при зверненні між обчислювальними ресурсами, планувальником задач та сховищем даних.

Більш наочне співставлення результатів в обох випадках наведено на рис. 2.9.

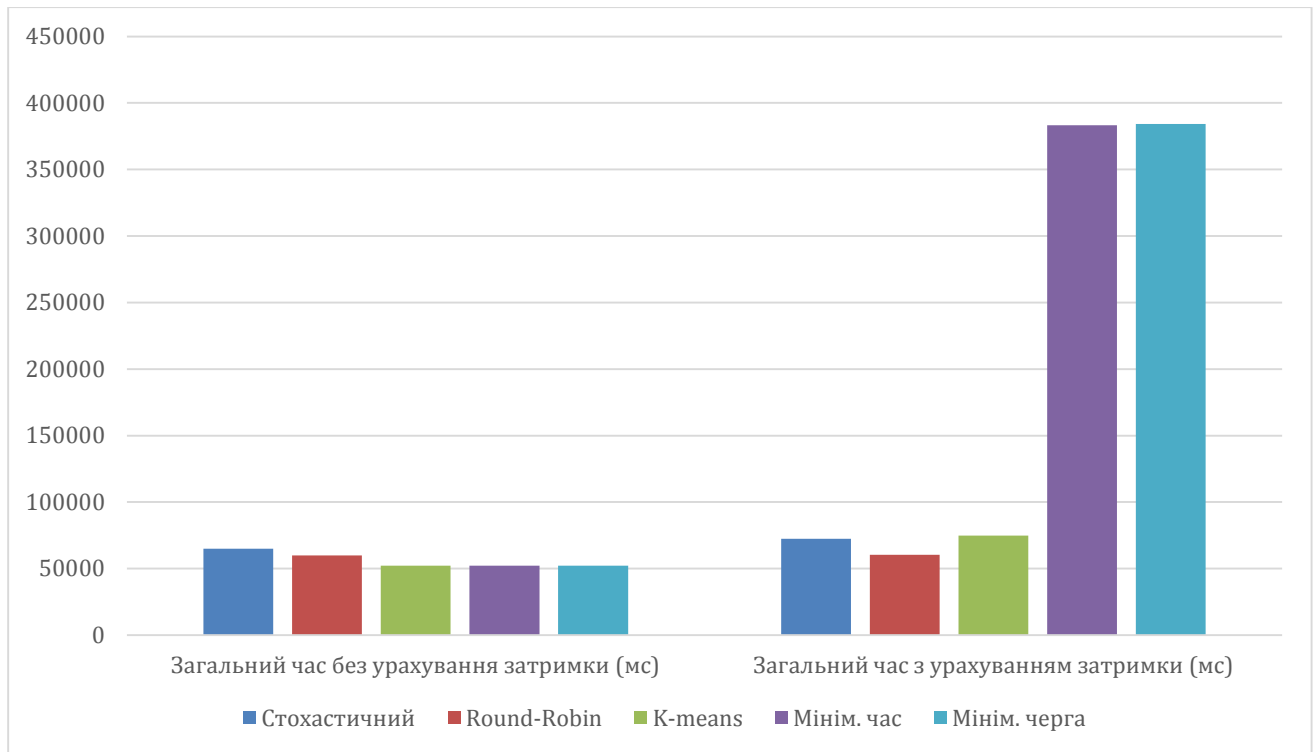


Рис. 2.9. Порівняння результатів загального часу виконання задач з та без врахування затримки зв'язку між розподіленими компонентами системи

Наступним важливим параметром є час очікування задачі в черзі на виконання. Він вираховується як різниця часу від постановки задачі у чергу до початку її виконання. Як видно з рис. 2.10, усі статичні алгоритми мають приблизно незмінне значення цього показника з введенням додаткового часу (затримки), необхідного для зв'язку планувальника задач з обчислювальними ресурсами. Варто відмітити що стохастичний алгоритм має значно гірші результати у порівнянні з іншими алгоритмами. Це пов'язано з тим, що він

може призначати задачі ресурсам що мають великі черги, відтак, збільшуючи загальний час виконання та середній час очікування задачі у черзі.

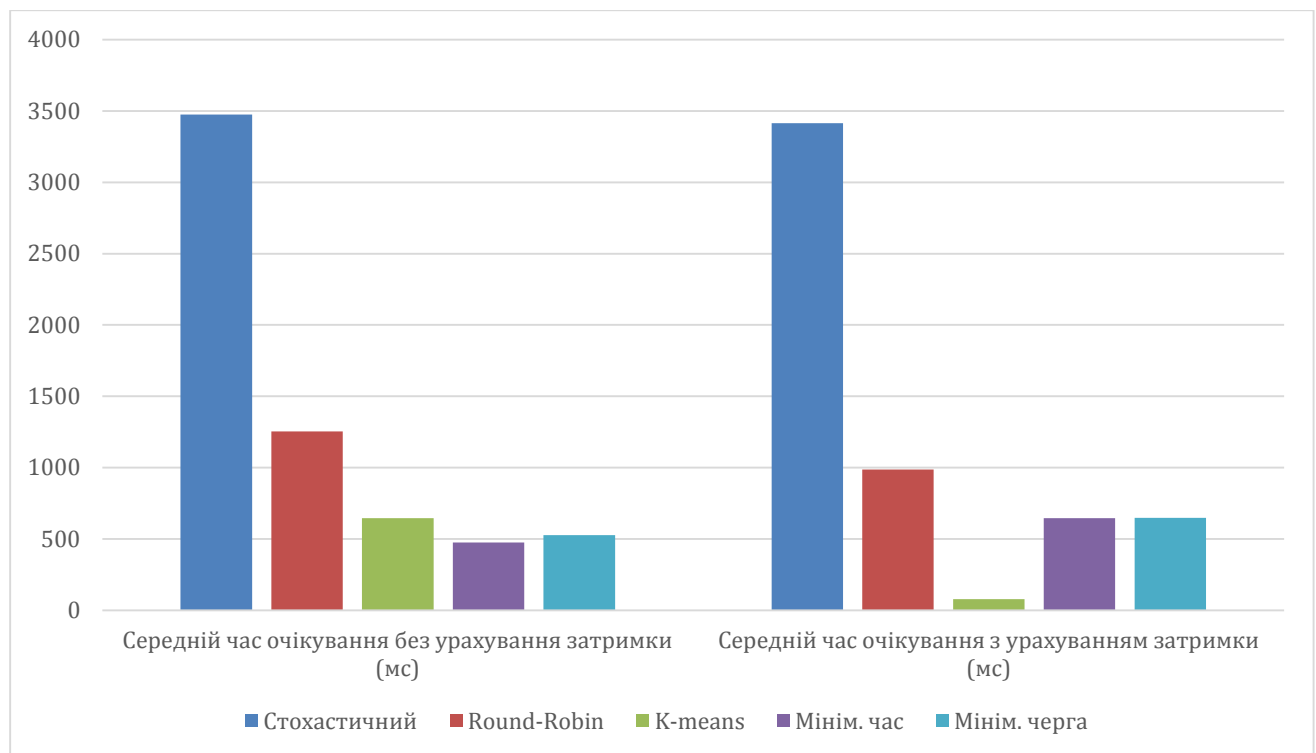


Рис. 2.10. Порівняння результатів середнього часу очікування задачі у черзі

Варто звернути увагу ще на один показник – середній час простою ресурсів. Фактично, це час що був витрачений ресурсом дарма, в той час як інші ресурси були зайняті та виконували деякі задачі. З рис. 2.11 видно що з введенням додаткової затримки між планувальником, обчислювальними ресурсами та сховищем даних. Цей час значно погіршився у динамічних алгоритмах, оскільки затримка при додаванні задачі у чергу проходить двічі: перший раз при зверненні від планувальника задач до кожного ресурсу для отримання його статусу, а другий раз при завантаженні даних зі сховища.

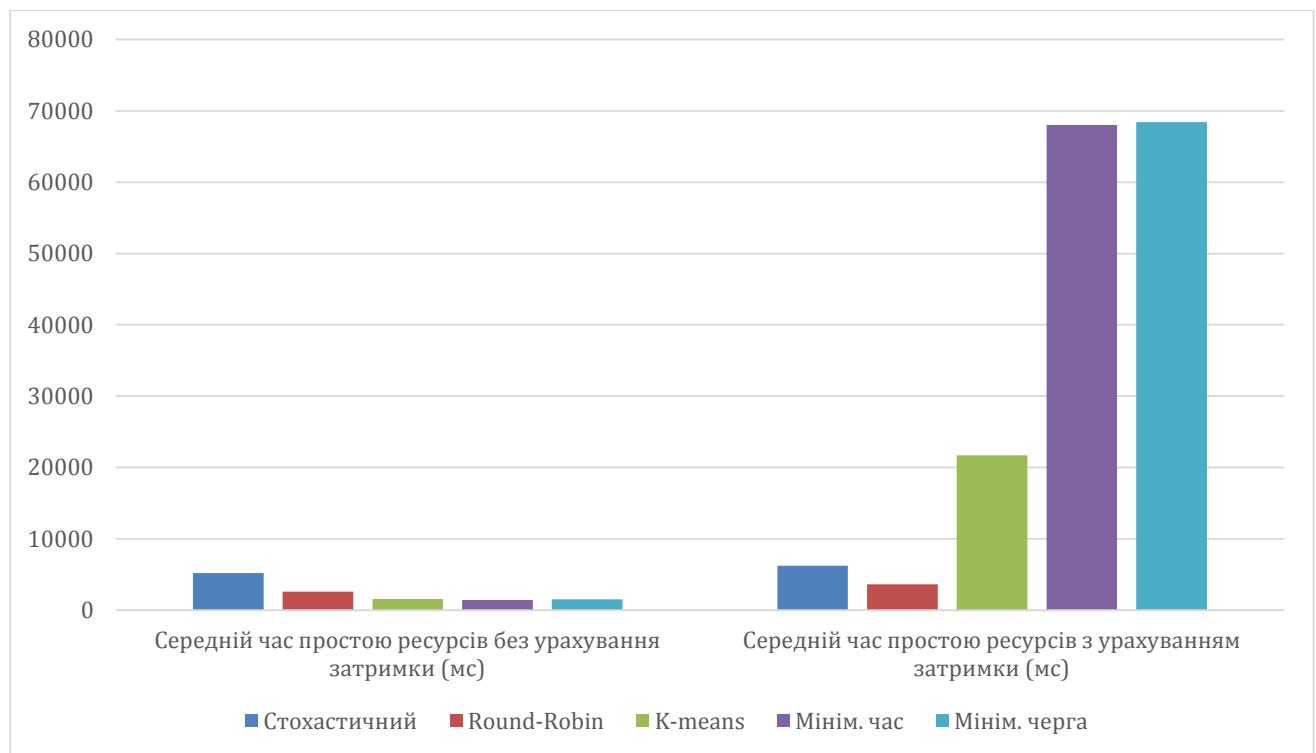


Рис. 2.11. Порівняння результатів середнього часу простою обчислювальних ресурсів

Отже, з вищенаведеного дослідження ефективності роботи алгоритмів балансування навантаження можна зробити висновок що кожен алгоритм має місце бути. Тому при моделюванні підсистеми планування задач слід враховувати специфіку архітектури grid-системи. При великій розподіленості системи слід звернути увагу на статичні алгоритми планування задач, що нехтують поточним станом ресурсу та не потребують актуальної поточної інформації з кожного обчислювального ресурсу. При невеликій розподіленості таких ресурсів та сховищ даних варто використовувати динамічні алгоритми, які більш ефективно розподіляють навантаження на ресурси шляхом врахування поточного стану кожного ресурсу та максимізують як загальний час виконання задач, так і середній час очікування задач у черзі.

Висновки до розділу

1. В даному розділі проаналізовано усі аспекти підсистеми планування задач у розподіленій grid-системі включаючи роль даного компоненту у системі.
2. Показано що він відіграє головну роль та потребує подальшого дослідження.
3. Наведений підхід до аналізу розподілення задач та оцінка ефективності алгоритмів планування використовується для верифікації різних методів розподілення задач.
4. Розроблено емулятор grid-системи, що може бути застосований для верифікації ефективності алгоритмів планування.
5. Для існуючих алгоритмів планування задач у grid-системах вказано недоліки та переваги, а також виявлено що вони потребують покращення.

3. МОДЕЛЮВАННЯ ОПТИМАЛЬНОГО АЛГОРИТМУ ПЛАНУВАННЯ ЗАДАЧ

Отже, проаналізувавши усі методи планування задач у розділі 2, було виявлено що кожен з них має свої переваги та недоліки. Тому метою даного розділу буде розробити новий алгоритм, що буде враховувати усі недоліки та вади кожного з існуючих алгоритмів. Новий алгоритм повинен враховувати гетерогенність та розподіленість обчислювальних ресурсів та сховища даних, що підвищить як загальний час обробки даних, так і мінімізує час очікування задач в черзі. Не менш важливим критерієм оптимальності алгоритму є також час простою обладнання. Зводячи даний параметр до мінімуму, grid-система буде функціонувати максимально ефективно, зводячи зайві витрати до мінімуму.

3.1. Варіанти покращення та виявлення впущених деталей у існуючих алгоритмах

Для розробки нового алгоритму необхідно виділити ряд недоліків існуючих алгоритмів планування задач та запропонувати варіанти їх виправлення. Також потрібно врахувати усі суттєві моменти алгоритмів, які відіграють ту чи іншу роль та мають бути застосовані у підсистемі планування задач. Спираючись на дослідження існуючих алгоритмів планування задач, проведеного у розділі 2, варто виділити наступні переваги та недоліки статичних та динамічних алгоритмів.

Реальна grid-система завжди є більш-менш розподіленою, тому необхідно враховувати затримку в сигналах, якими обмінюються обчислювальні ресурси з планувальником задач та сховищем даних. Статичні алгоритми (у нашому випадку це є стохастичний алгоритм та алгоритм Round-Robin) не приймають до уваги поточний стан обчислювального ресурсу, тому

швидкість додавання задачі у чергу у них максимальна. Недоліком в цьому плані виступає можливе неоптимальне розподілення задач по ресурсам. Тому варто скомбінувати статичну та динамічну складову алгоритмів шляхом опитування кожного ресурсу не кожного разу при вирішенні задачі балансування, а з деяким періодом. Це реалізується кешуванням станів ресурсів у підсистемі планування задач, що буде зберігати дані про них у пам'яті. Ці дані будуть оновлятися у фоновому потоці з періодом, що буде адаптивно змінюватися в залежності від навантаження на систему. Отже, при запиті до планувальника, алгоритм повинен буде брати до уваги інформацію про стан кожного ресурсу з кешу.

Алгоритм K-means гарно себе показав у результатах загального часу виконання задач. Це обґрунтовується тим, що він опрацьовує тільки деяку підмножину ресурсів, що зменшує час на обробку всього їх масиву. Тому, враховуючи те що новий алгоритм буде використовувати хешування даних про стан ресурсів, у фоновому потоці варто здійснювати ще й корегування пріоритету використання кожного ресурсу. Дана особливість може бути реалізована сортуванням списку ресурсів по рівню їх поточного завантаження. Алгоритм планування при обробці кешу буде проходити лише першу частину списку ресурсів що мають більший пріоритет. Отже, обробка лише частини списку ресурсів збільшить швидкість вирішення проблеми планування.

В якості правила вибору найкращого ресурсу для виконання поточної задачі слід порівнювати загальний час виконання задачі на них. Для цього необхідно враховувати віддаленість обчислювальних ресурсів від сховища даних. Отже, при однаковому часі виконання задачі на двох ресурсах слід віддавати перевагу тому ресурсу, який знаходиться ближче до ресурсу (сховища), де зберігаються дані необхідні для виконання поточної задачі.

З гетерогенності (різномірності) ресурсів grid-системи випливає що кожна обчислювальна ланка може візнитися наступними характеристиками: кількістю оперативної пам'яті, процесорною потужністю, поточною завантаженістю файлової системи. Тому при оновленні стану ресурсу у кеші, необхідно обчислювати поточний рівень продуктивності та періодично його оновляти. Даний коефіцієнт буде використовуватися як множник часу виконання задачі на ресурсі. Отже, при відносно однаковому часі виконання задачі на ресурсах, планувальник повинен вибирати більше потужний ресурс, що виконає одну й ту саму задачу за менший інтервал часу.

Вищенаведені примітки так чи інакше можуть суперечити одне одному, тому при розробці узагальненого алгоритму варто враховувати оптимальні параметри моделі, які система повинна адаптувати під завантаженість та поточний стан ресурсів, інтенсивності потоку задач та вимоги до швидкості обробки даних.

3.2. Розробка адаптивного алгоритму планування задач

При розробці алгоритму планування задач були агреговані усі умови та правила, наведені у розділі 2. Оскільки розроблений алгоритм видає рішення ґрунтуючись на поточному стані та властивостей ресурсів, а також на віддаленості ресурсів одне від одного, алгоритм отримав назву адаптивного.

Блок-схема роботи планувальника задач що викорисстовує розроблений алгоритм для балансування навантаження наведена на рис. 3.1.

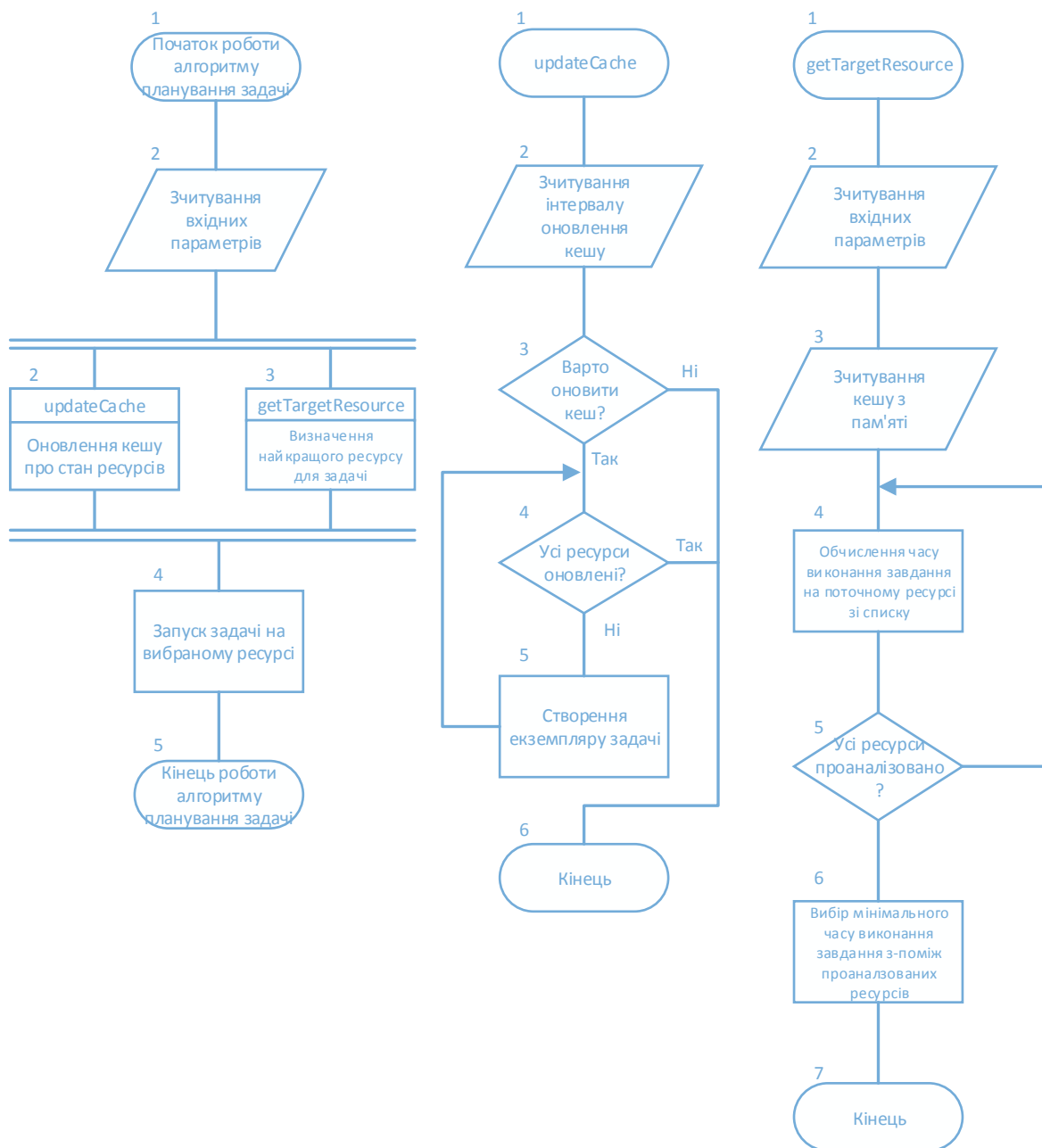


Рис. 3.1. Блок-схема адаптивного алгоритму планування задач

В першу чергу варто відмітити що, на відміну від інших, даний адаптивний алгоритм аналізує саме кеш стану ресурсів, що значно покращує загальний час виконання задач. Процедура аналізу відбувається наступним чином.

1. На вхід приймаються параметри алгоритму, а саме: задача, список ресурсів та період оновлення кешу.
2. Проводиться оновлення списку стану ресурсів тільки у випадку перевищення інтервалу з моменту останнього оновлення.
3. Паралельно з цим проводиться власне вирішення задачі планування:
 - 3.1. Кеш стану ресурсів зчитується з пам'яті.
 - 3.2. Вибирається ресурс з найменшим спрогнозованим часом виконання задачі, що вираховується за наступною формулою:

$$R_i = \min_{i=1..N} (t_i + rt_i^T + \frac{t_i^T}{p_i}),$$

де R_i – цільовий ресурс;

t_i – час виконання поточної черги задач на ресурсі;

rt_i^T – час доступу до сховища даних де зберігаються дані для задачі T від обчислювального ресурсу i;

t_i^T – час виконання задачі T на ресурсі i;

p_i – поточний рівень ефективності ресурсу i.

3.3. Кеш стану ресурсу що був вибраний для виконання задачі оновлюється згідно з тривалістю цієї задачі.

4. Вибраний ресурс для даної задачі далі повертається планувальнику для подальшого її запуску.

3.3. Визначення оптимальних коефіцієнтів моделі

Наведений вище алгоритм використовує показники, оптимальні значення яких варто дослідити в залежності від параметрів системи. В першу чергу необхідно звернути увагу на час оновлення кешу, що зберігає інформацію про стан ресурсів. Ефективність розподілення задач при варіюванні значення цього показника залежить від інтенсивності потоку нових задач та кількості ресурсів.

Для початку необхідно визначити оптимальне значення часу оновлення кешу про стан ресурсів при таких параметрах системи:

- 10 обчислювальних ресурсів;
- 200 задач;
- коефіцієнт продуктивності кожного ресурсу в межах від 0.9 до 1.1;
- середній час виконання однієї задачі становить $1 \text{ секунда} \pm 200 \text{ мс}$;
- період додавання задачі в чергу ресурсу становить 100 мс;
- час доступу до ресурсу та до сховища даних дорівнює $5 \text{ мс} \pm 5 \text{ мс}$.

Розглянемо детальніше графіки залежності часу оновлення кешу планувальника від загального часу виконання усіх задач, середнього часу очікування задачі у черзі та середнього часу простою обладнання, наведених на рис. 3.2, 3.3, 3.4 відповідно.

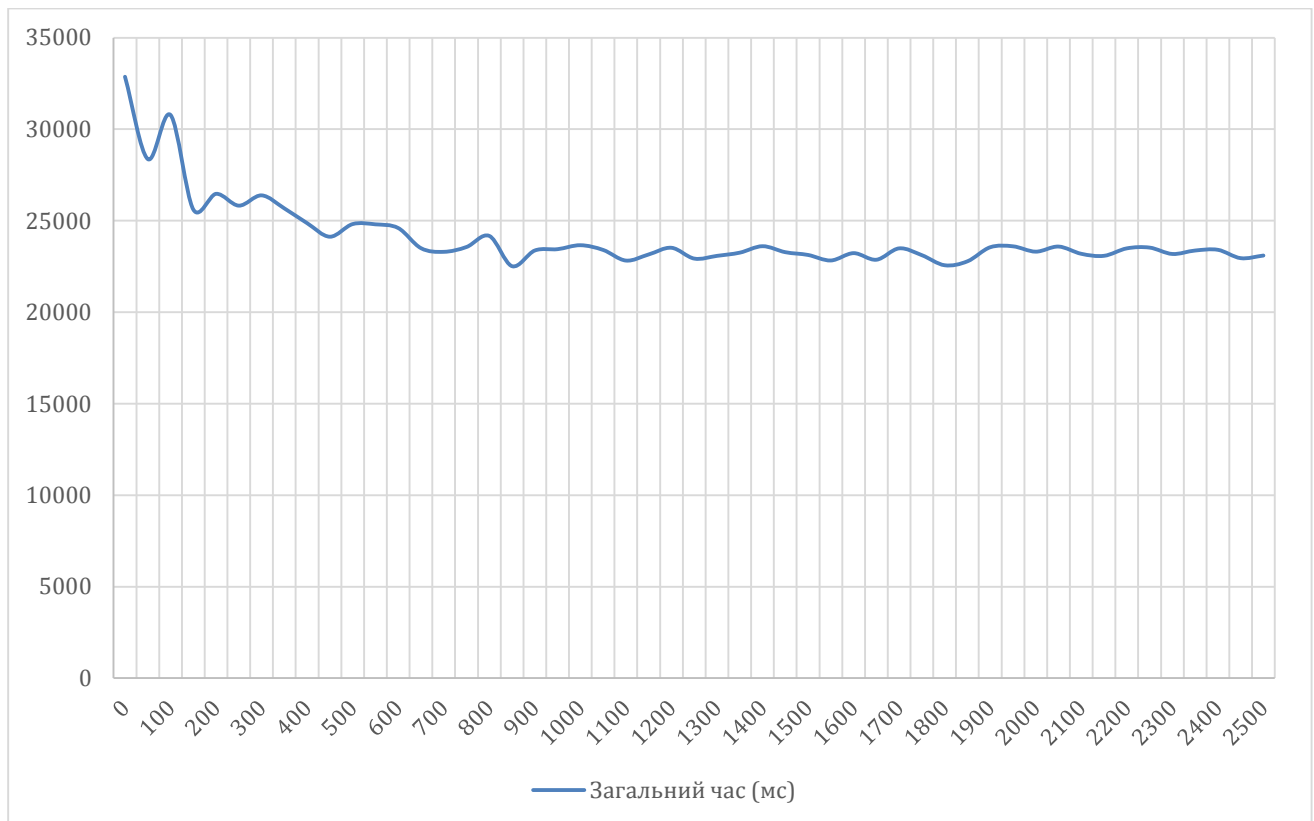


Рис. 3.2. Графік залежності часу оновлення кешу про стан обчислювальних ресурсів від загального часу виконання задач

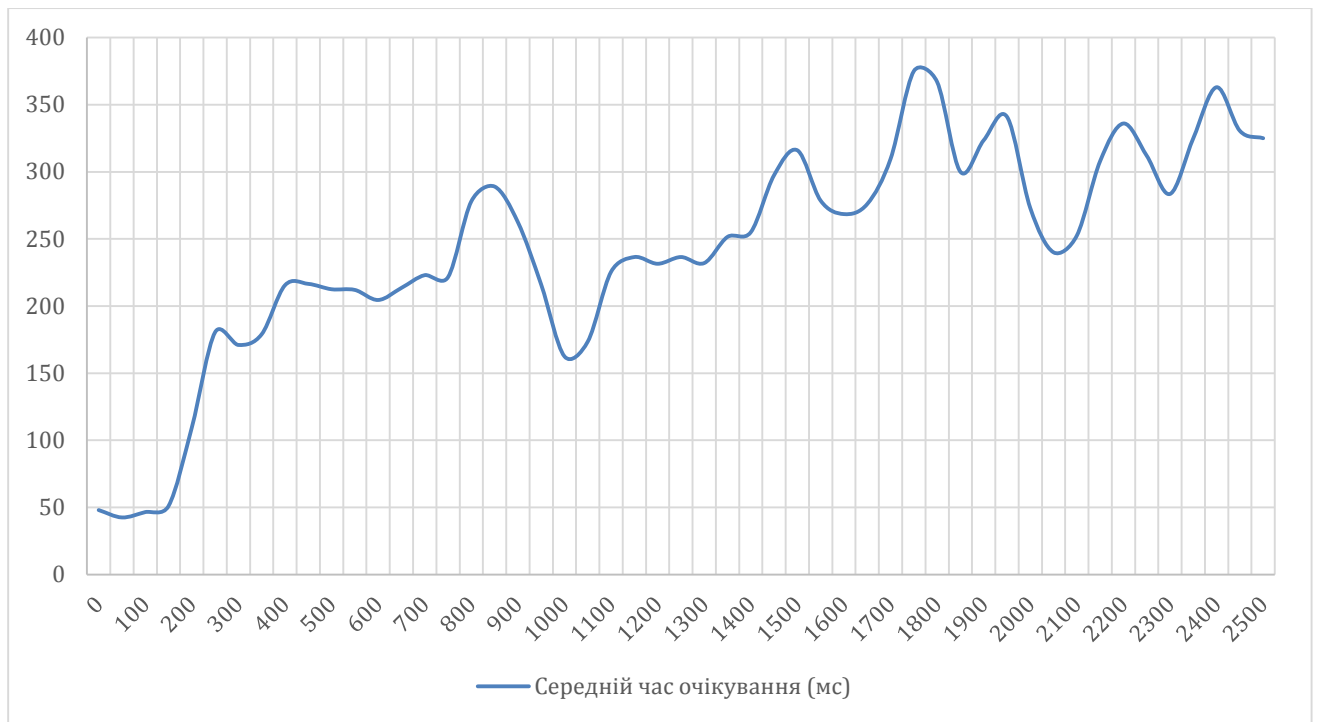


Рис. 3.3. Графік залежності часу оновлення кешу про стан обчислювальних ресурсів від середнього часу очікування задачі у черзі

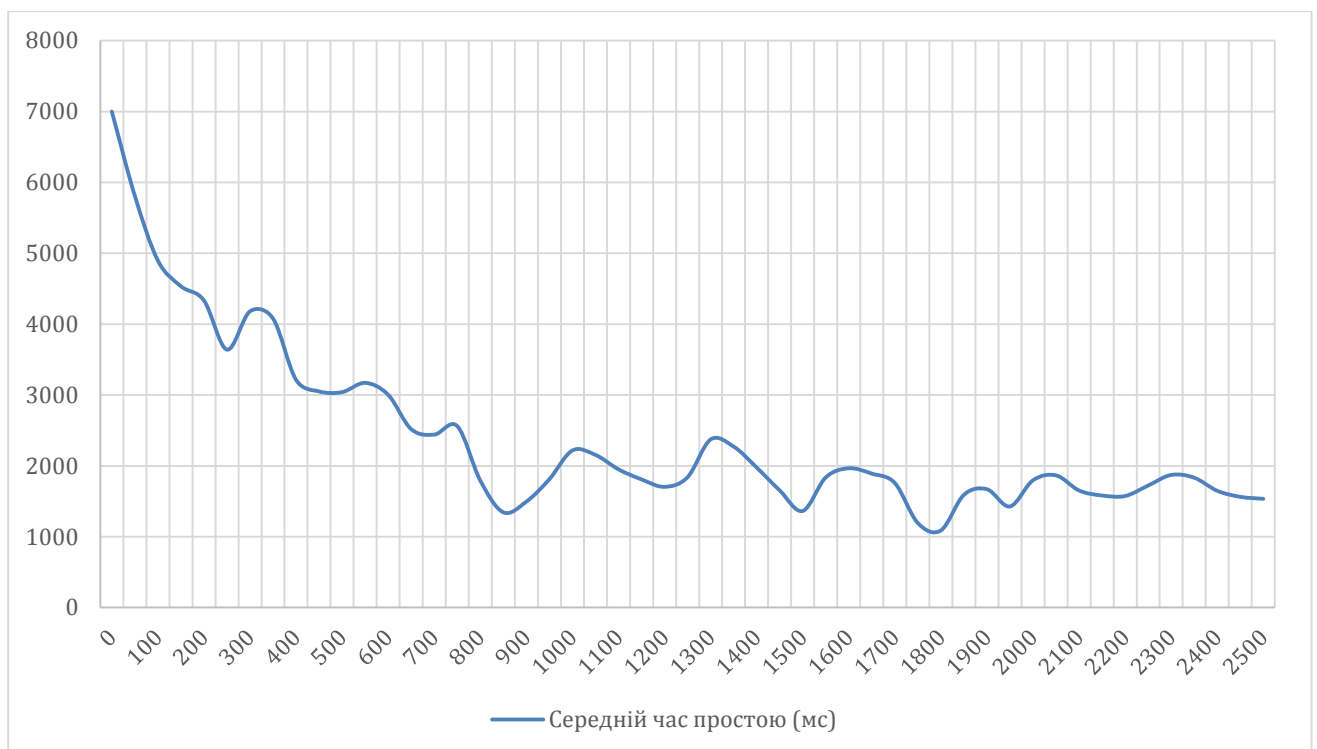


Рис. 3.4. Графік залежності часу оновлення кешу про стан обчислювальних ресурсів від середнього часу простою обчислювальних ресурсів

Очевидно, що при малому значенні оновлення кешу, запити до обчислювальних ресурсів відбуваються частіше, а отже це підвищує як загальний час обчислення усіх задач, так і збільшує середній час простою обладнання. Середній час очікування задач у черзі зі збільшенням періоду оновлення кешу зростає у зв'язку з тим що кеш стану ресурсів може бути застарілий, а тому планувальник може направляти поточну задачу на ресурс, який почне виконувати її довше ніж решта ресурсів. Тому скомбінувавши дані залежності, отримаємо оптимальне значення періоду оновлення кешу що мінімізує середній час очікування задачі у черзі при мінімальному значенні загального часу виконання усіх задач. Знайдене оптимальне значення наведене на рис. 3.5.

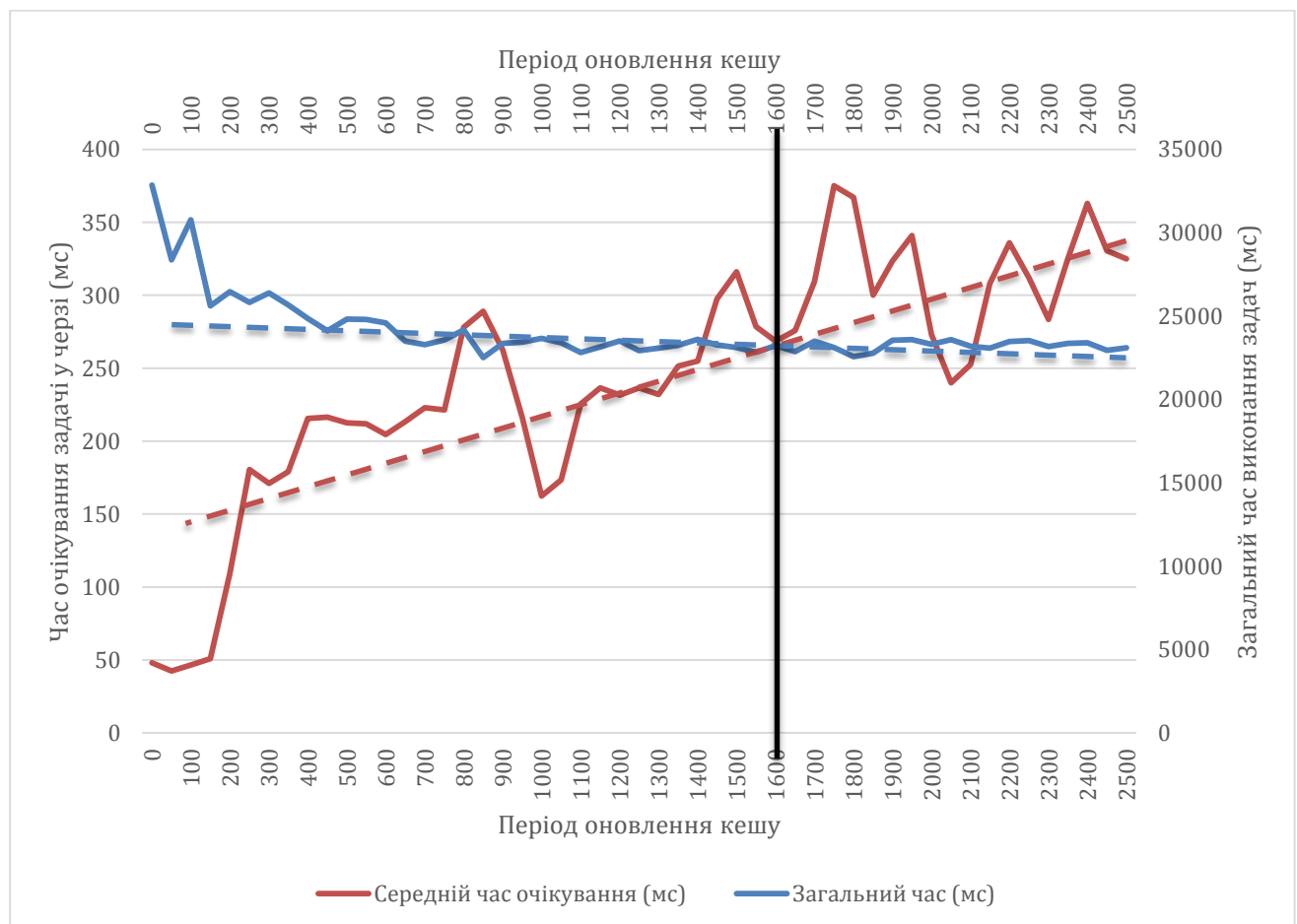


Рис. 3.5. Оптимальне значення періоду оновлення стану ресурсів у кеші

Наведений вище експеримент був проведений при незмінній інтенсивності потоку задач, тому для об'єктивної оцінки оптимального значення варто дослідити залежність параметрів системи від потоку задач що надходять до неї. Наочне представлення залежності трьох параметрів системи, а саме: період додавання задач у чергу (у мс), загальний час виконання задач (у мс) та період оновлення кешу наведене на рис. 3.6.

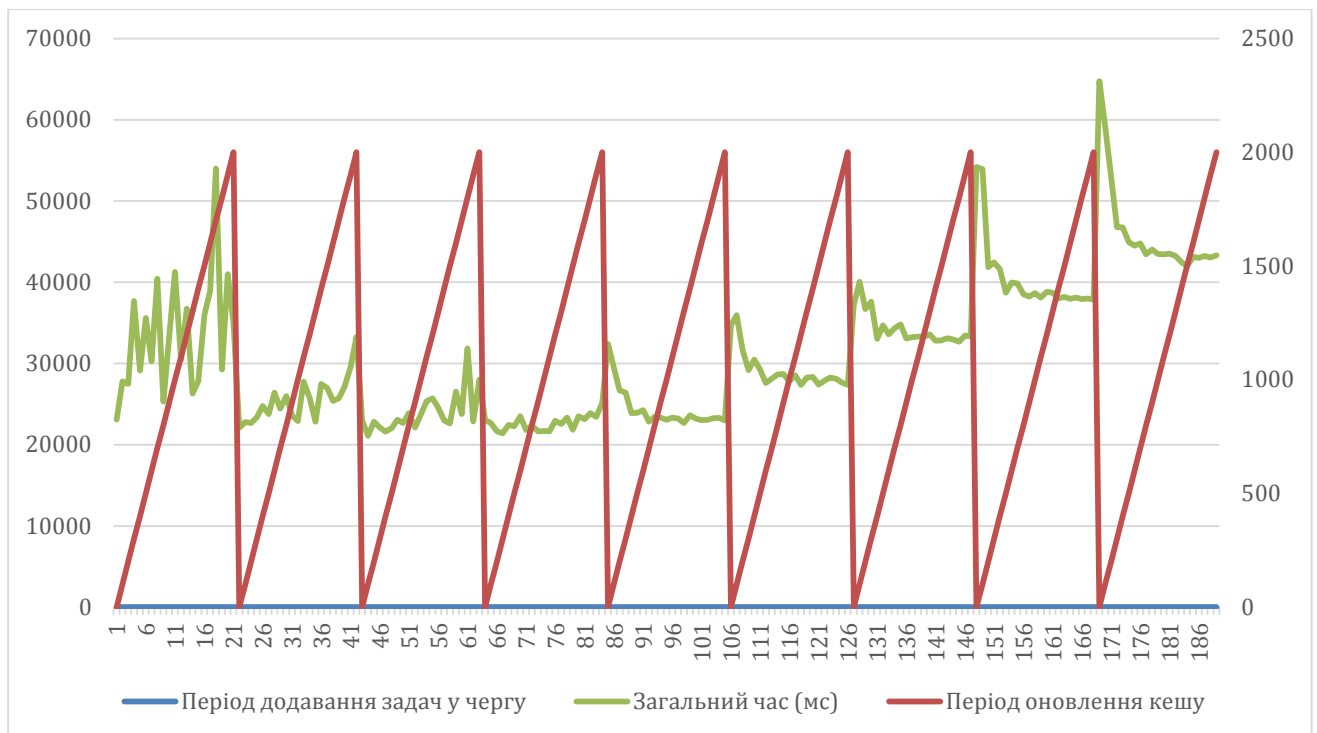


Рис. 3.6. Залежність періоду оновлення кешу від періоду додавання задач у чергу

Дослідження проводилось на 10 обчислювальних ресурсах (N_p) при 200 задачах з середньою тривалістю $t_z = 1$ сек. Нехтуючи затримками при виконанні рішення планування та проведенні обчислень, можна порахувати допустиме навантаження (інтенсивність потоку задач I) що мінімізує очікування задач у черзі на виконання та не призведе до простою обладнання за наступною формулою:

$$I \approx \frac{t_3}{N_p} = \frac{1 \text{ сек}}{10} = 100 \text{мс} ,$$

де I – інтенсивність потоку задач,

t_3 – середня тривалість задачі,

N_p – кількість обчислювальних ресурсів.

Тому у даному випадку інтенсивність потоку задач можна розбити на два періоди: до 100мс (недовантаженість системи, коли інтенсивність потоку задач менша за пропускну здатність grid-системи) та більша за 100мс (перевантаженість, коли задачі наколюються в черзі на виконання).

Отже, з наведеного вище графіка залежності інтенсивності потоку задач, та періоду оновлення кешу видно що для мінімізації загального часу виконання задач необхідно слідувати одній з наведених нижче стратегії.

- При такому потоці задач що недовантажує систему (рис. 3.7) та не виникає черги на виконання задач, варто проводити оновлення кешу якомога рідше. Це пов'язано з тим що ресурси знаходяться в режимі спокою та готові приймати нові задачі на виконання. Тому задачі не будуть затримуватися у черзі та виконуватися миттєво.

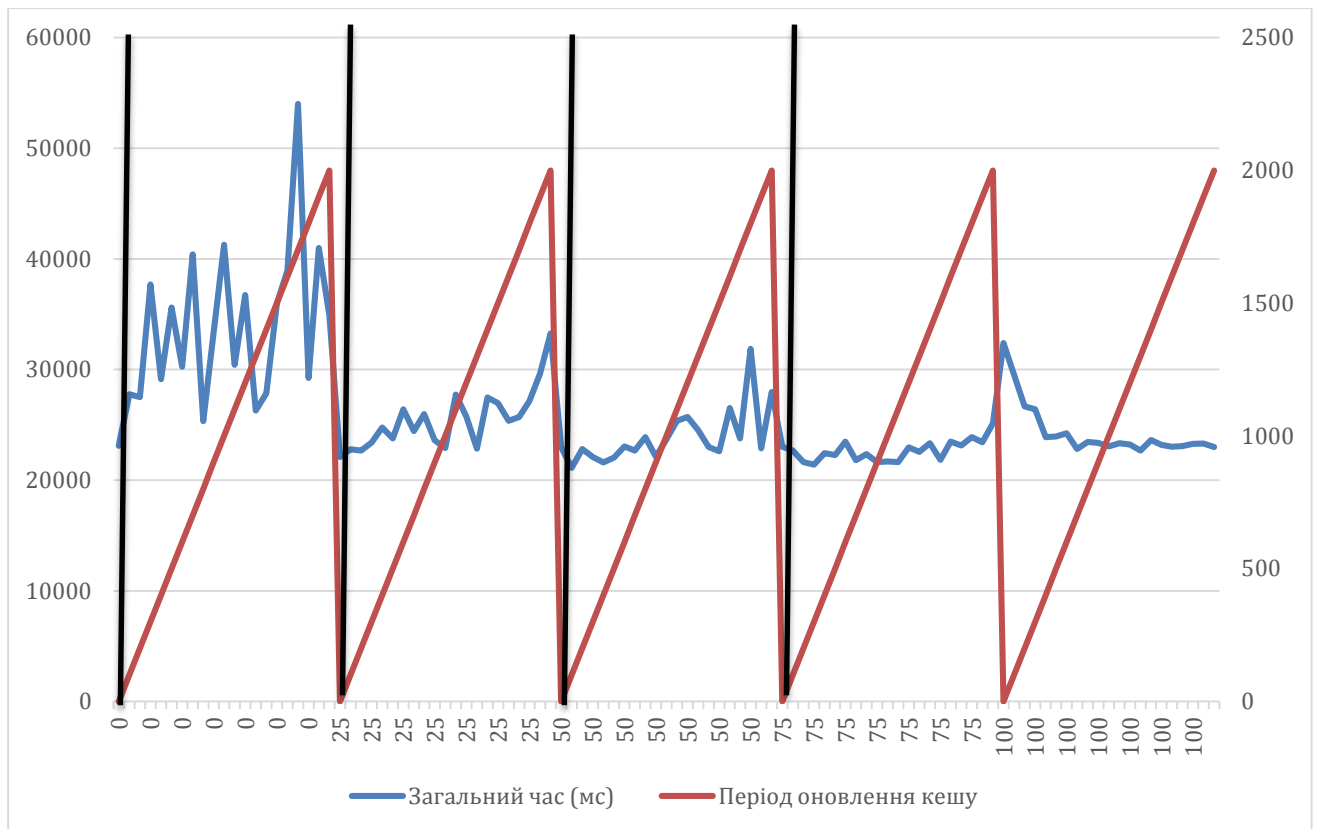


Рис. 3.7. Оптимальні значення оновлення кешу в режимі недовантаженості системи

- При перевантаженості системи (рис. 3.8) виникає зворотна ситуація. grid-система отримує велику кількість задач, що не відповідають кількості доступних обчислювальних ресурсів. Тому виникають черги на виконання та виникає необхідність розподіляти задачі на більш вільні ресурси. У зв'язку з цим потрібно оновляти кеш інформації про стан ресурсів якомога частіше. Це призведе до більшої ефективності розподілу задач та мінімізує загальний час їх обробки.

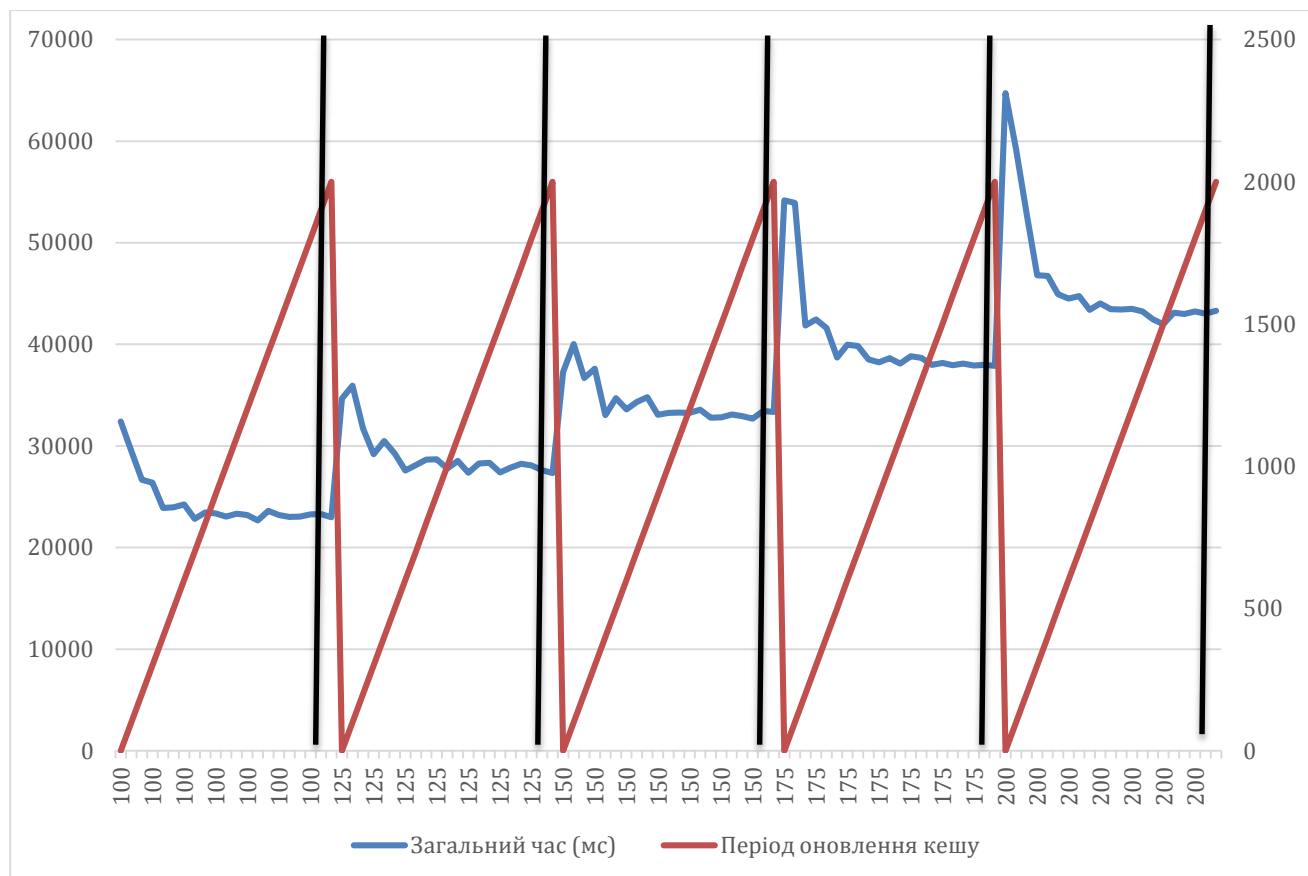


Рис. 3.8. Оптимальні значення оновлення кешу в режимі перевантаженості системи

3.4. Порівняння запропонованого та існуючих алгоритмів на віртуальній системі

Отже, визначивши оптимальну модель адаптивного алгоритму балансування навантаження розподіленої системи слід порівняти результати його емуляції з іншими алгоритмами на віртуальній системі описаній у розділі 2 при різних вхідних параметрах системи.

Використаємо такі статичні параметри системи для верифікації ефективності запропонованого алгоритму та порівняння з існуючими алгоритмами:

- 10 обчислювальних ресурсів;

- 200 задач;
- коефіцієнт продуктивності кожного ресурсу в межах від 0.9 до 1.1;
- середній час виконання однієї задачі становить 1 секунда ± 200 мс;
- час доступу до ресурсу дорівнює 5мс з флуктуацією 5мс;
- час доступу до сховища даних дорівнює також 5мс з флуктуацією 5мс.

Для комплексної оцінки ефективності необхідно перевірити систему в умовах недовантаженості (рис. 3.9) та перевантаженості (рис. 3.10), тому в нижченаведених експериментах змінювалась лише інтенсивність потоку задач та період оновлення кешу планувальника задач згідно з розрахованим оптимальним значенням.



Рис. 3.9. Порівняння загального часу виконання задач при використанні різних методів балансування навантаження grid-системи в режимі недовантаженості

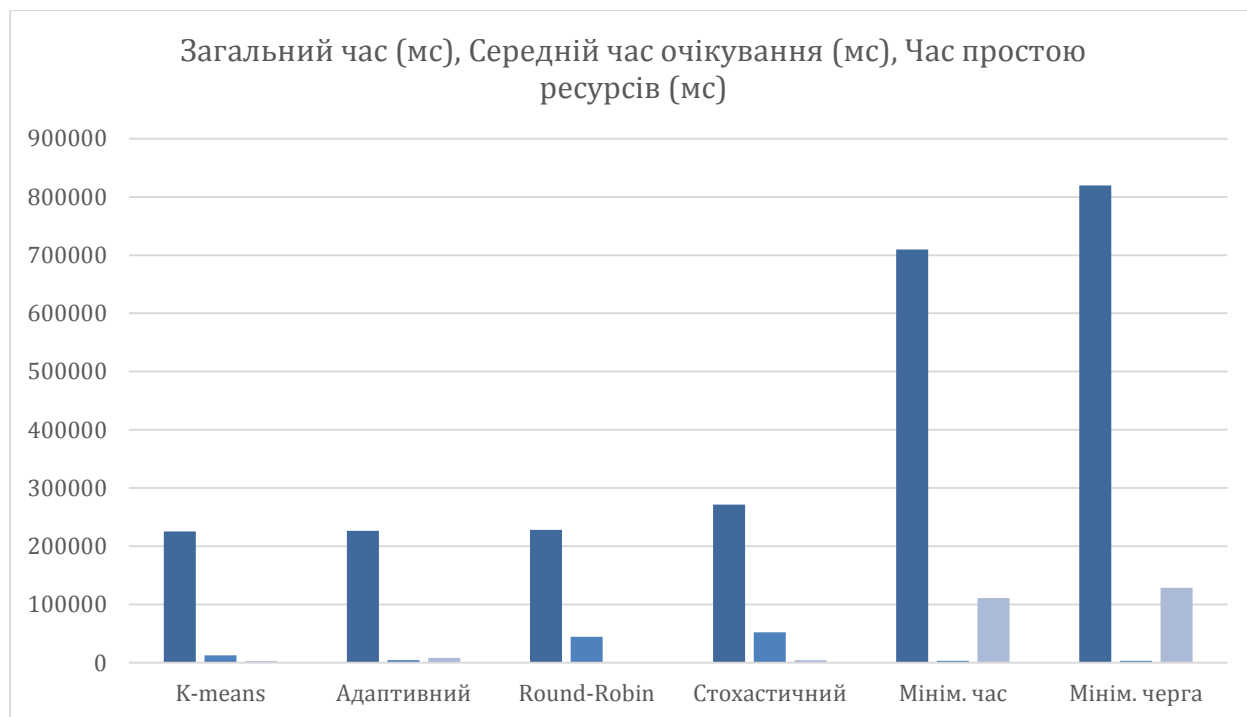


Рис. 3.10. Порівняння загального часу виконання задач при використанні різних методів балансування навантаження grid-системи в режимі перевантаженості

З наведених вище графіків видно що в алгоритмах балансування навантаження велику роль грає розподіленість обчислювальних ресурсів, адже навіть відносно незначний час затримки повідомлень між обчислювальними ресурсами, планувальником задач та сховищем даних може значно впливати як на загальний час виконання задач, так і на середній час очікування задач у черзі.

В випадку недовантаженості системи час очікування задач у черзі був мінімальний, оскільки задачі оброблювались швидше за їх подачу у загальну чергу. Відповідно, час простою ресурсів був значний. У цьому випадку в адаптивному алгоритмі використовувався великий інтервал оновлення кешу. Це сприяло підвищенню ефективності відносно інших алгоритмів за рахунок рідкого звернення планувальника до ресурсів.

В іншому випадку, коли grid-система була перевантажена, тобто задачі поступали до черги швидше ніж могли оброблятися, період оновлення кешу

був маленький. В результаті інформація про поточну завантаженість ресурсів оновлялася частіше, тому балансування відбувалось точніше.

Висновки до розділу

1. В даному розділі запропоновано варіанти покращення існуючих алгоритмів балансування навантаження та використання ефективних підходів, а саме: необхідність враховувати затримку в сигналах, якими обмінюються обчислювальні ресурси з планувальником задач та сховищем даних, пріорітизація списку ресурсів по рівню їх завантаженості та продуктивності, врахування віддаленості сховища даних від обчислювального ресурсу, а також продуктивності кожного ресурсу.
2. У запропонованому адаптивному алгоритмі реалізовані усі вищенаведені пункти та обрано оптимальні коефіцієнти моделі.
3. Усі алгоритми перевірені на розробленому емуляторі grid-в умовах недовантаженості та перевантаженості системи.
4. Виявлено що адаптивний алгоритм ефективно розподіляє задачі в обох випадках та показує гарні результати по всім критеріям, хоча іноді витрачає зайвий час на оновлення кешу, що може мати актуальне значення.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Вивчення і вирішення проблем, пов'язаних із забезпеченням здорових та безпечних умов праці людини – одне з найважливіших завдань в розробці нових технологій і систем на виробництві. Визначення причин нещасних випадків на виробництві, виникнення професійних захворювань, аварій, пожеж, і розробка заходів та вимог, спрямованих на усунення цих причин дозволяють створити безпечні та сприятливі умови праці людини. Комфортні та безпечні умови праці – один з основних факторів, що впливають на продуктивність і здоров'я працівників. Найбільш повним нормативним документом щодо забезпечення охорони праці користувачів ПК є НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації ЕОМ».

Планування робочого місця диспетчера передбачає раціональне розміщення в його кабінеті меблів, обладнання і технічних засобів, що відповідає функціональним, гігієнічним, психофізіологічним і естетичним вимогам. Крім того, воно повинно виключати витрати часу, пов'язані з недоцільним переміщенням, забезпечувати умови для раціонального виконання менеджером своїх функцій і економне використання площі службового приміщення.

4.1. Характеристики приміщення

Для кабінету диспетчера використовують кімнати з природним освітленням прямокутної форми з площею, що коливається в межах від 12 до 40 м². Для фарбування стін кабінету використовують кольори, які сприятливо впливають на нервову систему і поліпшують освітлення – слонової кістки, світло-жовтий, світло-голубий, світло-зелений, світло-бежевий. Приміщення знаходиться на 2 поверсі.

Приміщення, де передбачається використання програмного продукту (рис. 4.1) має наступні параметри:

- довжина $L = 4$ м;
- ширина $D = 3,5$ м;
- висота $H = 3,0$ м;
- загальна площа $S = L * D = 14$ м²;
- загальний об'єм $V = S * H = 42$ м³;
- кількість працюючих $N = 1$.

Аналіз робочого місця:

- площа поверхні столу – $1,12$ м²;
- глибина – $0,8$ м, ширина – $1,4$ м, висота столу – $0,75$ м;
- висота спинки та самого стільця регулюються;
- робочий стіл має простір для ніг заввишки $0,7$ м, шириною $0,8$ м;
- глибиною на рівні колін $0,48$ м, на рівні витягнутої ноги $0,7$ м;
- відстань від стіни з вікном – $1,5$ м;
- відстань від протилежної стіни – 1 м;
- відстань від екрану до ока працівника складає $0,75$ м.

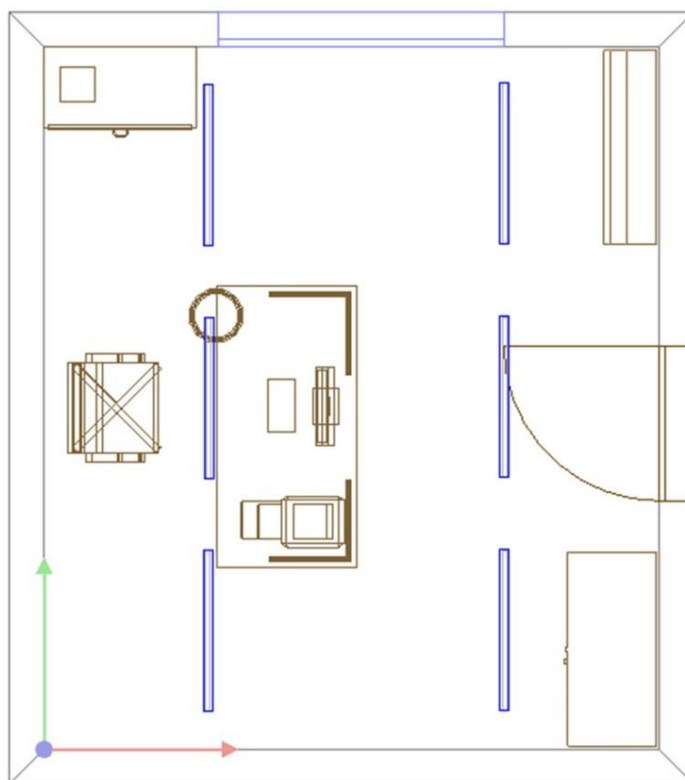


Рис. 4.1. План приміщення

Робоче місце, що розглядається, задовольняє вимогам щодо розташування і параметрів конструкції столу з ВДТ, стільця, екрану ВДТ, клавіатури.

Згідно вимог [9] до приміщень, обладнаних візуальними дисплейними терміналами ЕОМ площа на одне робоче місце повинна складати не менше ніж 6 м^2 , а об'єм – не менше ніж 20 м^3 . Площа кабінету диспетчера становить 14 м^2 , а об'єм – 42 м^3 .

Отже, дане приміщення за площею та об'ємом підпадає під норми [9].

4.2. Аналіз шкідливих виробничих факторів

4.2.1. Мікроклімат

Згідно з [10] категорія важкості робіт, що виконуються в даному приміщенні – легка Іа – це роботи, що виконуються сидячи, стоячи або

пов'язані з ходінням та супроводжуються деяким фізичним напруженням. В таблицях 4.1 та 4.2 наведені оптимальні та допустимі умови мікроклімату для виконання робіт даної категорії.

Таблиця 4.1. Оптимальні величини температури, відносної вологості та швидкості руху повітря в робочій зоні

Період року	Категорія робіт	Температура повітря	Відносна вологість, %	Швидкість руху повітря, м/сек
Холодний період року	Легка Іа	22-24	60-40	0,1
Теплий період року	Легка Іа	23-25	60-40	0,1

В приміщенні в холодний період року здійснюється централізоване опалення із низьким тиском. В теплий період використовується централізована система кондиціонування LG MULTI V+ II, яка розрахована на 20м² та має потужність 2 кВт. За рахунок цього забезпечуються оптимальні умови мікроклімату в приміщенні.

Таблиця 4.2. Допустимі умови мікроклімату для категорії робіт «легка Іа»

Період року	Температура, град, С		Відносна вологість, %	Швидкість руху повітря, м/сек
	На постійних робочих місцях	На непостійних робочих місцях		
Холодний період року	21-25	18-26	75	не більше 0,1
Теплий період року	22-28	20-30	55	0,2-0,1

Відповідно до [10] при тепловому опроміненні від 140 до 350 Вт/кв. м необхідно збільшувати на постійних робочих місцях швидкість руху повітря на 0,2 м/с більше за нормовані величини; при тепловому опроміненні, що

перевищує 350 Вт/кв. м, для легкої категорії робіт доцільно застосовувати повітряне душення робочих місць.

4.2.2. Освітлення

Нормативний рівень освітленості на робочому столі в зоні розміщення документа становить 300-500 лк згідно стандартам [11]. Освітлення в приміщеннях з ВДТ має бути суміщеним, при якому недостатнє за нормами природне освітлення доповнюється штучним. Природне освітлення повинно бути боковим, одностороннім.

Світильники загального освітлення необхідно розташовувати у вигляді ліній з боку від робочих місць паралельно лінії зору користувача. Як джерела штучного освітлення застосовуються люмінесцентні лампи, які краще поєднуються з природним освітленням.

Для створення сприятливих умов зорової роботи, виробниче освітлення повинно відповідати наступним вимогам:

- створювати на робочій поверхні освітленість, що відповідає характеру зорової роботи і знаходиться в межах встановлених норм;
- забезпечити достатню рівномірність та постійність рівня освітленості;
- не створювати на робочому місці різких і глибоких тіней;
- обмежити до мінімуму пульсацію світлового потоку;
- не зменшувати необхідний контраст фону та об'єктів, зображених на екрані дисплея.

В приміщенні є вікно з подвійним склопакетом, що виходить на північ, вікно обладнане регульованими жалюзі, а також є двері, які відкриваються всередину. Але природнього освітлення недостатньо – це обумовлено площею кімнати та орієнтацією вікна.

У приміщенні встановлено 6 світильників Philips TMS022 з 1 люмінесцентною енергозберігаючою лампою TL-D30W HFS в кожному. За паспортним даними потужність кожного складає 30 Вт. На основі аналізу стану штучного освітлення, на робоче місце припадає в середньому 416 Люкс, що відповідає нормативам.

4.2.3. Рівень шуму

Джерелом шуму в приміщенні виступають компоненти ЕОМ: вентилятори охолодження ПК, CD-дисковод; принтер та кондиціонер.

Рівень шуму охолоджувачів на всіх машинах становить 35дБА, дисководів – 46дБА; струменевий принтер – 42дБА. Принтер працює максимально 2 години на день; час роботи комп'ютера складає 8 годин; рівень шуму кондиціонера становить 48 дБА. Так як джерела шуму мають мінливий режим роботи, то необхідно розрахувати еквівалентний рівень звуку. Загальний рівень шуму складає 46,91дБА. Допустимий еквівалентний рівень звуку згідно [12] для програміста повинен бути не вище 50дБА. Наявний рівень шуму відповідає вимогам.

4.2.4. Аналіз випромінювання

ПК є значним джерелом електромагнітного випромінювання. Відповідно до [9], припустима інтенсивність потоку енергії 10Вт/м^2 , а напруженість електромагнітного поля по електричній складовій на відстані 0,5м від екрану – 10 В/м.

Сучасні комп'ютерні дисплеї мають інтенсивність потоку не більшу за 1Вт/м^2 , що відповідає нормам.

4.3. Електробезпека

Згідно з [13] за ступенем небезпеки враження електрострумом кабінет диспетчера відноситься до приміщення без підвищеної небезпеки. У приміщення проведено 1-х фазне електроживлення напругою 220 В, частотою 50 Гц та з максимальним струмом у 32 А.

Використовуються кабелі та проводи з мідними жилами перерізом не менше 2,5 мм². Застосована відкрита електропроводка в пластмасових коробах із важкогорючих матеріалів з помірною димоутворювальною здатністю.

Від електричної енергії в даному кабінеті живиться комп'ютер та штучне освітлення.

Технічні захисні заходи електробезпеки: виконана робоча ізоляція струмопровідних частин (поліетилен, пластмаса), а також застосовані пластмасові коробки з важкогорючих матеріалів з помірною димоутворювальною здатністю. Електромагнітне блокування безпеки перемикачів виконана за допомогою стержневого магніту.

Електричний роз'єм (розетка, вилка) у разі підключення до електричної мережі гарантує з'єднання корпусу з заземленням, а в разі відключення гарантує від'єднання корпусу від заземлювача в останню чергу.

4.4. Пожежна безпека

Основними причинами, які можуть призвести до виникнення пожежі в приміщенні лабораторії, є: несправності в електрообладнанні – порушення електроізоляції; несправності, що виникли внаслідок механічних пошкоджень і т.п.; несправності в обчислювальній техніці, наприклад, коротке замикання, порушення протипожежного режиму.

Предмети та матеріали, які можуть горіти: меблі – столи, стільці, шафи і т.д.; папір – документація, конструктивні елементи приміщення –

покриття підлоги, двері, віконні рами, елементи конструкції ПЕОМ та периферійних пристроїв – корпуси принтера, моніторів, клавіатури і т.п.

Відповідно до техніки безпеки користувачу ЕОМ, після закінчення роботи персональний комп'ютер і периферійні пристрої повинні бути відключені від електричної мережі. У разі виникнення аварійної ситуації необхідно негайно відключити персональний комп'ютер і периферійні пристрої від електричної мережі.

Не допускається:

- виконувати обслуговування, ремонт та налагодження персонального комп'ютеру та периферійних пристроїв безпосередньо на робочому місці оператора;
- зберігати біля персонального комп'ютеру та периферійних пристроїв папір, будь-які носії інформації (диски, флешки тощо), запасні блоки, деталі тощо, якщо вони не використовуються для поточної роботи;
- відключати захисні пристрої, самочинно проводити зміни у конструкції та складі персонального комп'ютеру та периферійних пристроїв або їх технічне налагодження;
- працювати з персональним комп'ютером, у яких під час роботи з'являються нехарактерні сигнали, нестабільне зображення на моніторі тощо.

4.4.1. Категорія приміщення за вибухопожежною небезпекою

Дане приміщення відноситься до категорії В. Згідно з [14], приміщення, що містять ЕОМ, відносяться до категорії В, як приміщення, де перебувають спалимі та важко спалимі рідини, тверді спалимі та важко спалимі речовини та матеріали (в тому числі пил та волокна), а також

речовини і матеріали, які здатні при взаємодії з водою, киснем повітря та одне з одним тільки горіти.

4.4.2. Пожежна сигналізація

В даному приміщенні встановлена система пожежної сигналізації System Sensor. Обладнання сигналізації складається з трьох основних частин наведених нижче:

- прилад приймально-контрольний пожежний АРТОН-04П. (призначений для організації централізованої і автономної охорони різних об'єктів від пожеж, шляхом цілодобового контролю стану до 4-х шлейфів пожежної сигналізації);
- телефонний комунікатор БСКТ-1 (є блоком розширення приладів приймально-контрольних пожежних «Артон-04П», «Артон-08П» і призначений для автоматичного дозвону по 8 запрограмованим номерами і передачі кодованих повідомлень про зміну стану ППКП на ПЦН або у вигляді голосових повідомлень, а також для віддаленого моніторингу та управління ППКП);
- двох сповіщувачів пожежний димових оптичних точкових СПД-3 (призначені для виявлення спалахів в закритих приміщеннях різних будівель і споруд, що супроводжуються появою диму і передачі сигналу «ПОЖЕЖА» на ППК).

Вибір пожежних сповіщувачів здійснюється в залежності від характерних приміщень, виробництв, технологічних процесів. Згідно з цим, приміщення обладнане адресованими оповіщувачами.

4.4.3. Засоби гасіння пожежі

Оскільки основним можливим джерелом загоряння у приміщенні є електрообладнання, використовувати воду для пожежогасіння категорично заборонено. Натомість варто використовувати порошкове пожежогасіння, яке працює разом з системою автоматичного виявлення та сповіщення про пожежу. Встановлена установка модульного типу, розташована безпосередньо у приміщенні.

Оскільки площа приміщення дорівнює 14 кв. м., можна використати 1 вогнегасник порошковий ВП-3 з масою заряду 3 кг з урахуванням 3 кг. на 20 м. кв. Вогнегасник розміщено біля вхідної двері та прикріплено до стіни.

4.5. Техніка безпеки

Напруга живлення офісної техніки (220 В) є небезпечною для життя людини. Тому, незважаючи на те що в конструкції комп'ютера та принтера передбачена достатня ізоляція від струмопровідних ділянок, необхідно знати та чітко виконувати ряд правил техніки безпеки.

Забороняється:

- знаходитися на робочому місці у верхньому одязі;
- класти одяг і сумки на стіл;
- зберігати на робочому місці напої та їжу;
- працювати на комп'ютері у вологому одязі та вологими руками;
- розташовуватися збоку або ззаду від включеного монітора;
- торкатись екрана, тильного боку дисплея, проводів живлення, заземлення, з'єднувальних кабелів
- пересувати комп'ютери і монітори, знімати кришку корпуса системного блоку;

- самостійно намагатися усунути будь-які неполадки в роботі комп'ютера, незалежно від того, коли і з чийої вини вони сталися;
- перекривати вентиляційні отвори на системному блоці та моніторі;
- класти речі на клавіатуру, монітор і системний блок.

Під час роботи на комп'ютері необхідно:

- забезпечити відстань від екрану до очей – 60-70 см. (відстань витягнутої руки);
- вертикально пряма спина, плечі опущені і розслаблені, ноги на підлозі і не схрещені, лікті, зап'ястя і кисті рук на одному рівні, ліктьові, тазостегнові, колінні, гомілковостопні суглоби під прямим кутом.

4.6. План ліквідації надзвичайної ситуації

У ході роботи потрібно створити план локалізації та ліквідації аварійних ситуацій, які можуть виникнути під час роботи на підприємстві.

4.6.1. Можливі аварійні ситуації

В залежності від масштабності наслідків аварійної ситуації розрізняють наступні рівні:

Надзвичайна ситуація загальнодержавного рівня – це надзвичайна ситуація, яка розвивається на території двох та більше областей (Автономної республіки Крим, міст Києва та Севастополя) або загрожує транскордонним перенесенням, а також у разі, коли для її ліквідації необхідні матеріали і технічні ресурси в обсягах, що перевищують власні можливості окремої області (Автономної республіки Крим, міст Києва та Севастополя), але не менше 1% обсягів видатків відповідного бюджету.

Надзвичайна ситуація регіонального рівня – це надзвичайна ситуація, яка розвивається на території двох або більше адміністративних районів

(міст обласного значення), Автономної республіки Крим, областей, міст Києва та Севастополя або загрожує перенесенням на територію суміжної області, а також у разі, коли для її ліквідації необхідні матеріальні і технічні ресурси в обсягах, що перевищують власні можливості окремого району, але не менше 1% обсягів видатків відповідного бюджету.

Надзвичайна ситуація місцевого рівня – це надзвичайна ситуація, яка виходить за межі потенційно-небезпечного об'єкта, загрожує поширенням самої ситуації або її вторинних наслідків на довкілля, сусідні населені пункти, інженерні споруди, а також у разі, коли для її ліквідації необхідні матеріальні і технічні ресурси в обсягах, що перевищують власні можливості об'єкта. До місцевого рівня також належать всі надзвичайні ситуації, які виникають на об'єктах житлово-комунальної сфери.

Надзвичайна ситуація об'єктового рівня – це надзвичайна ситуація, яка не підпадає під зазначені вище визначення, тобто така, що розгортається на території об'єкта або на самому об'єкті, її наслідки не виходять за межі об'єкта або його санітарно-захисної зони.

В нашому випадку існує можливість радіаційного забруднення під час аварійної ситуації комунального рівня на атомній електростанції.

В Україні діють 5 атомних електростанцій з 16 енергетичними ядерними реакторами, 2 дослідних ядерних реактори та більше 8 тис. підприємств і організацій, які використовують у виробництві, науково-дослідній роботі та медичній практиці різноманітні радіоактивні речовини, а також зберігають і переробляють радіоактивні відходи. Найближчою АЕС до міста Київ є Чорнобильська АЕС з реакторами типу РБМК. На даний момент вона повністю виведена з експлуатації, проте на території Росії до цього часу використовується 11 реакторів цього типу.

4.6.2. Оцінка радіаційної обстановки в зоні радіаційного забруднення

Потрібно розрахувати радіаційну обстановку в зоні радіаційного забруднення в результаті аварії на реакторі типу РБМК. Оцінку будемо проводити згідно з інструкціями в [15].

Аварія сталась о 3:00. Вимірний рівень радіації на 3:30 становить 45 р/год. Астрономічний час початку робіт щодо ліквідації наслідків аварії – 5:00. Тривалість робіт оцінюється в 3.5 години. Допустима доза радіації становить 20 радіан. Коефіцієнт ослаблення радіації – 3.

Всі розрахунки для оцінки радіаційного стану прийнято проводити у відносному часі. Точкою відліку вважається момент аварії: $t_{\text{астрон}} = 3:00$, $t_{\text{відн}} = 0$. З огляду на це, відносний час початку роботи є наступним: $t_{\text{пастрон}} = 5:00$, $t_{\text{пвідн}} = 2$. Слід також розрахувати час кінця роботи:

$$t_{\text{квідн}} = t_{\text{пвідн}} + T = 2 + 3,5 = 5,5 \quad (4.3)$$

Для наступних розрахунків слід вирахувати рівень радіації на першу годину після аварії. Для цього маючи коефіцієнт для перерахунку рівнів радіації на рівний час після аварії на АЕС $K_{0,5\text{н}} = 0,81$ та вимірний на цей час рівень радіації – 45 Р/год по формулі 4.4 знайдемо шукане значення.

$$P_1 = P_{\text{вим}} \cdot K_{t_{\text{вим}}} \quad (4.4)$$

За формулою 4.4 маємо, що рівень радіації на першу годину після аварії – 36,45 Р/год.

Далі розрахуємо дозу отриманої радіації під час роботи. Розрахунки будемо проводити за формулою 4.5.

$$D = \frac{P_{\text{ср}} \cdot t_{\text{р}}}{K_{\text{осл}}} \quad (4.5)$$

Для використання формули 4.4 слід розрахувати значення середнього випромінювання, яке вимірюється як середнє арифметичне рівня радіації

на початок робіт та на кінець робіт. Рівень радіації на певний момент розраховується за формулою 4.6.

$$P_t = \frac{P_1}{K_{t_H}} \quad (4.6)$$

На початок ліквідаційних робіт рівень радіації – 21,96 Р/год, на кінець ліквідаційних робіт – 6,63 Р/год. Середній рівень радіації – 14,29 Р/год.

Провівши необхідні обчислення за формулою 4.4 отримаємо, що доза отриманого при роботі випромінювання рівна 16,67 Р.

Далі слід розрахувати допустимий час роботи на зараженому об'єкті. Для цього слід розрахувати коефіцієнт α . Для цього скористаємось формулою 4.7.

$$\alpha = \frac{P_1}{K_{\text{осл}} \cdot D_{\text{доп}}} \quad (4.7)$$

Вирахувана величина α рівна 0,6. Маючи її значення та час початку роботи з ліквідації $t_{\text{пвідн}} = 2$ скориставшись графіком на рис. 4.5 виміряємо допустимий час роботи на реакторі.

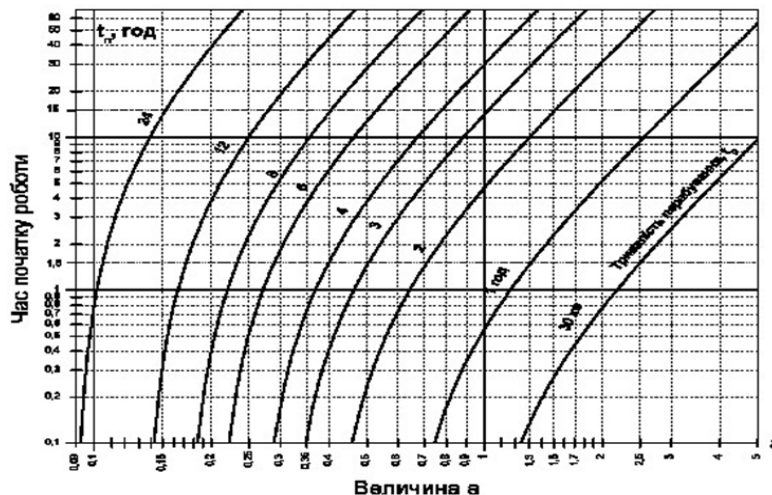


Рис. 4.5. Графік визначення тривалості перебування в зоні радіоактивного зараження при аварії на АЕС з реактором РБМК-1000

За графіком бачимо, що в даних умовах допустимий час роботи рівний 2 години 20 хвилин.

Отже, при роботі на даній АЕС слід обов'язково проводити інструктаж з техніки безпеки та завжди мати наготові службу екстреного реагування на випадок аварійної ситуації. Також забезпечити весь особовий склад необхідною захисною екіпіровкою.

4.7. Техніка безпеки після закінчення роботи

Після закінчення роботи на ЕОМ диспетчер повинен дотримуватись наступних правил:

1. після закінчення операцій з ЕОМ необхідно закрити усі програмні додатки, що виконувались на персональному комп'ютері;
2. по завершенню роботи із комп'ютером необхідно завершити його роботу та вийняти вилку з розетки. Якщо робота буде продовжена за машиною, то необхідно завершити сеанс користувача та не вимикати ЕОМ;
3. у кінці дня необхідно перед виходом з приміщення закрити вікно та знеструмити кабінет з головного рубильника;
4. про всі недоліки, що виявились у процесі роботи повідомити керівника робіт та інженера з охорони праці.

Висновки до розділу

1. У ході виконання розділу з охорони праці проведено аналіз шкідливих та небезпечних виробничих факторів при роботі з ЕОМ.
2. Спроековано найбільш підходяще робоче місце для роботи диспетчера, що забезпечене раціональним освітленням, розраховані показники освітленості робочої області, що відповідає всім вимогам [9].
3. Проведені всі необхідні міри, щодо забезпечення приміщення засобами пожежної безпеки: пожежною сигналізацією та засобами гасіння пожежі.

ЗАГАЛЬНІ ВИСНОВКИ

1. Grid-системи дозволяють зберігати, оброблювати та збирати інформацію на різноманітних розподілених ресурсах, що можуть бути географічно розподілені, належати різним організаціям та використовуватися для вирішення різноманітних масштабних задач.
2. Не існує єдиного алгоритму планування задач в grid-системах, який би задовольняв усі потреби та був ефективним на всіх grid-системах з різними додатками, інструментарієм та функціями. А існуючі алгоритми балансування навантаження мають недоліки які можуть бути усунені врахуванням наступних параметрів: різноманітність системи, віддаленість обчислювальних ресурсів, їх продуктивність та відсутність необхідності часто опитувати ресурси про їх стан.
3. В алгоритмах балансування навантаження велику роль грає віддаленість та гетерогенність обчислювальних ресурсів, що досить сильно впливає на загальний час виконання задач, на середній час очікування задач у черзі та на час простою ресурсів.
4. У випадку близького розташування обчислювальних ресурсів динамічні алгоритми є більш ефективними за статичні, оскільки обчислення оптимального ресурсу для виконання задачі відбувається миттєво, в той час як навантаження на ресурси є збалансованим.
5. При великій розподіленості системи кожен запит до обчислювального ресурсу та до сховища даних займає деякий час. Тому більш ефективними є статичні методи.
6. Розроблено емулятор grid-системи, що дає можливість проаналізувати ефективність кожного алгоритму балансування навантаження завдяки гнучкому конфігуруванню параметрів системи.

7. Запропонований адаптивний алгоритм враховує усі недоліки існуючих алгоритмів, а саме: затримка в повідомленнях, якими обмінюються обчислювальні ресурси з планувальником задач та сховищем даних, кешування інформації про стан ресурсів, пріорітизація списку ресурсів по рівню їх завантаженості, врахування віддаленості сховища даних, а також продуктивності кожного ресурсу.
8. Виявлено що при високій інтенсивності потоку задач варто проводити оновлення кешу адаптивного алгоритму якомога рідше. Це пов'язано з тим що ресурси знаходяться в режимі спокою та готові приймати нові задачі на виконання з мінімальною затримкою.
9. При перевантаженості системи виникають черги на виконання та виникає необхідність розподіляти задачі на більш вільні ресурси. У зв'язку з цим потрібно оновляти кеш інформації про стан ресурсів якомога частіше. Це призведе до більшої ефективності розподілу задач та мінімізує загальний час їх обробки.
10. У ході виконання розділу з охорони праці проведено аналіз шкідливих та небезпечних виробничих факторів при роботі з ЕОМ та спроектовано сприятливе робоче місце для роботи диспетчера, проведені всі необхідні міри щодо забезпечення приміщення засобами пожежної безпеки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Информационные технологии в туризме [Электронный ресурс]: учеб. пособие / В.Н. Бочарников, Е.Г. Лаврушина, Я.Ю. Блиновская. 2-е изд., стер. – М. : ФЛИНТА, 2013. – 358 с.
2. Мирошник М.А. Размещение подзадач в распределенных вычислительных системах кластерно-метакомпьютерного типа. / М.А. Мирошник, Л.А. Клименко // Інформаційно – керуючі системи на залізничному транспорті. – 2014. – №4. – С. 71-77.
3. Топорков В.В. Моделі розподілених обчислень. – М.: Физматлит. 2014 – 320с.
4. В. Кореньков, Н. Кутовский. Инфраструктура обучения grid-технологиям // Открытые системы. 2009. № 10.
5. Петренко А.І. Національна Grid — інфраструктура для забезпечення наукових досліджень і освіти. — <http://netalltd.cad.kiev.ua/downloads/Grid.pdf>.
6. А.С. Хританков. «Модели и алгоритмы распределения нагрузки. Модель коллектива вычислителей. Модели с соперником». Информационные технологии и вычислительные системы, №2 2009, с.65-80.
7. Аветисян А.И., Гайсарян С.С., Грушин Д.А., Кузюрин Н.Н., Шокуров А.В. Эвристики распределения задач для брокера ресурсов Grid // Тр. Ин-та системного программирования РАН. — 2004. — Т. 5. — С. 269–280.
8. Полежаев П.Н. Экспериментальное исследование алгоритмов планирования задач для вычислительной грид-системы // Системы управления и информационные технологии, №3.2(45), 2011. – с. 266–270.

9. НПАОП 0.00-1.28-10. Правила охорони праці під час експлуатації електронно-обчислювальних машин : затверджені наказом Держкомітету України з промислової безпеки, охорони праці та гірничого нагляду від 26.03.2010 № 65 [Електронний ресурс]. – Режим доступу: <http://dnop.com.ua>
- 10.ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень. – Київ, 2000
- 11.ДБН.В2.5.-28-2006. Штучне та природне освітлення [Електронний ресурс]. – Режим доступу : <http://dnop.com.ua/dnaop/act8425.htm>
- 12.ДНС 3.3.6.037-99. Санітарні норми виробничого шуму, ультразвуку та інфразвуку МОЗ України [Електронний ресурс]. – Режим доступу : <http://dnop.com.ua/dnaop/act4878.htm>.
- 13.ПУЕ-2009. Правила улаштування електроустановок вид. 3-тє, перероб. і доп. – 736 с.
- 14.НАПБ Б.03.002-2007. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою : Затвердж. наказом МНС від 03.12.2007 № 833 – Київ, 2007.
- 15.Стеблюк М.І. Цивільна оборона та цивільний захист: Підручник. – 2-ге вид., переробл. – К.: Знання, 2009 р. – 487 с.

ДОДАТОК А
ВІДОМІСТЬ МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ

ВІДОМІСТЬ МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	A4		Завдання на дипломний проект	2	
2	A4		Пояснювальна записка	90	
3	A4		Лістинг коду розробленого емулятора grid-системи	9	
3	A3		Архітектура обчислювального середовища grid	1	
4	A3		Візуалізація алгоритмів балансування навантаження	1	
5	A3		Блок-схема алгоритму роботи емулятора grid-системи	1	
6	A3		Архітектура системи планування задач у grid-системах	1	
7	A3		Графіки порівняння результатів роботи різних алгоритмів планування	1	
8	A3		Діаграма класів розробленого емулятора grid-системи	1	

ДОДАТОК Б
ЛІСТИНГ КОДУ РОЗРОБЛЕНОГО ЕМУЛЯТОРА
GRID-СИСТЕМИ

Клас GridEmulator:

```
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import java.util.List;

public class GridEmulator {

    public static void main(String[] args) throws Exception {
        LoadBalancer.Rule[] rules = new LoadBalancer.Rule[]{
            LoadBalancer.Rule.RANDOM,
            LoadBalancer.Rule.ROUND_ROBIN,
            LoadBalancer.Rule.MIN_EXECUTION,
            LoadBalancer.Rule.MIN_QUEUE,
            LoadBalancer.Rule.K_MEANS,
            LoadBalancer.Rule.ADAPTIVE};

        int experiments = 10;

        int amountOfTasks = 200;
        int amountOfResources = 10;

        double resourcesFluctuationRate = 0.1d;
        int minTaskDuration = 800;
        int maxTaskDuration = 1200;
        int cacheUpdateInterval = 2000;

        int maxPingTime = 10;
        int maxDataAccessTime = 10;

        int queueAppendingInterval = 200; // 100 by default

        String reportHeader = "Алгоритм, Загальний час (мс), Середній  

        час очікування (мс), Максимальний час очікування (мс), Середній час  

        простою (мс), Максимальний час простою (мс)";
        String fileName = String.format("./s.csv",
            String.format("%d_%dres_%dtasks_%ddur_withoutping",
                System.currentTimeMillis(), amountOfResources, amountOfTasks,
                (minTaskDuration + maxTaskDuration) / 2));
        BufferedWriter bufferedWriter = new BufferedWriter(new
            OutputStreamWriter(new FileOutputStream(fileName), "UTF-8"));
        bufferedWriter.write(reportHeader);
        bufferedWriter.newLine();

        for (int i = 1; i <= experiments; i++) {
            List<Resource> resources =
                Utils.generateResources(amountOfResources, resourcesFluctuationRate,
                    maxPingTime);
            System.out.println(resources);
            List<Task> tasks = Utils.generateTasks(amountOfTasks,
                minTaskDuration, maxTaskDuration, resources, maxDataAccessTime);
            resources.forEach(Resource::start);

            for (final LoadBalancer.Rule rule : rules) {
                tasks.forEach(Task::reset);
                resources.forEach(Resource::reset);
            }
        }
    }
}
```

```

        long[] results = LoadBalancer.executeAllTasks(rule,
tasks, resources, cacheUpdateInterval, queueAppendingInterval);

        String resultString = rule.toString() + ", " +
results[0] + ", " + results[1] + ", " + results[2] + ", " + results[3]
+ ", " + results[4];
        bufferedWriter.write(resultString);
        bufferedWriter.newLine();
        System.out.println(String.format("Experiment %d/%d
completed with results: %s", i, experiments, resultString));
    }
    resources.forEach(Resource::shouldBeTerminated);
}
bufferedWriter.close();
}
}

```

Клас LoadBalancer:

```

import sun.reflect.generics.reflectiveObjects.NotImplementedException;

import java.util.*;

public class LoadBalancer {

    private static final Map<Resource, Long>
resourcesQueueDurationCache = new HashMap<>();
    public static long resourceCacheUpdateTime;
    private static int roundRobinLastAssignedResourceId = -1;

    public static long[] executeAllTasks(LoadBalancer.Rule rule,
List<Task> tasks, List<Resource> resources,
                                long cacheUpdateInterval,
long queueAppendingInterval) throws InterruptedException {
        long startTime = System.currentTimeMillis();
        for (Task task : tasks) {
            task.addToTheQueue();
            getTargetResource(rule, task, resources,
cacheUpdateInterval).executeTask(task);
            Thread.sleep(queueAppendingInterval);
        }
        while (tasks.stream().filter(Task::isCompleted).count() !=
tasks.size()) {
            Thread.sleep(25);
        }
        long endTime = System.currentTimeMillis();

        long duration = endTime - startTime;

        final long maxWaitTime = tasks.stream().max((t1, t2) ->
Long.compare(t1.getWaitTime(), t2.getWaitTime())).get().getWaitTime();
        long avgWaitTime = 0;
        for (Task task : tasks) avgWaitTime += task.getWaitTime();
        avgWaitTime /= tasks.size();
    }
}

```

```

        final long maxIdleTime = resources.stream().max((r1, r2) ->
Long.compare(r1.getIdle(), r2.getIdle())).get().getIdle();
        long avgIdleTime = 0;
        for (Resource resource : resources) avgIdleTime +=
resource.getIdle();
        avgIdleTime /= resources.size();

        return new long[]{duration, avgWaitTime, maxWaitTime,
avgIdleTime, maxIdleTime};
    }

    public static Resource getTargetResource(Rule rule, Task task,
List<Resource> resources, long cacheUpdateInterval) {
        switch (rule) {
            case RANDOM:
                return getResourceByRandom(resources);
            case ROUND_ROBIN:
                return getResourceByRoundRobin(resources);
            case MIN_EXECUTION:
                return getResourceByMinExecution(resources);
            case MIN_QUEUE:
                return getResourceByMinQueueSize(resources);
            case K_MEANS:
                return getResourceByKMeans(resources);
            case ADAPTIVE:
                return getResourceAdaptively(task, resources,
cacheUpdateInterval);
            default:
                throw new NotImplementedException();
        }
    }

    private static Resource getResourceByRandom(List<Resource>
resources) {
        return resources.get((int) (Math.random() *
resources.size()));
    }

    private static Resource getResourceByRoundRobin(List<Resource>
resources) {
        return resources.get(++roundRobinLastAssignedResourceId <
resources.size() ?
roundRobinLastAssignedResourceId :
(roundRobinLastAssignedResourceId = 0));
    }

    private static Resource getResourceByMinExecution(List<Resource>
resources) {
        Resource resourceWithMinQueueDuration = resources.get(0);
        for (Resource resource : resources) {
            if (resource == resourceWithMinQueueDuration) continue;
            if (resource.getQueueDuration() <
resourceWithMinQueueDuration.getQueueDuration()) {
                resourceWithMinQueueDuration = resource;
            }
        }
    }

```

```

        return resourceWithMinQueueDuration;
    }

    private static Resource getResourceByMinQueueSize(List<Resource>
resources) {
        Resource resourceWithMinQueue = resources.get(0);
        for (Resource resource : resources) {
            if (resource == resourceWithMinQueue) continue;
            if (resource.getQueueSize() <
resourceWithMinQueue.getQueueSize()) {
                resourceWithMinQueue = resource;
            }
        }
        return resourceWithMinQueue;
    }

    private static Resource getResourceByKMeans(List<Resource>
resources) {
        Collections.shuffle(resources);
        Resource resourceWithMinQueue = resources.get(0);
        for (int i = 0; i < resources.size() / 2; i++) {
            Resource resource = resources.get(i);
            if (resource == resourceWithMinQueue) continue;
            if (resource.getQueueDuration() <
resourceWithMinQueue.getQueueDuration()) {
                resourceWithMinQueue = resource;
            }
        }
        return resourceWithMinQueue;
    }

    private static Resource getResourceAdaptively(Task task,
List<Resource> resources, long cacheUpdateInterval) {
        if (System.currentTimeMillis() - resourceCacheUpdateTime >
cacheUpdateInterval ||
            resourceCacheUpdateTime == 0) {
            updateCache(resources);
            resourceCacheUpdateTime = System.currentTimeMillis();
        }

        Resource targetResource =
resourcesQueueDurationCache.keySet().stream().findFirst().get();
        long targetResourceFutureQueueDuration = (long)
(resourcesQueueDurationCache.get(targetResource) +
            task.getDataAccessTime(targetResource) +
            task.getExecutionTime() / targetResource.getProcessingRate());

        for (Resource resource : resourcesQueueDurationCache.keySet())
        {
            if (resource == targetResource) continue;
            long resourceFutureQueueDuration = (long)
(resourcesQueueDurationCache.get(resource) +
                task.getDataAccessTime(resource) +
                task.getExecutionTime() / resource.getProcessingRate());
            if (resourceFutureQueueDuration <
targetResourceFutureQueueDuration) {

```

```

        targetResource = resource;
        targetResourceFutureQueueDuration =
resourceFutureQueueDuration;
    }
}
Random random = new Random();
resourcesQueueDurationCache.put(targetResource, (long)
(targetResourceFutureQueueDuration * (0.75 + random.nextDouble() *
0.5)));
return targetResource;
}

private static void updateCache(List<Resource> resources) {
    resourcesQueueDurationCache.clear();
    for (Resource resource : resources) {
        resourcesQueueDurationCache.put(resource,
resource.getQueueDuration());
    }
}

enum Rule {
    RANDOM, ROUND_ROBIN, MIN_EXECUTION, MIN_QUEUE, K_MEANS,
ADAPTIVE
}
}

```

Клас Resource:

```

import java.util.Queue;
import java.util.concurrent.LinkedBlockingQueue;

public class Resource extends Thread {

    private static int nextId;
    private final Queue<Task> queue;
    private int id;
    private boolean shouldBeTerminated;

    private double processingRate;
    private long pingTime;

    private long idleStartTime;
    private long idle;

    public Resource(double processingRate, long pingTime) {
        id = nextId++;
        queue = new LinkedBlockingQueue<>();
        shouldBeTerminated = false;
        this.processingRate = processingRate;
        this.pingTime = pingTime;
    }

    public void executeTask(Task task) {
        sleep();
        synchronized (queue) {

```

```

        queue.add(task);
    }
}

@Override
public void run() {
    idleStartTime = System.currentTimeMillis();
    while (!shouldBeTerminated) {
        Task task;
        synchronized (queue) {
            task = queue.peek();
        }
        if (task != null) {
            if (idleStartTime != 0) {
                idle += (System.currentTimeMillis() -
idleStartTime);
                idleStartTime = 0;
            }
            task.execute(this);
            synchronized (queue) {
                queue.remove(task);
                if (queue.isEmpty()) {
                    idleStartTime = System.currentTimeMillis();
                }
            }
        }
    }
}

public boolean isBusy() {
    sleep();
    return queue.size() != 0;
}

public int getQueueSize() {
    sleep();
    return queue.size();
}

public long getIdle() {
    return idle;
}

public void shouldBeTerminated() {
    shouldBeTerminated = true;
}

public long getQueueDuration() {
    sleep();

    long timeLeft = 0;
    for (Task task : queue) {
        timeLeft += task.getExecutionTime();
    }
    return timeLeft;
}

```

```

    public double getProcessingRate() {
        return processingRate;
    }

    private void sleep() {
        if (pingTime != 0) {
            try {
                Thread.sleep(pingTime);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public String toString() {
        return String.format("Resource[%d]: {perf=%f, pingTime=%d}",
            id, processingRate, pingTime);
    }

    public void reset() {
        idle = 0;
        idleStartTime = 0;
    }
}

```

Клас Task:

```

import java.util.HashMap;
import java.util.Map;

public class Task {

    private static int nextId;
    private int id;
    private long executionTime;

    private Map<Resource, Integer> resourceDataAccessTime = new
HashMap<>();
    private long startTime;
    private long endTime;
    private long waitTime;

    public Task(int executionTime) {
        id = nextId++;
        this.executionTime = executionTime;
    }

    public void addedToTheQueue() {
        startTime = System.currentTimeMillis();
    }

    public long getWaitTime() {
        return waitTime;
    }
}

```

```

    public boolean isCompleted() {
        return endTime != 0;
    }

    public long getExecutionTime() {
        return executionTime;
    }

    public void execute(Resource resource) {
        if (startTime == 0) throw new IllegalStateException("Task
should be added to the queue first!");
        waitTime = System.currentTimeMillis() - startTime;
        try {
            Thread.sleep(getDataAccessTime(resource) + (long)
(executionTime / resource.getProcessingRate()));
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        endTime = System.currentTimeMillis();
    }

    @Override
    public String toString() {
        return "Task[" + id + "]:{" + executionTime + "}";
    }

    public void reset() {
        startTime = 0;
        endTime = 0;
        waitTime = 0;
    }

    public int getDataAccessTime(Resource resource) {
        if (resourceDataAccessTime != null && resource != null) {
            return resourceDataAccessTime.get(resource);
        } else return 0;
    }

    public void setResourceDataAccessTime(Map<Resource, Integer>
resourceDataAccessTime) {
        this.resourceDataAccessTime = resourceDataAccessTime;
    }
}

```

Клас Utils:

```

import java.util.*;

public class Utils {

    static List<Task> generateTasks(int amount, int minTaskDuration,
int maxTaskDuration, List<Resource> resources,
                                int maxAccessTime) {
        ArrayList<Task> tasks = new ArrayList<>(amount);
        Random random = new Random();
    }
}

```

```

        for (int i = 0; i < amount; i++) {
            Task task = new Task(random.nextInt(maxTaskDuration -
minTaskDuration) + minTaskDuration);
            Map<Resource, Integer> dataAccessTime = new HashMap<>();
            for (Resource resource : resources) {
                dataAccessTime.put(resource,
random.nextInt(maxAccessTime));
            }
            task.setResourceDataAccessTime(dataAccessTime);
            tasks.add(task);
        }
        return tasks;
    }

    static List<Resource> generateResources(int amount, double
fluctuationRate, int maxPingTime) {
        ArrayList<Resource> resources = new ArrayList<>(amount);
        Random random = new Random();
        for (int i = 0; i < amount; i++) {
            Resource resource = new Resource(1 - fluctuationRate + 2f
* fluctuationRate * random.nextDouble(), random.nextInt(maxPingTime));
            resources.add(resource);
        }
        LoadBalancer.resourceCacheUpdateTime = 0;
        return resources;
    }
}

```

ДОДАТОК В

АРХІТЕКТУРА ОБЧИСЛЮВАЛЬНОГО GRID-СЕРЕДОВИЩА

ДОДАТОК Г

ВІЗУАЛІЗАЦІЯ АЛГОРИТМІВ БАЛАНСУВАННЯ

НАВАНТАЖЕННЯ

ДОДАТОК Д

БЛОК-СХЕМА АЛГОРИТМУ РОБОТИ ЕМУЛЯТОРА GRID-
СИСТЕМИ

ДОДАТОК Е

АРХІТЕКТУРА СИСТЕМИ ПЛАНУВАННЯ ЗАДАЧ У GRID-
СИСТЕМАХ

ДОДАТОК Ж

ГРАФІКИ ПОРІВНЯННЯ РЕЗУЛЬТАТІВ РОБОТИ РІЗНИХ

АЛГОРИТМІВ ПЛАНУВАННЯ

ДОДАТОК К
ДІАГРАМА КЛАСІВ РОЗРОБЛЕНОГО ЕМУЛЯТОРА GRID-
СИСТЕМИ