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

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Проміжне програмне забезпечення розподілених систем зі спеціалізованим функціоналом
Назва теми

КвРКІ 101065.21.01.15 ПЗ
Шифр

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

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

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

Виконав: студент III курсу, група КІ2с-21-1 _____ Н. В. Шаварський
Підпис, Ініціали, прізвище

Керівник _____ Д. М. Медзатий
Підпис, дата, Ініціали, прізвище

Нормоконтролер _____ І.О. Засорнова
Підпис, дата, Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис, Ініціали, прізвище

«19» червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

"10" 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Шаварському Назару Віталійовичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Проміжне програмне забезпечення розподілених систем зі спеціалізованим функціоналом

Керівник проекту (роботи) Медзатий Д.М., к.т.н., доцент.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.02.2024 р. № 8

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

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

Дослідження предметної області її структурних та функціональних особливостей

Проєктування програмно-технічного засобу

Програмно-апаратна реалізація та тестування програмно технічного засобу

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

Схема інтерфейсу передачі повідомлень (MPI) у кластерній системі

Блок-схема алгоритму програми

Загальна діаграма роботи програми

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Засорнова І.О. к.т.н., доцент		
Антиплагіат	Нічепорук А.О., доцент кафедри КІПС		

7. Дата видачі завдання «_10_»_____01_____2024 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2024	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2024	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2024	виконано
4	Робота над розділом 2 – проєктування програмного засобу	01.04.2024	виконано
5	Робота над розділом 3 – програмно-апаратна реалізація та тестування програмно-технічного засобу	29.04.2024	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2024	виконано
7	Попередній захист ВКР	30.05.2024	виконано
8	Захист ВКР на засіданні ЕК	Червень 2024 року	

Студент _____ Н.В. Шаврарський
Підпис Ініціали, прізвище

Керівник роботи _____ Д.М. Медзатий
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 101065.21.01.15 ПЗ	Пояснювальна записка	58		
			Графічні матеріали			
2		КвРКІ 101065.21.01.15 Е8	Схема інтерфейсу передачі повідомлень (MPI) у кластерній системі	1		
3		КвРКІ 101065.21.01.15 Е8	Блок-схема алгоритму програми	1		
4		КвРКІ 101065.21.01.15 Е8	Загальна діаграма роботи програми	1		

КвРКІ 101065.21.01.15 ВП

Зм	Арк	№ докум	Підпис	Дата	Відомість проекту	Літера	Аркуш	Аркушів
Розробив		Шаварський		13.06		У	1	1
Перевір.		Медзатий		13.06				
Н. контр.		Засорнова		14.06		ХНУ, КІ2с-21-1		
Затв.		Говорущенко		19.06				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Проміжне програмне забезпечення розподілених систем зі спеціалізованим функціоналом».

Автор роботи: Шаварський Назар Віталійович

Керівник роботи: Медзатий Дмитро Миколайович

Пояснювальна записка: 58 с., 17 рис., 3 табл., 61 джерело.

Графічна частина: Схема інтерфейсу передачі повідомлень (MPI) у кластерній системі, Блок-схема алгоритму програми, Загальна діаграма роботи програми

ПРОМІЖНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, РОЗПОДІЛЕНІ СИСТЕМИ, СПЕЦІАЛЬНИЙ ФУНКЦІОНАЛ

Метою роботи є дослідження та розробка проміжного програмного забезпечення (middleware) для розподілених систем, яке має спеціалізований функціонал.

У цій роботі розроблено проміжне програмне забезпечення для розподілених систем зі спеціалізованим функціоналом. Розроблено архітектурні рішення, що дозволяють ефективно взаємодіяти між компонентами розподілених систем. Також розроблено інтеграційні методи, спрямовані на забезпечення надійності та продуктивності середовища. Результати досліджень ідентифікували ключові функції проміжного програмного забезпечення і його переваги для розподілених систем.

_____ 30.05.2024

Підпис студента Дата

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі інформаційних технологій розподілені системи набувають все більшого значення, забезпечуючи основу для різноманітних додатків і сервісів, від хмарних обчислень до Інтернету речей (IoT).

Проміжне програмне забезпечення (middleware) є ключовим компонентом цих систем, оскільки воно забезпечує взаємодію між різними програмними і апаратними компонентами, полегшуючи інтеграцію та управління розподіленими ресурсами. Актуальність теми розробки та впровадження проміжного програмного забезпечення зі спеціалізованим функціоналом обумовлена кількома важливими аспектами.

По-перше, зростаюча складність і гетерогенність сучасних розподілених систем вимагає більш ефективних рішень для забезпечення надійної та безперебійної роботи.

Традиційні підходи часто не справляються з такими викликами, як масштабованість, безпека, адаптивність і оптимізація використання ресурсів. Спеціалізоване проміжне програмне забезпечення, розроблене для конкретних сценаріїв і додатків, дозволяє більш точно вирішувати ці питання, пропонуючи оптимізовані алгоритми і протоколи, які враховують особливості конкретного середовища.

По-друге, розвиток нових технологій, таких як машинне навчання, обробка великих даних і блокчейн, створює додаткові вимоги до інфраструктури розподілених систем.

Спеціалізоване проміжне програмне забезпечення може включати функціональні можливості для підтримки цих технологій, що дозволяє ефективно обробляти великі обсяги даних, забезпечувати динамічну маршрутизацію і управління даними, а також гарантувати безпеку і прозорість транзакцій.

По-третє, зростаюча популярність IoT і мобільних обчислень створює потребу у високоефективних і низькоенергетичних рішеннях для управління великими кількостями пристроїв і сенсорів. Проміжне програмне забезпечення зі

спеціалізованим функціоналом може забезпечити оптимізацію комунікаційних протоколів і енергозберігаючі механізми, що критично важливо для подовження часу роботи пристроїв і зниження загальних витрат на експлуатацію систем.

Таким чином, актуальність теми дослідження та розробки проміжного програмного забезпечення розподілених систем зі спеціалізованим функціоналом визначається необхідністю підвищення ефективності, надійності і безпеки сучасних IT-інфраструктур. Впровадження таких рішень відкриває нові можливості для розвитку різноманітних галузей, від промисловості і транспорту до охорони здоров'я та фінансів, забезпечуючи інноваційні підходи до вирішення складних задач у динамічному і швидкозмінному середовищі.

Предметом дослідження є проміжне програмне забезпечення для розподілених систем, що включає спеціалізований функціонал для оптимізації управління ресурсами, комунікацій, безпеки та масштабованості в умовах різних сценаріїв використання, таких як IoT, обробка великих даних та машинне навчання.

Об'єктом дослідження є розподілені системи та їхні компоненти, які використовують проміжне програмне забезпечення для забезпечення взаємодії, управління ресурсами та виконання спеціалізованих функцій в умовах гетерогенної IT-інфраструктури.

Мета полягає у розробкі та впровадженні ефективного проміжного програмного забезпечення для розподілених систем, яке має спеціалізований функціонал для підвищення надійності, масштабованості, безпеки та адаптивності IT-інфраструктури.

У дипломній роботі на тему «Проміжне програмне забезпечення розподілених систем зі спеціалізованим функціоналом» сформульовано такі завдання:

- проаналізувати існуючі рішення та методи проміжного програмного забезпечення для розподілених систем;
- розробити проектування проміжного програмного забезпечення з урахуванням визначених вимог;

– реалізувати прототип проміжного програмного забезпечення зі спеціалізованим функціоналом;
– сформулювати висновки на основі результатів проведеного дослідження. Оформити курсову роботу відповідно до методичних вимог і стандартів оформлення. Скласти список використаних джерел, дотримуючись наукових стандартів цитування та вказівок.

Структура та обсяг проєкту**.** Робота складається зі вступу, трьох розділів, трьох підрозділів, висновків, списку використаних джерел а саме 61.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Розподілені системи – це не лише технічне чудо сучасності, але й спосіб організації обчислень, який має значний вплив на наше щоденне життя. Зазначене визначення підкреслює сутність цих систем – вони є складною мережею автономних обчислювальних елементів, які спільно працюють для досягнення загальної мети [19]. Ця мета може бути різноманітною – від обробки великих обсягів даних до підтримки масштабованих онлайн-сервісів. Що означає "автономність" у контексті розподілених систем? Це означає, що кожен елемент мережі має власний рівень незалежності та можливість приймати рішення відповідно до свого стану та потреб. При цьому вони спільно координуються для досягнення загальної мети. Ця властивість робить розподілені системи гнучкими та стійкими до відмов, оскільки вони можуть компенсувати проблеми на рівні окремих компонентів [26].

Централізовані системи, які були попередньою нормою, мають свої обмеження, особливо коли мова йде про масштабування та надійність. Розподілені системи пропонують альтернативу, де потужність обчислень розподіляється між багатьма вузлами, що дозволяє їм ефективно працювати з великими обсягами даних та високими навантаженнями. Проте, існують виклики, пов'язані з розподіленими системами, такі як забезпечення консистентності даних, управління ресурсами та забезпечення безпеки [1]. Однак розвиток технологій, таких як блокчейн та розподілені бази даних, допомагає вирішувати ці проблеми та робить розподілені системи ще більш привабливими для широкого кола застосувань. Розподілені системи є важливим елементом сучасної інформаційної інфраструктури, який відкриває нові можливості у різних сферах, від бізнесу до науки та повсякденного життя [27].

Таблиця 1.1 - Змістовний аналіз предметної області: проміжне програмне забезпечення розподілених систем зі спеціалізованим функціоналом

Аспект аналізу	Опис
Предметна область	Проміжне програмне забезпечення (middleware) для розподілених систем
Основні функції	Підтримка взаємодії між компонентами системи, забезпечення масштабованості, підвищення надійності
Типи проміжного ПЗ	Віддалений виклик процедур (RPC), обмін повідомленнями, оркестрація сервісів, управління даними
Структурні особливості	Модульність, багатошаровість, підтримка різних протоколів і стандартів
Функціональні особливості	Забезпечення транзакцій, балансування навантаження, безпека даних, управління сесіями
Приклади реалізацій	Apache Kafka (обмін повідомленнями), Spring Cloud (оркестрація сервісів), Apache Ignite (управління даними)
Переваги використання	Підвищення продуктивності, зниження складності розробки, покращення масштабованості
Виклики та обмеження	Складність налаштування та підтримки, можливі затримки в обробці, вимоги до ресурсів
Важливі технології та стандарти	SOAP, REST, gRPC, AMQP, MQTT
Тенденції розвитку	Зростання ролі хмарних обчислень, мікросервісна архітектура, використання контейнерів (Docker, Kubernetes)

Дві важливі характеристики розподілених систем, детально визначають їхню сутність та функціонування. По-перше, розподілена система складається з кількох обчислювальних елементів, кожен з яких може працювати самостійно. Це можуть бути як фізичні пристрої, так і програмні процеси, що працюють на цих пристроях. Автономність кожного вузла дозволяє системі бути більш гнучкою та надійною, оскільки відмова одного вузла не паралізує всю систему, оскільки інші вузли продовжують працювати [35]. По-друге, користувачі (люди чи програми) сприймають розподілену систему як єдине ціле, незважаючи на те, що вузли різні. Це означає, що для досягнення загальної мети необхідно забезпечити взаємодію

між вузлами. Для забезпечення правильної функціональності та ефективності системи в цілому важлива координація роботи вузлів [16].

Розробка механізмів співпраці та взаємодії між автономними вузлами лежить в основі розробки розподілених систем. Це включає в себе розробку протоколів комунікації, алгоритмів реплікації даних, стратегій маршрутизації та інші аспекти, які забезпечують злагоджену роботу всієї системи. Таким чином, розподілені системи забезпечують не лише розділення обчислювальної потужності, але й забезпечують її координацію та спільну дію для досягнення поставлених цілей користувачів [20].

Сучасні розподілені системи представляють собою фантастичну мозаїку різноманітних обчислювальних вузлів, які можуть бути різного типу та розміру. Вони охоплюють від потужних серверів до невеликих пристроїв, які можна легко помістити в кишені. Ця різноманітність дозволяє розподіленим системам пристосовуватися до різних завдань та середовищ. Фундаментальний принцип, на якому ґрунтуються розподілені системи, полягає в тому, що кожен вузол може діяти незалежно від інших. Однак важливо розуміти, що ця автономність вузлів не означає їхню ізоляцію [13]. Навпаки, взаємодія між вузлами є ключовою для успішної роботи розподіленої системи. Ігнорування один одного звільняється від сенсу, оскільки спільна мета досягається саме через координацію та обмін інформацією між вузлами. На практиці, вузли програмуються для досягнення загальних цілей, які вони реалізують через обмін повідомленнями. Коли вузол отримує вхідне повідомлення, він реагує на нього, обробляє отримані дані та вживає відповідних дій. Часто це включає в себе подальшу передачу повідомлень іншим вузлам, що допомагає досягти спільної мети [10].

Факт, що кожен вузол розподіленої системи має своє власне уявлення про час, є ключовим аспектом, який виникає з їхньої незалежності. Уявімо, що ми маємо два вузли, які взаємодіють у розподіленій системі. Кожен з них має свій власний внутрішній лічильник часу, який може по-різному рухатися внаслідок різних обставин, таких як навантаження, швидкість обробки, мережеві затримки

тощо [14]. Ця відсутність загального глобального часу створює складнощі в синхронізації та координації подій в системі. Наприклад, якщо один вузол намагається виконати дію, яка залежить від часу, а інший вузол має відмінне уявлення про поточний час, можуть виникнути ситуації, коли ці дії не відбудуться у потрібний момент часу або будуть виконані з помилковими даними [59]. Один зі способів вирішення цього виклику полягає в розробці алгоритмів синхронізації, які дозволяють вузлам узгоджувати свої локальні часи з іншими вузлами у системі. Такі алгоритми можуть використовувати спеціальні протоколи обміну повідомленнями або механізми корекції часу для забезпечення відносної узгодженості між вузлами [30].

Недоліком цих підходів є те, що вони можуть бути витратними з точки зору обчислювальних та мережевих ресурсів, особливо у великих та складних системах [38]. Тому постійно виникає потреба в пошуку більш ефективних та надійних методів синхронізації в розподілених системах. У підсумку, відсутність загального відліку часу у розподілених системах ставить перед розробниками важливі завдання з синхронізації та координації, які потребують пошуку та розвитку нових технологій та методів [18].

Керування членством і організацією вузлів у розподіленій системі є критичним аспектом, який вимагає уваги розробників. У розподілених системах, що складаються з множини незалежних вузлів, необхідні механізми для реєстрації та управління цими вузлами [60]. Реєстрація вузлів дозволяє системі відстежувати, які вузли є частиною системи, а які ні, що важливо для забезпечення безпеки і контролю доступу до системних ресурсів. Вона також сприяє взаємодії вузлів, забезпечуючи ефективну комунікацію і обмін даними [34].

Крім того, організація вузлів включає створення списків вузлів, з якими кожен учасник системи може спілкуватися безпосередньо. Це особливо важливо в системах з великою кількістю вузлів, де не кожен вузол може безпосередньо взаємодіяти з усіма іншими. Створення таких списків сприяє оптимальній маршрутизації і ефективному обміну даними між вузлами [61].

Управління членством та організацією вузлів також може включати в себе механізми для додавання нових вузлів до системи, видалення вузлів з системи, виявлення та вирішення несправностей вузлів тощо. Ці механізми дозволяють системі підтримувати стабільність та продуктивність навіть у змінних умовах [58].

Управління членством у групі є ключовим аспектом для забезпечення безпеки, ефективності та контролю в розподілених системах. Залежно від конкретних потреб та вимог системи, можуть застосовуватися різні підходи до управління членством, такі як відкриті та закриті групи.

У відкритій групі будь-якому вузлу дозволено приєднатися до розподіленої системи без обмежень. Це означає, що будь-який вузол може надсилати повідомлення будь-якому іншому вузлу в системі. Такий підхід може бути корисним у випадках, коли система розглядається як відкрита для всіх користувачів або коли не потрібне складне управління допуском [57].

Навпаки, у закритій групі тільки члени цієї групи мають можливість спілкуватися один з одним. Це створює додаткові виклики для управління членством, оскільки потрібно мати механізм для дозволу вузлам приєднатися або залишити групу. Такий підхід може бути корисним у випадках, коли потрібно обмежити доступ до ресурсів або інформації лише для обраних користувачів або вузлів.

Управління членством у відкритих і закритих групах вимагає розробки відповідних механізмів для реєстрації, авторизації та аутентифікації вузлів. Ці механізми дозволяють забезпечити безпеку та контроль у розподілених системах, забезпечуючи доступ до ресурсів лише вповноваженим користувачам або вузлам [48].

Контроль прийому учасників у групу є складним завданням, оскільки потрібно ретельно контролювати аутентифікацію та створювати довіру між вузлами. Нижче наведено деякі важливі елементи цього нагляду. По-перше, необхідний механізм аутентифікації вузлів для ефективного контролю прийому. Дозволити вузлам приєднуватися до групи без перевірки їхніх ідентичності — це

недостатньо [56]. Тим не менш, відсутність достатнього механізму аутентифікації може створити проблеми з масштабованістю та підвищити ризик несанкціонованого доступу до системи. По-друге, кожен вузол повинен переконатися, що він спілкується з реальним членом групи, а не з потенційним зловмисником, який намагається зіпсувати систему. Це необхідно для забезпечення безпеки та запобігання атакам зсередини на систему. Використання протоколів і методів криптографії може допомогти зменшити ймовірність несанкціонованого доступу [2].

Нарешті, конфіденційність є ще одним важливим аспектом у контролі прийому. У розподілених системах, де вузли можуть легко спілкуватися з нечленами, важливо забезпечити захист конфіденційної інформації. Використання криптографічних методів шифрування та підпису може допомогти запобігти проникненню та зловживанню конфіденційними даними [28].

Практика показує, що розподілені системи часто реалізуються у вигляді накладних мереж. У цьому підході вузол зазвичай є програмним процесом, що має список інших процесів, з якими він може напряму обмінюватися повідомленнями. В окремих випадках для передачі повідомлень спочатку потрібно звертатися до сусідніх вузлів. Обмін повідомленнями може здійснюватися через TCP/IP або UDP канали, а також через засоби вищого рівня. Існують два основні типи накладних мереж [12]:

- структуроване накладення, кожен вузол має чітко визначений набір сусідів, з якими він може спілкуватися, як, наприклад, у деревоподібних або кільцевих структурах [22];
- неструктуроване накладення, кожен вузол має випадково вибрані посилання на інші вузли.

У будь-якому випадку, накладна мережа повинна бути підключеною, що означає наявність шляху зв’язку між будь-якими двома вузлами для маршрутизації повідомлень. Одним із прикладів накладних мереж є однорангові (P2P) мережі. Важливо розуміти, що організація вузлів є складним завданням і часто вимагає

значних зусиль, будучи однією з найскладніших частин управління розподіленими системами.

Зазначалося, що для досягнення успішної роботи розподіленої системи необхідно, щоб вона виглядала як єдина цілісна система. Деякі дослідники йдуть ще далі, стверджуючи, що має бути забезпечений єдиний системний погляд, коли кінцевий користувач навіть не помічає, що процеси, дані та контроль розподілені по комп'ютерній мережі. Однак досягнення цього ідеального уявлення про єдину систему може виявитися дуже складним завданням [46].

Уявлення про єдину систему часто вимагає великих зусиль та технічних рішень, щоб забезпечити синхронність та взаємодію між різними складовими системи. Однак у більшості випадків це є ідеалом, який може бути складно досягнути в реальних умовах. Тому в нашому визначенні розподіленої системи ми вибрали щось більш реалістичне та слабке, а саме, що вона здається цілісною [55].

Грубо кажучи, розподілена система вважається цілісною, якщо вона веде себе відповідно до очікувань своїх користувачів. Це означає, що сукупність вузлів працює як єдине ціле, незалежно від того, де, коли і як відбувається взаємодія між користувачем і системою. Хоча можуть бути деякі різниці у механізмах та процесах за кулісами, для кінцевого користувача це має значення лише у випадках, коли це впливає на продуктивність та ефективність роботи системи [54].

Запропонувати єдиний узгоджений погляд у розподілених системах – це завдання, що часто виявляється досить складним. В таких системах має значення не тільки здатність виконувати завдання ефективно, але й забезпечувати користувачам відчуття єдності та прозорості [45].

Наприклад, ідеальна розподілена система має бути такою, де кінцевий користувач не може точно визначити, на якому саме комп'ютері виконується його процес. Підходи, такі як реплікація даних для підвищення продуктивності, дозволяють системі працювати більш ефективно, але при цьому мають забезпечити враження єдності та прозорості для кінцевого користувача. Така прозорість розподілу є важливою метою проектування розподілених систем. У певному сенсі,

це подібно до підходу, який застосовується у багатьох Unix-подібних операційних системах. Там доступ до ресурсів, незалежно від того, чи це файли, пристрої зберігання даних або мережі, здійснюється через уніфікований інтерфейс файлової системи. Цей підхід допомагає "приховати" відмінності між різними ресурсами та забезпечує єдність у взаємодії з ними [53].

Такий же підхід важливий і в розподілених системах. Користувач повинен мати можливість взаємодіяти з системою, незалежно від того, де розміщені дані або який саме вузол обробляє їхню обробку. Це вимагає розробки стандартизованих протоколів та інтерфейсів, що дозволяють забезпечити прозорість та єдність у взаємодії з системою для кінцевого користувача.

Прагнення до створення єдиної цілісної системи в розподіленій архітектурі призводить до важливого компромісу, який стосується стійкості та надійності. Одна з ключових особливостей розподілених систем полягає в тому, що вони складаються з мережевих вузлів, які можуть бути розташовані у різних місцях і піддаються різноманітним умовам експлуатації. Внаслідок цього, неминуче, що в будь-який момент часу частина системи може вийти з ладу або перестати працювати належним чином [41].

Цей факт відкриває двері для несподіваної поведінки системи, де деякі частини можуть продовжувати функціонувати, тоді як інші можуть втратити здатність до роботи. Це є реальністю, з якою доводиться зіткнутися в контексті розподілених систем. У зв'язку з тим, що розподілені системи складніші та включають більше складових, вони стають вразливішими до такого роду часткових відмов [52].

Лауреат премії Тьюрінга Леслі Лемпорт гостро сформулював цю проблему, описавши розподілену систему як "таку, в якій поломка комп’ютера, про існування якого ви навіть не знали, може зробити ваш комп’ютер непридатним для використання" [5]. Це свідчить про складність виявлення та управління несправностями у розподілених системах, де взаємозв'язок між вузлами може бути складним та непередбачуваним.

1.2 Аналіз наявного програмно-апаратного забезпечення предметної області

Аналіз існуючих розподілених систем з проміжним програмним забезпеченням із спеціалізованим функціоналом відкриває широкий спектр застосувань та можливостей. Такі системи відіграють ключову роль у різних сферах, включаючи телекомунікації, фінанси, медицину, науку, виробництво, транспорт та багато інших [7].

Один з найпоширеніших видів проміжного програмного забезпечення для розподілених систем - сервіси управління даними. Наприклад, системи управління базами даних (СУБД) такі як Oracle, MySQL, або MongoDB, надають засоби для зберігання, організації та обробки великих обсягів даних в розподілених середовищах [6]. Ці СУБД дозволяють розробникам додатків ефективно використовувати дані в різних масштабах та забезпечують високу доступність та надійність. Ще одним прикладом є мікросервісна архітектура, де проміжне програмне забезпечення забезпечує комунікацію між різними мікросервісами [39]. Наприклад, сервіси, які відповідають за аутентифікацію, авторизацію, обробку платежів тощо, можуть бути реалізовані як окремі мікросервіси, які взаємодіють за допомогою проміжного програмного забезпечення для маршрутизації запитів та обробки даних [31].

У сфері телекомунікацій існують проміжні програмні рішення для керування трафіком, балансування навантаженням та забезпечення безпеки мережі. Ці системи можуть автоматично перерозподіляти трафік між серверами, реалізувати заходи з обробки помилок та захисту від кібератак. У фінансовому секторі проміжне програмне забезпечення може використовуватися для платіжних систем, онлайн-банкінгу, торгівлі на фондових ринках та аналізу ризиків [24]. Воно допомагає забезпечити безпеку та надійність операцій, здійснюваних у реальному часі, та забезпечити швидку обробку великого обсягу фінансових транзакцій.У медичній галузі проміжне програмне забезпечення може бути використане для керування медичними записами, обробки медичних зображень, телемедицини та

інших медичних додатків. Воно дозволяє забезпечити конфіденційність та безпеку медичних даних, координацію роботи між різними медичними закладами та забезпечити доступ до потрібної інформації у відповідний час [8].

Ще одним прикладом є використання проміжного програмного забезпечення в наукових дослідженнях та обчислювальних задачах. Розподілені системи можуть використовувати проміжне програмне забезпечення для керування розподіленими обчислювальними ресурсами, координації роботи різних обчислювальних вузлів та обробки великих обсягів даних [29]. Нарешті, у виробничому секторі проміжне програмне забезпечення може використовуватися для автоматизації виробничих процесів, моніторингу та контролю за обладнанням, прогнозування виробничих потреб та оптимізації виробничих ланцюгів [23].

Усі ці приклади свідчать про важливість проміжного програмного забезпечення у розподілених системах зі спеціалізованим функціоналом. Воно відіграє ключову роль у забезпеченні ефективності, надійності та масштабованості таких систем, дозволяючи розробникам фокусуватися на реалізації бізнес-логіки своїх додатків, а не на деталях мережевої комунікації та взаємодії з розподіленими ресурсами [47].

Для полегшення розробки розподілених додатків, системи часто організовуються таким чином, щоб мати окремий рівень програмного забезпечення, яке логічно розміщується поверх відповідних операційних систем комп’ютерів, що є частиною системи [40]. Ця організація, зображена на рисунку 1.1, призводить до створення так званого проміжного програмного забезпечення. Проміжне програмне забезпечення відіграє ключову роль у розподіленій системі, допомагаючи взаємодії між різними компонентами системи та забезпечуючи спільний доступ до ресурсів. Воно дозволяє забезпечити зручний та ефективний обмін даними та повідомленнями між вузлами системи, незалежно від їхнього фізичного розташування та операційної системи [43]. Одним з ключових завдань проміжного програмного забезпечення є забезпечення надійності та безпеки взаємодії між компонентами системи. Воно може включати в себе різноманітні

сервіси та протоколи для забезпечення аутентифікації, авторизації та шифрування даних, а також механізми обробки помилок та відновлення системи після відмов. Завдяки проміжному програмному забезпеченню, розробники можуть зосередитися на функціональних аспектах своїх додатків, маючи можливість використовувати стандартизовані та ефективні засоби взаємодії з іншими компонентами системи. Це сприяє покращенню якості та швидкості розробки, а також сприяє створенню більш надійних та масштабованих розподілених додатків.

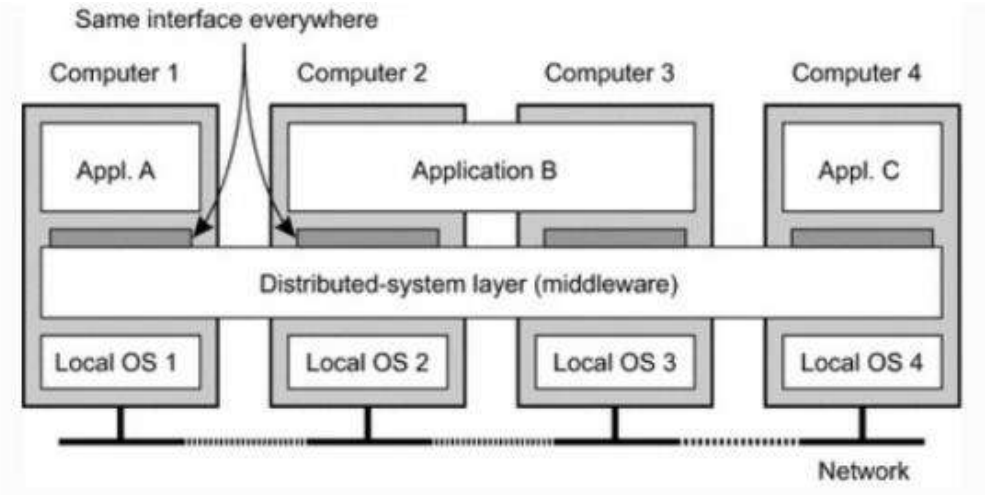


Рисунок 1.1 – організація рівня програмного забезпечення [1]

На рисунку 1.1 намалювано чотири комп'ютери, які з'єднані в мережу, та три програми, серед яких додаток B розподілений між комп'ютерами 2 і 3. Кожній програмі надається однаковий інтерфейс. Ця організація структури дозволяє забезпечити взаємодію між компонентами однієї розподіленої програми та надає можливість комунікації між різними програмами. Розподілена система, яка представлена на рисунку, забезпечує засоби комунікації між компонентами однієї розподіленої програми, дозволяючи їм обмінюватися даними та повідомленнями через мережу. Крім того, вона також надає можливість різним програмам взаємодіяти між собою, що дозволяє створювати складні та функціональні системи з різними програмними модулями [36].

Однак, незважаючи на різноманітність комп'ютерів та операційних систем, що використовуються в системі, розподілена система намагається приховати ці відмінності якнайкраще і розумно [51]. Це означає, що для кінцевого користувача важливо, щоб він не мав проблем з використанням програми через технічні характеристики комп'ютера або операційної системи. Таким чином, розподілена система намагається забезпечити єдність і стабільність у взаємодії з користувачем, незалежно від технічних особливостей обладнання чи програмного забезпечення, яке використовується [44].

У певному сенсі, проміжне програмне забезпечення для розподіленої системи виконує роль, схожу на операційну систему для окремого комп'ютера: воно управляє ресурсами, забезпечуючи ефективний обмін і розподіл цих ресурсів у мережі для додатків. Окрім управління ресурсами, таке програмне забезпечення надає послуги, схожі на ті, що є в більшості операційних систем, включаючи [25]:

- засоби для міжпрограмної комунікації;
- служби безпеки;
- бухгалтерські сервіси;
- маскування та відновлення після збоїв.

Основна відмінність послуг проміжного програмного забезпечення від їх еквівалентів у операційній системі полягає в тому, що ці послуги пропонуються в мережевому середовищі. Це означає, що проміжне програмне забезпечення забезпечує функціональність, яка дозволяє програмам на різних комп’ютерах взаємодіяти та працювати разом, як частина єдиної розподіленої системи. Крім того, більшість служб проміжного програмного забезпечення корисні для багатьох програм [49]. Це робить проміжне програмне забезпечення схожим на контейнер із часто використовуваними компонентами та функціями, які тепер більше не потрібно реалізовувати окремо в кожному додатку. Це значно спрощує процес розробки, дозволяючи зосередитися на специфічній логіці додатків, замість того, щоб займатися реалізацією базових мережевих функцій [33].

Кілька прикладів типових служб проміжного програмного забезпечення:

- служби віддаленого виклику процедур (RPC);
- служби управління даними;
- служби безпеки;
- служби повідомлень та черги повідомлень;
- служби управління іменами.

Ці приклади ілюструють, як проміжне програмне забезпечення допомагає стандартизувати та спростити розробку розподілених додатків, надаючи готові рішення для поширених завдань і забезпечуючи єдиний інтерфейс для взаємодії з різними компонентами системи [21].

1.3 Визначення вимог до системи автоматизації та розробка технічного завдання

Вимоги до проміжного програмного забезпечення для розподілених систем зі спеціалізованим функціоналом можуть бути визначені з урахуванням конкретних потреб та характеристик таких систем [9].

Основні функціональні вимоги можна розглядати у контексті їхнього впливу на ефективність, надійність, масштабованість та безпеку системи.

Нижче наведено кілька ключових вимог, які можуть бути важливими при проектуванні проміжного програмного забезпечення для таких систем.

Таблиця 1.2 - Вимоги до автоматизації та ТЗ для проміжного ПЗ розподілених систем

Аспект вимог	Опис
Функціональні вимоги	Визначення потрібного функціоналу для проміжного ПЗ, такого як обмін повідомленнями, оркестрація сервісів, управління даними

Продовження таблиці 1.2 – Вимоги до автоматизації та ТЗ для проміжного ПЗ розподілених систем

Нефункціональні вимоги	Встановлення вимог до продуктивності, масштабованості, надійності та безпеки проміжного ПЗ
Інтеграційні вимоги	Опис інтерфейсів для взаємодії з іншими компонентами системи, стандарти обміну даними
Вимоги до підтримки і супроводу	Мінімальні вимоги до технічної підтримки, документації, навчання персоналу
Вимоги до безпеки	Захист від несанкціонованого доступу, шифрування комунікацій, контроль доступу до ресурсів
Вимоги до масштабованості	Здатність проміжного ПЗ ефективно масштабуватися з ростом обсягу даних та навантаження
Вимоги до продуктивності	Мінімальні показники продуктивності, швидкість обробки запитів та передачі даних
Вимоги до сумісності	Забезпечення сумісності з різними операційними системами, мовами програмування, протоколами
Вимоги до документування	Потрібність створення технічної документації, API-документації, інструкцій користувача

Ці вимоги можуть бути доповнені або модифіковані відповідно до конкретних потреб та обмежень розподіленої системи зі спеціалізованим функціоналом. Розробники мають враховувати ці вимоги під час проектування та реалізації проміжного програмного забезпечення, щоб забезпечити ефективну та надійну роботу системи [37].

Процес проектування інформаційної системи тісним образом пов’язаний з її архітектурним описом, що відбито в деяких визначеннях терміну "архітектура".

Можна виділити п’ять різних підходів до проектування:

– метод, який формує процес керування вимогами;
– метод, який базується на процедурі розробки документації;
– метод, заснований на системі керування якістю;
– підхід до архітектури.

Архітектурний підхід до проектування інформаційних систем дійсно можна вважати одним із найбільш зрілих та ефективних. Його ключовим аспектом є створення фреймворка або каркаса, який може бути легко адаптований під потреби конкретної системи [50]. Цей підхід передбачає розбиття завдання проектування на дві основні підзадачі: розробка багаторазово використовуваного каркаса та створення самої системи на його основі. Важливою перевагою використання фреймворків є можливість швидкої адаптації та змінення функціональності системи через ітеративність процесу проектування [42]. Це означає, що розвиток системи може відбуватися поетапно, дозволяючи здійснювати корекції та внесення змін на кожному етапі відповідно до поточних потреб та вимог користувачів. У процесі використання архітектурного підходу різні групи фахівців можуть працювати над вирішенням різних підзадач [15]. Наприклад, одна група може бути відповідальна за розробку та підтримку каркаса, тоді як інша група займається створенням конкретної системи на його основі. Важливим аспектом архітектурного підходу є його спрямованість на ліквідацію недоліків, що можуть виникати в процесі проектування, особливо при використанні методологій керування вимогами. Цей підхід дозволяє підвищити якість та ефективність процесу розробки інформаційних систем, зменшити ризики та забезпечити більшу гнучкість у відповіді на зміни у вимогах та умовах проекту [32].

2 ПРОЄКТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

Після того, як були проведені аналіз предметної області та визначені вимоги до програмно-технічного засобу, наступним кроком є вивчення можливих способів вирішення поставлених задач. Цей етап проектування визначає конкретні технічні рішення та архітектурні підходи, які будуть використовуватися для створення програмного продукту.

Під час розробки проєкту обґрунтовуються проєктні рішення, які дають змогу реалізувати вимоги технічного завдання, забезпечити сумісність та взаємодію різних компонентів програмно-технічного засобу. У процесі вибору рішень розглядаються різні підходи до реалізації функціональних і нефункціональних вимог, що дозволяє знайти оптимальний варіант з точки зору продуктивності, надійності та економічної доцільності.

На цьому етапі важливо детально проаналізувати можливі технічні рішення. Наприклад, можуть розглядатися різні архітектурні стилі, такі як монолітна архітектура, мікросервіси, або багатоланкова архітектура. Кожен з цих підходів має свої переваги та недоліки, і вибір конкретного залежить від специфіки проекту, вимог до масштабованості, продуктивності та підтримки.

Моделі реальної системи, отримані на етапі аналізу, у цьому розділі розширюються і коригуються таким чином, щоб вони могли бути реалізовані апаратно або програмно. Це включає деталізацію компонентів системи, визначення їх функцій та способів взаємодії. Важливо забезпечити, щоб всі компоненти були добре документовані та зрозумілі для команди розробників.

Одним з ключових аспектів цього етапу є забезпечення сумісності та взаємодії різних компонентів системи. Це досягається шляхом визначення чітких інтерфейсів та протоколів взаємодії, що дозволяє знизити ризики несумісності та забезпечити коректну роботу всієї системи. Також важливо враховувати можливість майбутніх змін та оновлень, щоб система була гнучкою та легко масштабованою.

Крім того, необхідно приділити увагу питанням безпеки та захисту даних. Це включає розробку механізмів аутентифікації та авторизації, забезпечення захищеної передачі даних та реалізацію заходів для запобігання можливим загрозам. Безпека повинна бути інтегрована в архітектуру системи з самого початку, а не додаватися пізніше як додатковий шар.

У кінцевому підсумку, вибір та обґрунтування технічних рішень є критичним етапом проектування, від якого залежить успішність всієї системи. Правильно обрані технічні рішення дозволяють забезпечити високу якість, продуктивність та надійність програмно-технічного засобу, що відповідає вимогам замовника та очікуванням користувачів.Проєктування програмно-технічного засобу у загальному випадку виконується на таких стадіях: ескізний проєкт (архітектурний дизайн); технічний проеєт (детальне проектування).

Основні завдання розробки структури програмно-технічного засобу:

- виділення апаратних або програмних підсистем і відображення на них зовнішніх функцій програмно-технічного засобу;
- визначення способів взаємодії між підсистемами.

Залежно від типу структури, програмно-технічного засобу може розбиватися на рівні, що у подальшому вимагатиме додаткових описів при декомпозиції та проектуванні.

При розробці структури системи автоматизації потрібно:

- застосування елементів інтерфейсу тощо.
- описати функціональне призначення основних модулів та інформаційних ресурсів, їх взаємозв’язок, а також обмін даними.

Розробка структури програмно-технічного засобу може бути заснована на деякому шаблоні проектування.

У проекті людино-машинного інтерфейсу визначають:

- визначення способів взаємодії між підсистемами.
- зв’язок елементів інтерфейсу один з одним;
- застосування елементів інтерфейсу тощо.

Опис людино-машинного інтерфейсу користувача доцільно супроводжувати схемами екранних форм, систем меню, діалогів тощо.

Також у цьому розділі здійснюється аналіз методів комп'ютерної інженерії для реалізації програмно-технічного засобу із зазначенням переваг та недоліків в контексті їх використання для реалізації проєкту. У результаті аналізу визначається оптимальний варіант.

Спроектувати архітектуру системи, враховуючи потреби розподіленості та спеціалізованого функціоналу.

Визначити принципи взаємодії між компонентами системи.

Розробити схему комунікації між вузлами розподіленої системи.

Корпоративна архітектура представляє собою ключовий аспект управління організаційними системами, де кожен з типів архітектур відіграє свою важливу роль у забезпеченні ефективного функціонування компанії. Розглянемо кожен з них детальніше:

1. Бізнес архітектура (Business architecture), данна архітектура визначає стратегічні цілі організації, структуру бізнес-процесів, взаємозв'язки між ними, а також взаємодію з зовнішнім середовищем, включаючи клієнтів, партнерів та конкурентів.
2. IT-архітектура (Information Technology architecture), цей тип архітектури описує всі компоненти технологій інформаційної системи, такі як апаратне забезпечення, програмне забезпечення, мережі та інфраструктура, які використовуються для підтримки бізнес-процесів.
3. Архітектура даних (Data architecture), ця архітектура визначає структуру та організацію даних в організації, включаючи методи їх зберігання, обробки, доступу та захисту. Вона забезпечує основу для ефективного управління даними та їх використання в бізнес-процесах.
4. Програмна архітектура (Software architecture), ця архітектура описує структуру та організацію програмного забезпечення, включаючи компоненти, модулі, інтерфейси та взаємодію між ними. Вона визначає

загальний дизайн програмних продуктів та гарантує їхню взаємодію та сумісність.

5. Технічна архітектура (Hardware architecture), ця архітектура визначає конфігурацію та організацію апаратного забезпечення, таке як сервери, мережеве обладнання, сховища даних та інші пристрої, необхідні для підтримки IT-інфраструктури.

Використання моделі корпоративної архітектури дозволяє організаціям створити систематизований підхід до управління своїми бізнес-процесами та інформаційними ресурсами, що в результаті сприяє підвищенню ефективності та конкурентоспроможності компанії.

Корпоративна архітектура забезпечує структуру та методологію для аналізу, проектування, впровадження та управління інформаційними системами в організації. Вона інтегрує різні аспекти діяльності компанії, такі як стратегічні цілі, бізнес-процеси, інформаційні потоки, інформаційні технології та інфраструктуру. Це дозволяє забезпечити узгодженість між бізнес-цілями та інформаційними системами, що підтримують ці цілі.

Однією з головних переваг використання корпоративної архітектури є можливість створення цілісного погляду на всі аспекти діяльності організації. Це дозволяє керівництву краще розуміти взаємозв'язки між різними підрозділами та процесами, що сприяє прийняттю більш обґрунтованих рішень. Корпоративна архітектура також допомагає виявити дублювання функцій та процесів, що дозволяє оптимізувати ресурси та знизити витрати.

Крім того, корпоративна архітектура сприяє покращенню управління змінами в організації. Вона забезпечує структуру для планування та впровадження змін, що дозволяє мінімізувати ризики та знижувати вплив змін на діяльність компанії. Це особливо важливо в умовах динамічного бізнес-середовища, де швидкі та ефективні зміни можуть стати ключовим фактором успіху.

Корпоративна архітектура також сприяє підвищенню якості даних та інформації в організації. Вона забезпечує стандартизацію даних та процесів, що дозволяє забезпечити їхню точність, актуальність та доступність. Це, в свою чергу, покращує якість прийняття рішень та дозволяє більш ефективно використовувати інформаційні ресурси.

Отже, використання моделі корпоративної архітектури дозволяє організаціям систематизувати управління своїми бізнес-процесами та інформаційними ресурсами. Це сприяє підвищенню ефективності та конкурентоспроможності компанії, забезпечуючи узгодженість між стратегічними цілями та інформаційними системами, покращуючи управління змінами та підвищуючи якість даних.

Рисунок 2.1 – Модель корпоративної інформаційної системи

Технічна архітектура дійсно є важливим аспектом будь-якої інформаційної системи. Вона визначає фізичну структуру системи, включаючи апаратне забезпечення та мережеві компоненти, необхідні для забезпечення функціональності системи. Перш за все, до складу технічної архітектури входять периферійні пристрої. Це всі зовнішні пристрої, які взаємодіють з системою, такі як клавіатура, миша, монітори, принтери, сканери та інші. Вони забезпечують

спосіб введення даних до системи та виведення результатів її роботи. Другим важливим компонентом є обчислювальне обладнання, таке як процесори, оперативна пам'ять та жорсткі диски. Процесори відповідають за обробку даних та виконання програмного коду, оперативна пам'ять - за тимчасове зберігання даних та інструкцій, а жорсткі диски - за постійне зберігання інформації.

Окремо слід відзначити мережеві компоненти, такі як мережні комутатори, маршрутизатори та кабельна інфраструктура. Вони забезпечують зв'язок між різними частинами системи та дозволяють обмінюватися даними між ними. Надійність цих компонентів є критичною для стабільної роботи системи.

Крім того, джерела безперебійного живлення також важливі для забезпечення неперервності роботи системи, особливо в умовах можливих відключень електроенергії або інших проблем з електропостачанням.

Програмна архітектура є важливою складовою будь-якої інформаційної системи, оскільки вона визначає структуру та організацію комп'ютерних програм, які вирішують конкретні завдання. Цей тип архітектури орієнтований на опис додатків, які становлять основну функціональну частину інформаційної системи.

На рівні програмної архітектури описуються програмні інтерфейси, які визначають способи взаємодії між різними компонентами програмного забезпечення. Це можуть бути інтерфейси API, які визначають доступ до функцій або сервісів, а також специфікації взаємодії між різними модулями програми.

Крім того, програмна архітектура включає в себе компонентну структуру програми, тобто розподіл програмного забезпечення на окремі модулі або компоненти, які відповідають за певні функції або області відповідальності. Ці компоненти можуть взаємодіяти між собою через визначені інтерфейси та виконувати певні завдання.

Щодо архітектури даних, вона поєднує в собі як фізичні сховища даних, так і засоби керування цими даними. Вона також включає в себе логічні сховища даних, де визначаються логічні та фізичні моделі даних, правила цілісності та обмеження для даних. При орієнтації компанії на роботу зі знаннями, може бути

виділений окремий рівень – архітектура знань (Knowledge architecture), де визначаються способи зберігання, організації та розподілу знань в системі.

IT-архітектура відіграє критичну роль у сучасних інформаційних системах, оскільки вона є сполучним елементом між програмною архітектурою та архітектурою даних. Цей рівень формує базовий набір сервісів, які використовуються на рівнях програмної архітектури та архітектури даних. Брак особливих функцій на цьому рівні може призвести до збоїв у роботі інформаційної системи та втрат для бізнесу.

У деяких випадках, особливо при високому рівні інтеграції додатків, IT-архітектура може бути нерозрізненою від архітектури окремого додатка. Такий підхід допомагає забезпечити глибоку взаємодію та спільну роботу різних компонентів системи.

Прикладом IT-архітектури є продукт SharePoint від компанії Microsoft. SharePoint надає сервіси для спільної роботи та зберігання інформації, що є критичним для функціонування багатьох компаній. Базові системні модулі SharePoint відносяться до IT-архітектури, тоді як користувальницькі інтерфейси та функціональність становлять програмну архітектуру.

Основна функція IT-архітектури - забезпечити функціонування важливих бізнес-додатків для досягнення стратегічних бізнес-цілей. Якщо певна функція потрібна у декількох додатках, вона може бути перенесена на рівень IT-архітектури, що сприяє підвищенню інтеграції системи та зниженню складності архітектури додатків.

Рівень бізнес-архітектури або архітектури бізнес-процесів є останнім в ієрархії архітектурної організації інформаційних систем. На цьому рівні визначаються стратегії ведення бізнесу, способи керування, принципи організації та ключові бізнес-процеси, які відображають величезну важливість для діяльності підприємства.

З урахуванням принципу декомпозиції, проектування інформаційних систем зазвичай виконується з поділом функціонального призначення їхніх компонентів.

Це означає створення багаторівневої архітектури, де функції та процеси розподіляються на різні рівні залежно від їхньої природи та значущості для бізнесу.

На рівні бізнес-архітектури вирішуються стратегічні питання, такі як визначення місії та цілей підприємства, а також розробка стратегій для досягнення цих цілей. Тут також визначається, які бізнес-процеси потрібно оптимізувати чи автоматизувати за допомогою інформаційних систем.

Цей рівень архітектури визначає основні принципи та стратегії, які лягають в основу подальшого проектування і реалізації інформаційних систем на рівні програмної, технічної та даних архітектур. Розробка бізнес-архітектури дозволяє підприємству краще розуміти свої потреби і забезпечує стратегічну основу для всіх подальших дій з розробки інформаційних систем.

Можна розподілити функціонал на три основні групи, спрямовані на вирішення різних завдань:

– взаємодія з користувачами;
– бізнес-логіка;
– управління ресурсами.

Для реалізації цих функцій створюється відповідна програмна система, яка має багаторівневу структуру компонентів.

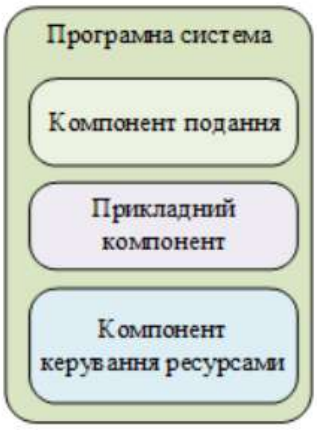


Рисунок 2.2 – Компоненти програмної системи

Компонент подання в інформаційних системах відіграє ключову роль у забезпеченні ефективної взаємодії користувачів з програмою. Його основні завдання включають обробку натискання клавіш, руху різних контролерів, а також висновок інформації на екран – користувацький інтерфейс, який дозволяє взаємодіяти з програмою. Наприклад, веб-сайти, мобільні додатки та десктопні програми мають свої власні компоненти подання, які відповідають за відображення інформації та сприйняття користувацьких дій.

Прикладний компонент, з іншого боку, є набором правил і алгоритмів, які реалізовані для виконання певних функцій системи. Ці компоненти відповідають за обробку даних, реакцію на дії користувачів або внутрішні події, а також за забезпечення логіки функціонування програми в цілому. Наприклад, у веб-додатку прикладні компоненти відповідають за обробку запитів користувачів, взаємодію з базою даних, валідацію даних та генерацію відповідей для користувачів.

Ці два типи компонентів часто співпрацюють між собою, створюючи функціонально повноцінні програмні рішення, які задовольняють потреби користувачів та бізнес-вимоги. Інтеграція компонентів подання та прикладних компонентів дозволяє створювати високоефективні та зручні для використання інформаційні системи.

Компонент керування ресурсами в інформаційних системах забезпечує основні операції з даними, такі як їх зберігання, модифікацію, вибірку та видалення. Цей компонент відповідає за управління даними, що пов’язані з розв’язуваним прикладним завданням, та забезпечує доступ до цих даних з інших компонентів системи. Він включає в себе бази даних, файлові системи, кеші, а також механізми забезпечення консистентності, цілісності та безпеки даних.

Одним з ключових етапів проектування архітектури інформаційної системи є розподіл функціональних компонентів по обраній платформній архітектурі. Це важлива стратегічна вирішальна точка, яка визначає, де будуть розміщені різні частини системи та як вони будуть взаємодіяти між собою. Вибір платформної архітектури може включати в себе використання монолітної архітектури,

мікросервісної архітектури, серверних архітектур або хмарних сервісів, залежно від потреб і вимог проекту.

Наприклад, у монолітній архітектурі компонент керування ресурсами може бути реалізований як частина єдиного застосунку, який включає в себе інші функціональні компоненти. У мікросервісній архітектурі цей компонент може бути розділений на окремі мікрослужби, які незалежно масштабуються та взаємодіють між собою через мережу. Важливо обирати архітектурний підхід, який найкращим чином відповідає вимогам проекту та забезпечує його майбутнє розвиток.

Існують різновиди розподілених архітектур, включаючи:

- архітектуру, де використовується файловий сервер;
- архітектуру, засновану на взаємодії клієнта і сервера;
- архітектуру, спеціалізовану для веб-додатків.

Файл-серверна архітектура є однією з типових моделей розподілених систем, де основний акцент робиться на централізованому зберіганні даних. У цій архітектурі ресурс для зберігання даних, що включає файли та інші ресурси, зазвичай знаходиться на окремому сервері, відомому як "файловий сервер". Основною ідеєю цієї архітектури є розділення функціональності між клієнтськими і серверними компонентами системи. На клієнтських комп'ютерах знаходяться всі програми та інтерфейси для роботи з даними, в той час як самі дані фактично зберігаються на файловому сервері. Такий підхід дозволяє забезпечити централізований доступ до даних для всіх користувачів мережі.

Однією з переваг такої архітектури є спрощення адміністрування та управління даними, оскільки всі файли знаходяться на одному сервері. Це також дозволяє ефективно використовувати ресурси мережі, оскільки можна централізовано керувати доступом до даних та виконанням операцій з ними. Централізоване зберігання даних значно спрощує резервне копіювання і відновлення інформації, оскільки всі важливі файли знаходяться в одному місці.

Проте, існують деякі обмеження такої архітектури. Наприклад, централізоване зберігання може призвести до вузького місця в мережі, особливо в

разі великого обсягу даних або великого числа одночасних користувачів. У такій ситуації продуктивність мережі може знижуватися через високі вимоги до пропускної здатності. Крім того, можуть виникати проблеми з доступністю даних у випадку відмови файлового сервера. Відмова сервера може призвести до втрати доступу до важливих даних для всіх користувачів, що підключені до нього.

Для мінімізації ризиків, пов'язаних із вузьким місцем і відмовами сервера, можна використовувати різні підходи, такі як балансування навантаження, використання кластерів серверів або налаштування систем резервування. Це дозволяє підвищити надійність та продуктивність системи, забезпечуючи безперервний доступ до даних навіть у випадку технічних збоїв.

Ця системна організація має наступні переваги:

- можливість роботи в багатокористувацькому режимі з даними, які зберігаються на сервері;
- централізоване управління правами доступу до загальних даних.
- низькі витрати на розробку;
- швидка розробка;

Недоліки архітектури файлового сервера включають:

- послідовний доступ до загальних даних і відсутність гарантії їхньої цілісності;
- залежність продуктивності від мережі, клієнта та сервера.

Класична реалізація файлової серверної архітектури зображена на рисунку 2.3.

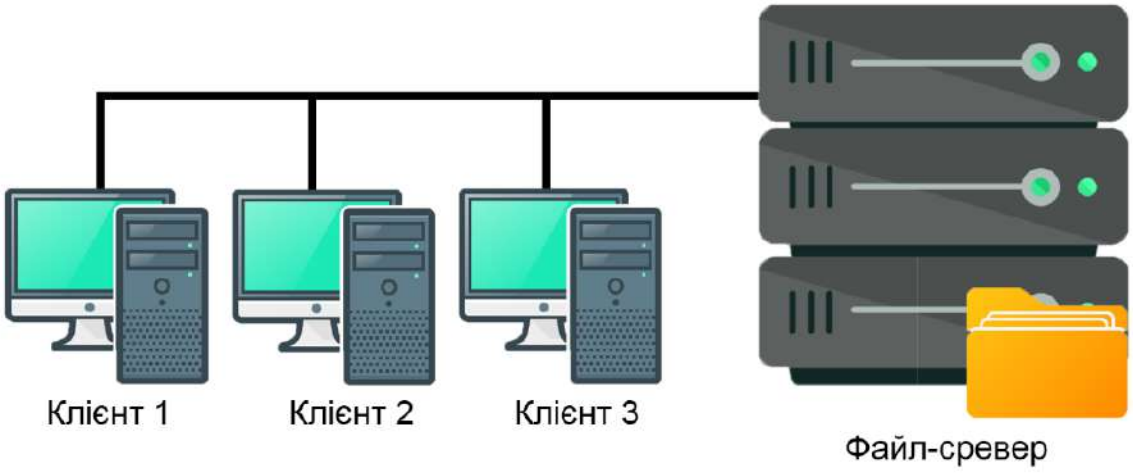


Рисунок 2.3 – Подання «файл-серверної» архітектури

Архітектура «клієнт-сервер» є одним із найпоширеніших підходів до організації мережевих систем. У цій архітектурі сервери виступають у ролі постачальників різноманітних сервісів чи послуг, тоді як клієнтські комп'ютери використовують ці сервіси у своїй роботі. Класична модель клієнт-серверної архітектури передбачає, що у мережі присутній один або кілька серверів, які обслуговують декілька підключених до них клієнтів. В такій моделі сервери зазвичай відіграють роль постачальників послуг, таких як доступ до бази даних, зберігання та обробка інформації, обчислення та інші.

У дворівневій архітектурі, як показано на рисунку 2.4, взаємодія відбувається між клієнтами і серверами. Клієнти зазвичай звертаються до серверів за певними послугами або ресурсами, а сервери надають ці послуги або ресурси відповідно до запитів клієнтів. Ця модель може бути застосована в різних областях, включаючи веб-додатки, бази даних, спільний доступ до файлів та інші.

Перевагою клієнт-серверної архітектури є можливість централізованого управління та обслуговування, а також можливість масштабування системи шляхом додавання нових серверів або клієнтів. Централізоване управління дозволяє легко підтримувати та оновлювати систему, а також забезпечувати

високий рівень безпеки, оскільки більшість критичних даних зберігається на сервері. Масштабування досягається шляхом додавання нових серверів або збільшення обчислювальних ресурсів існуючих серверів, що дозволяє обробляти більшу кількість запитів та підтримувати більше користувачів.

Однак, клієнт-серверна модель має певні обмеження. Одним з них є потенційне перевантаження серверів, що може призвести до зниження продуктивності системи. В разі відмови сервера всі клієнти, що підключені до нього, можуть втратити доступ до необхідних послуг, що впливає на надійність системи. Для вирішення цих проблем можуть бути використані методи балансування навантаження та резервування серверів, що дозволяють зменшити ризики перевантаження та підвищити надійність системи.

Загалом, клієнт-серверна архітектура є ефективним та широко використовуваним підходом для розробки мережевих додатків та систем. Вона забезпечує гнучкість у розподілі ресурсів, централізоване управління та можливість масштабування, що робить її придатною для використання у різних галузях, від корпоративних мереж до інтернет-сервісів.

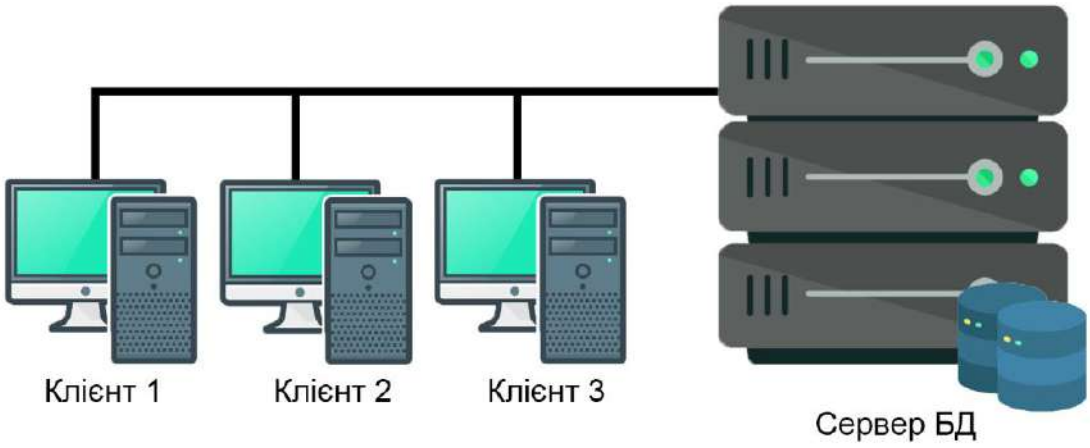


Рисунок 2.4 – Подання «клієнт-сервер» архітектури

Переваги цієї архітектури включають:

– підтримка роботи для багатьох користувачів;
– забезпечення цілісності даних;
– механізми для управління правами доступу до ресурсів сервера;
– можливість розподілу функцій між вузлами мережі;

Серед недоліків можна відзначити:

– ризик недоступності всієї системи у разі виходу з ладу сервера;
– потребу у висококваліфікованому технічному персоналі;
– велику вартість устаткування.

При розвитку системи й збільшенні обсягів роботи, деякі проблеми, які раніше не були так актуальними, стають більш вагомими. Однією з таких проблем є необхідність синхронізації версій програмних компонентів для великої кількості користувачів. Це може стати особливо проблематичним, коли використовуються різні версії програм, що може призвести до несумісності, помилок або збоїв. Одним із способів розв'язання цієї проблеми є використання багатоланкових архітектур, які включають три або більше рівнів функціональності. У таких архітектурах частину загальних додатків переносять на спеціально виділений сервер, що дозволяє знизити вимоги до продуктивності клієнтських машин.

Важливою концепцією в багатоланковій архітектурі є розподіл функціональності між сервером і клієнтськими машинами. Клієнти з низькою обчислювальною потужністю, які виконують обмежені функції, називають «тонкими клієнтами». Тонкі клієнти використовують ресурси сервера для виконання більшої частини обчислювальної роботи, що дозволяє їм працювати на менш продуктивному обладнанні, знижуючи таким чином загальні витрати на інфраструктуру.

Тоді як клієнти з високою продуктивністю, які можуть виконувати складні завдання, відомі як «товсті клієнти». Товсті клієнти мають власні обчислювальні ресурси для виконання більшості завдань, що забезпечує високу швидкість роботи

та незалежність від центрального сервера. Вони можуть працювати автономно, що робить їх більш надійними в умовах нестабільного мережевого з'єднання.

При цьому багатоланкові архітектури забезпечують гнучкість у налаштуванні системи та її масштабуванні. Розподіл функцій між різними рівнями дозволяє оптимізувати використання ресурсів та підвищити загальну продуктивність системи. Наприклад, сервери можуть зосередитися на виконанні важких обчислень і обробці даних, тоді як клієнти забезпечують користувацький інтерфейс і взаємодію з користувачем.

Важливою перевагою багатоланкових архітектур є можливість централізованого оновлення програмного забезпечення. Адміністратори можуть оновлювати серверні компоненти без необхідності одночасного оновлення всіх клієнтських машин. Це значно спрощує процес підтримки та забезпечує збереження сумісності між різними компонентами системи.

Отже, використання багатоланкових архітектур є ефективним підходом для забезпечення стабільної роботи великих інформаційних систем, що потребують високої продуктивності та надійності.

Вони дозволяють знизити вимоги до клієнтських машин, централізувати управління програмним забезпеченням і забезпечити гнучкість у масштабуванні системи відповідно до зростаючих потреб бізнесу.

Окрім того, багатоланкова архітектура дає можливість використовувати портативні пристрої, такі як смартфони чи планшети, як клієнтські пристрої для взаємодії з системою.

На рисунку 2.5 показано приклад багатоланкової архітектури, де функціональність розділена між сервером та різними клієнтськими машинами, що сприяє ефективній роботі системи в умовах збільшення масштабу.

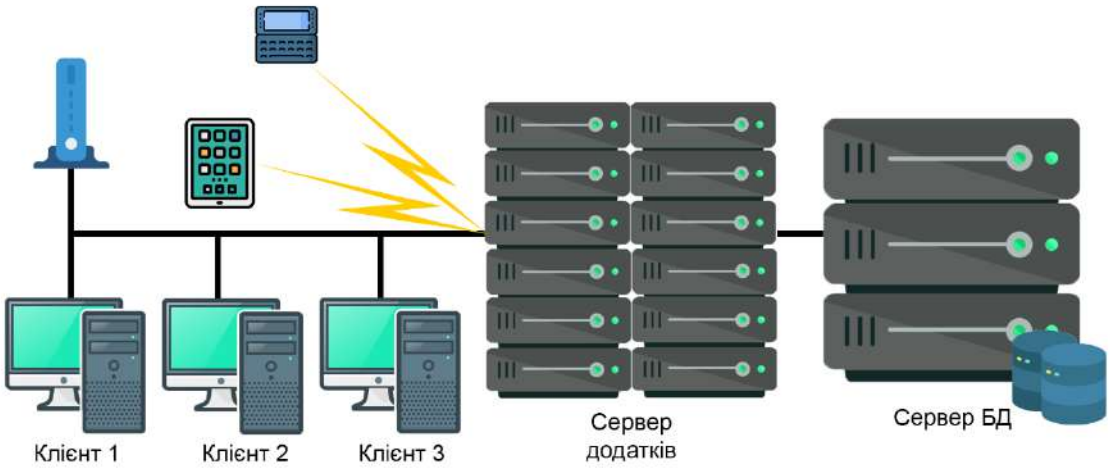


Рисунок 2.5 – Приклад багатоланкової «клієнт-серверної» архітектури

Коли додаток вимагає значних ресурсів для ефективної роботи, використання багатоланкової архітектури стає обґрунтованим. У таких випадках розумно винести додаток на окремий сервер, щоб знизити навантаження на робочі станції користувачів і забезпечити більш ефективне використання ресурсів.

Важливим аспектом успішного реалізації багатоланкової архітектури є грамотний підбір характеристик сервера додатків, сервера баз даних та робочих станцій користувачів. Правильно підібрані характеристики дозволяють створити інформаційну систему з прийнятною вартістю володіння, забезпечуючи оптимальний рівень продуктивності та ефективності використання ресурсів.

При розподілі функціональних компонентів системи за клієнт-серверною архітектурою можуть використовуватися різні підходи. На рисунку 2.6 зображено декілька можливих варіантів розподілу компонентів, які демонструють різноманітність цього підходу та його гнучкість у виборі оптимального розподілу функціональності.

Рисунок 2.6 – Приклади розподілу функціональних компонентів

Розділ проектування є надзвичайно важливим етапом у процесі розробки будь-якого інформаційного продукту. На цьому етапі здійснюється всебічне дослідження можливих шляхів вирішення поставлених завдань, визначення технічних аспектів реалізації програми та вибір оптимального підходу до створення програмного забезпечення.

Проектування починається з аналізу предметної області. Це включає детальне вивчення контексту застосування майбутнього продукту. Етап охоплює дослідження існуючих процесів, визначення основних потреб користувачів та виявлення ключових проблем, які потребують вирішення. Цей аналіз забезпечує глибоке розуміння проблем, які має вирішити продукт, а також контексту, в якому він буде використовуватися.

Процес проектування починається з аналізу предметної області, що включає детальне вивчення контексту застосування майбутнього продукту. На цьому етапі розробники вивчають існуючі процеси, визначають основні потреби користувачів та виявляють ключові проблеми, які потребують вирішення. Це дослідження

допомагає створити чітке уявлення про завдання, що стоять перед проектом, та способи їх вирішення.

Наступним кроком є визначення вимог до програмного забезпечення. Цей етап є надзвичайно важливим, оскільки від правильного визначення вимог залежить успішність усього проекту. Збір вимог включає як функціональні вимоги, що визначають конкретні дії, які програма повинна виконувати, так і нефункціональні вимоги, що включають показники продуктивності, безпеки, масштабованості тощо.

Функціональні вимоги описують основні функції, які повинна виконувати програма, зокрема, взаємодію з користувачами, обробку даних та інші операції. Нефункціональні вимоги визначають якісні характеристики системи, такі як швидкість виконання, безпека, надійність та інші показники, які забезпечують ефективність роботи програмного забезпечення.

Після визначення вимог здійснюється розгляд можливих способів реалізації функцій. Це включає вибір найбільш відповідних технологій та інструментів, які сприятимуть досягненню поставлених цілей. Кожен підхід аналізується з точки зору його переваг та недоліків, відповідності вимогам проекту та потенційних ризиків.

Вибір технологій та інструментів є критичним етапом проектування, оскільки від нього залежить ефективність, надійність та масштабованість системи. Розробники повинні враховувати різні аспекти, включаючи сумісність з існуючими системами, можливості масштабування, простоту використання та підтримки.

Одним з найважливіших аспектів проектування є розробка архітектури програмного забезпечення. Архітектура визначає структуру системи, способи взаємодії між її компонентами, а також основні принципи та стандарти, яких слід дотримуватися під час реалізації.

Архітектура програмного забезпечення є основою для всієї системи. Вона визначає, як різні компоненти системи взаємодіятимуть між собою, як буде організовано обробку даних та які технології будуть використовуватися. Важливо

забезпечити ефективність, зручність у підтримці та масштабованість розподілу функціональних компонентів системи.

Коректне проектування програмно-технічного засобу є вирішальним фактором успішної реалізації проекту. Вдало спроектована система дозволяє вирішити низку питань ще до початку її розробки, знижує ризики, уникає потенційних проблем та забезпечує високу якість кінцевого продукту.

Аналіз предметної області дозволяє виявити основні потреби користувачів і визначити ключові проблеми, які повинні бути вирішені. Визначення вимог до програмного забезпечення є наступним кроком, який забезпечує чітке розуміння функцій і характеристик, які система повинна мати. Це включає як функціональні, так і нефункціональні вимоги, що гарантують ефективність і безпеку системи.

Обґрунтування технічних рішень та вибір відповідних архітектурних підходів є центральними аспектами проектування. Розглядаються різні варіанти реалізації, оцінюються їх переваги та недоліки, а також потенційні ризики. Вибрані рішення повинні забезпечувати сумісність компонентів, легкість у підтримці та можливість масштабування системи.

Розробка архітектури програмного забезпечення включає деталізацію компонентів системи та визначення способів їх взаємодії. Це забезпечує чітку структуру системи, яка є основою для подальшої реалізації та тестування. Важливо також враховувати питання безпеки та захисту даних, інтегруючи відповідні механізми з самого початку.

Таким чином, проектування є не просто першим кроком у створенні інформаційного продукту, але й фундаментом, на якому будується вся система. Науково обґрунтований підхід до проектування допомагає уникнути багатьох проблем на етапі реалізації та забезпечує високий рівень якості кінцевого продукту. Успіх проекту значною мірою залежить від того, наскільки добре було виконано етап проектування, оскільки саме на цьому етапі закладаються основи для стабільної, продуктивної та надійної системи.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

Данний розділ включає написання програмного коду, розгортання необхідних середовищ для виконання програми та налаштування апаратних засобів, що будуть використовуватися для роботи програми.

Однак просто створення програмного забезпечення недостатньо. Важливо переконатися, що розроблений продукт працює правильно та відповідає вимогам, встановленим на етапі аналізу та проектування. Тому велику увагу буде приділено тестуванню, яке допоможе виявити та виправити можливі помилки та недоліки у програмному забезпеченні перед введенням його в експлуатацію.

Для реалізації програмно-технічного засобу було вирішено використовувати мову програмування C++. Цей вибір обумовлений рядом важливих переваг, які надає C++ у порівнянні з іншими мовами програмування.

По-перше, C++ є однією з найбільш продуктивних мов програмування. Це забезпечується оптимізаціями на рівні компілятора, які дозволяють виконувати складні обчислювальні операції швидше та ефективніше. C++ також надає розробнику контроль над управлінням пам'яттю, що дозволяє мінімізувати витрати ресурсів та покращити швидкодію програми.

Це особливо важливо в контексті паралельних обчислень, де ефективність використання ресурсів критично важлива.

Другою вагомою перевагою є підтримка паралельного програмування. C++ має вбудовану підтримку таких бібліотек для паралельного програмування, як MPI (Message Passing Interface), OpenMP, TBB (Threading Building Blocks) та інші. Це дозволяє ефективно реалізовувати розподілені та багатопоточні програми, що є ключовою вимогою для сучасних високопродуктивних обчислень. Контроль за потоками в C++ дає можливість розробникам створювати, синхронізувати та управляти потоками вручну, що дозволяє реалізовувати складні алгоритми паралельної обробки даних.

Третя перевага C++ полягає в його сумісності з низькорівневими операціями. Це забезпечується доступом до апаратних ресурсів на рівні операційної системи, що дозволяє оптимізувати виконання коду на рівні заліза. В C++ можна використовувати інструкції процесора, маніпулювати регістрами та виконувати інші низькорівневі операції, які недоступні в багатьох високорівневих мовах програмування. Це дає змогу досягати максимальної продуктивності та ефективності програм.

Ще однією значною перевагою є широке поширення та підтримка C++. Це означає, що існує велика кількість готових бібліотек, інструментів та фреймворків, які можуть бути використані в розробці.

Наприклад, бібліотеки для роботи з мережею, обробки графіки, роботи з базами даних та інші вже давно існують і активно використовуються розробниками по всьому світу. Це значно скорочує час розробки та дозволяє зосередитись на реалізації основної функціональності програмного забезпечення.

Окрім переваг, слід також зазначити деякі недоліки використання C++.

По-перше, це складність мови. C++ є однією з найскладніших мов програмування через велику кількість синтаксичних конструкцій та особливостей. Розробка на C++ вимагає високого рівня кваліфікації та досвіду, що може бути перешкодою для новачків.

По-друге, ручне управління пам'яттю може призводити до помилок, таких як витоки пам'яті або неправильне звільнення ресурсів. Це потребує додаткової уваги та ретельного тестування коду.

Ще одним недоліком є тривала компіляція. Великі проекти на C++ можуть компілюватись досить довго, що ускладнює процес розробки та налагодження. Крім того, відсутність вбудованих засобів для роботи з сучасними високорівневими концепціями, такими як функціональне програмування, також може бути обмеженням у деяких випадках.

Таким чином, вибір C++ для реалізації програмно-технічного засобу обумовлений його високою продуктивністю, широкими можливостями для

паралельного програмування, сумісністю з низькорівневими операціями та великою кількістю доступних бібліотек.

Попри деякі недоліки, такі як складність мови та потреба в ручному управлінні пам'яттю, переваги C++ роблять його оптимальним вибором для задач, які вимагають високої продуктивності та ефективного використання апаратних ресурсів.

Серед різних бібліотек для паралельного програмування, таких як OpenMP, TBB (Threading Building Blocks) та інші, було вирішено обрати MPI (Message Passing Interface) для реалізації нашого програмно-технічного засобу. Це рішення було прийняте на основі ряду вагомих причин, які роблять MPI оптимальним вибором для даного проекту.

OpenMP є простою у використанні бібліотекою, яка використовує директиви для керування паралельністю, що значно спрощує написання та підтримку коду. Вона дозволяє програмістам швидко додавати паралельність до існуючого послідовного коду, використовуючи директиви, які включаються безпосередньо в код. Це забезпечує зручний та ефективний спосіб перетворення послідовних програм у паралельні, з мінімальними змінами до вихідного коду.

Однак, OpenMP надає менший контроль над комунікацією та розподілом задач порівняно з іншими методами паралельного програмування, що може бути недоліком для складних застосувань. Наприклад, в OpenMP немає таких детальних механізмів для управління потоками та їх синхронізацією, як у інших моделей паралельного програмування, таких як MPI (Message Passing Interface). Це може призвести до обмежень у продуктивності для програм, які потребують точного контролю над розподілом задач і синхронізацією між потоками.

OpenMP оптимально підходить для систем з архітектурою спільної пам'яті, де всі процесори мають доступ до одного і того ж простору пам'яті. Це дозволяє ефективно розподіляти задачі між потоками без необхідності передавати дані між різними пам'ятями. Проте, OpenMP погано масштабується на великі кластерні системи з розподіленою пам'яттю, де кожен процесор має свою власну локальну

пам'ять. У таких випадках використання OpenMP може призвести до значних втрат продуктивності через високі витрати на комунікацію між процесорами.

Налаштування OpenMP для досягнення максимальної продуктивності може бути складним завданням. Хоча додавання директив для паралельності є відносно простим, оптимізація коду для максимально ефективного використання ресурсів вимагає глибшого розуміння архітектури системи та поведінки паралельних алгоритмів. Наприклад, потрібно враховувати питання балансування навантаження, уникнення конфліктів доступу до спільних ресурсів та ефективне використання кеш-пам'яті.

В цілому, OpenMP є потужним інструментом для швидкого додавання паралельності до існуючих програм, особливо в системах зі спільною пам'яттю. Однак, для досягнення максимальної продуктивності та ефективного використання ресурсів у складних застосуваннях або на кластерних системах з розподіленою пам'яттю можуть бути потрібні додаткові зусилля та розгляд інших методів паралельного програмування.

TBB, з іншого боку, забезпечує продуктивність та простоту за рахунок автоматичного керування розподілом задач між потоками. Це спрощує розробку паралельних програм, оскільки програмісту не потрібно вручну розподіляти задачі між потоками або управляти їх синхронізацією. TBB використовує високий рівень абстракції для автоматичного керування задачами, що дозволяє швидко та ефективно створювати паралельні програми.

Однак, надання меншого контролю над точним розподілом задач може бути недоліком для деяких високопродуктивних застосувань, де необхідна детальна оптимізація. У таких випадках автоматичне керування задачами може не забезпечити оптимальної продуктивності, оскільки програміст не має можливості точно налаштувати розподіл задач відповідно до специфічних вимог застосування.

TBB оптимально підходить для багатоядерних процесорів з архітектурою спільної пам'яті. Вона дозволяє ефективно використовувати всі доступні ядра процесора для паралельного виконання задач, що значно підвищує продуктивність

програм. Однак, TBB не підходить для систем з розподіленою пам'яттю, де кожен процесор має свою власну локальну пам'ять. У таких системах використання TBB може бути неефективним через відсутність підтримки комунікації між різними пам'ятями.

Високий рівень абстракції, який надає TBB, полегшує написання та підтримку коду. Це дозволяє розробникам зосередитися на логіці програми, а не на низькорівневих деталях керування потоками. Проте, цей високий рівень абстракції може обмежувати можливості оптимізації для специфічних завдань, де потрібна детальна налаштування паралельного виконання для досягнення максимальної продуктивності.

Таким чином, TBB є потужним інструментом для розробки паралельних програм на багатоядерних процесорах зі спільною пам'яттю. Вона забезпечує високу продуктивність та простоту використання за рахунок автоматичного керування задачами, але може мати обмеження для високопродуктивних застосувань та систем з розподіленою пам'яттю. Вибір між TBB та іншими методами паралельного програмування залежить від специфічних вимог застосування та архітектури системи, на якій він буде виконуватися.

Однією з найважливіших причин для вибору MPI є його висока масштабованість. MPI підтримує виконання на великій кількості процесів, що можуть бути розподілені по різних вузлах кластерів та суперкомп'ютерів. Це дозволяє ефективно використовувати ресурси великих обчислювальних систем для вирішення великих задач, які вимагають значних обчислювальних потужностей. У контексті нашого проекту, де паралельні обчислення є ключовим компонентом, можливість масштабування на тисячі і навіть мільйони процесів є надзвичайно важливою.

Гнучкість та контроль, які надає MPI, також відіграють важливу роль у його виборі. Бібліотека MPI дозволяє розробникам мати детальний контроль над комунікацією між процесами, що включає точне управління передачею повідомлень, синхронізацією процесів та розподілом задач. Така гнучкість

дозволяє оптимізувати комунікаційні патерни та ефективно розподіляти навантаження між процесами, що є критичним для досягнення високої продуктивності в паралельних програмах. Інші бібліотеки, такі як OpenMP або TBB, зазвичай використовують більш високорівневі абстракції, що можуть обмежувати можливості оптимізації.

Портативність MPI є ще одним значущим фактором. MPI є стандартом де-факто для паралельного програмування в наукових обчисленнях і широко підтримується на різних обчислювальних платформах. Це означає, що код, написаний з використанням MPI, може бути легко перенесений на інші системи без значних змін. Така портативність забезпечує гнучкість у виборі обчислювальної платформи та дозволяє використовувати MPI як на персональних комп'ютерах, так і на великих кластерних системах.

Широка підтримка та спільнота, яку має MPI, значно спрощують процес вивчення та використання бібліотеки для новачків. Велика кількість ресурсів, включаючи документацію, навчальні матеріали та підтримку з боку спільноти, є вагомою перевагою. Крім того, велика кількість наукових статей та досліджень, присвячених MPI, свідчить про його ефективність та надійність у різних застосуваннях.

Висока продуктивність, яку забезпечує MPI, робить її привабливим вибором для реалізації програмно-технічного засобу. Бібліотека надає оптимізовані механізми комунікації, що мінімізують затримки та витрати на обмін повідомленнями між процесами. Це забезпечує високий рівень продуктивності, що є необхідним для виконання складних обчислювальних задач.

Сумісність MPI з іншими інструментами та бібліотеками, що використовуються в наукових обчисленнях, є додатковою перевагою. Це включає в себе засоби для профілювання та налагодження паралельних програм, бібліотеки для роботи з великими обсягами даних та інші інструменти, що допомагають розробникам створювати ефективні та надійні програми.

Попри всі переваги, MPI має й певні недоліки. Наприклад, налаштування та оптимізація MPI для роботи на великих кластерах можуть вимагати значних зусиль. Крім того, така гнучкість та контроль над комунікацією вимагають більшої уваги до деталей і можуть призвести до складнішого коду, що важче підтримувати. Також різні реалізації MPI можуть мати свої особливості та оптимізації, що вимагає додаткових налаштувань при зміні платформи. Велика кількість ресурсів може ускладнити вибір правильного підходу для новачків, а оптимізація комунікаційних патернів може вимагати значних зусиль та експериментів.

Таким чином, вибір MPI серед інших бібліотек для паралельного програмування обумовлений його масштабованістю, гнучкістю та контролем над комунікацією, портативністю, широкою підтримкою та спільнотою, високою продуктивністю та сумісністю з іншими інструментами. OpenMP та TBB також мають свої переваги, такі як простота використання та швидка розробка, але їхні обмеження в масштабованості та контролі роблять їх менш підходящими для великих та складних систем з розподіленою пам'яттю. MPI, з іншого боку, надає необхідні інструменти та можливості для створення ефективних паралельних програм, які можуть масштабуватись на великі обчислювальні ресурси.

Робота з MPI у середовищі Visual Studio на локальному комп'ютері

Налагодження, компіляція та запуск програм, написаних на MPI, на локальній машині здійснюється за допомогою програми mpiexec.exe. У цій статті покажемо, як встановити цю програму та зв'язати її з Visual Studio (на прикладі MPI v10.1.2 та VS 2017). Після коректної установки програми на MPI компілюватимуться і запускатимуться як з командного рядка, так і з середовища Visual Studio.

Для роботи зі стандартом MPI необхідно встановити:

Microsoft Visual Studio - це інтегроване середовище розробки (IDE), розроблене компанією Microsoft. Воно надає розробникам широкий набір інструментів для створення програмного забезпечення на різних мовах програмування, включаючи C++, C#, Visual Basic і багато інших. Visual Studio має

багато функцій, таких як розширене налагодження, автоматизоване тестування, інтегрована система керування версіями, інструменти для роботи з веб-розробкою та мобільними додатками.

msmpisetup.exe - це утиліта, яка встановлює MPI (Message Passing Interface) для Windows у вашій системі. MPI є стандартом для розподіленого обчислення та обміну повідомленнями між різними процесами. msmpisetup.exe забезпечує зручний спосіб встановлення та налаштування MPI на вашому комп'ютері, щоб ви могли розробляти та виконувати паралельні програми.

msmpisdk.msi - це пакет розробника MPI, який містить набір бібліотек, заголовкових файлів і інструментів, необхідних для розробки програм, що використовують MPI. Цей пакет дозволяє розробникам легко і швидко інтегрувати MPI в свої програми та створювати паралельні додатки для Windows. Встановлення msmpisdk.msi дозволяє розпочати роботу з MPI у ваших проектах розробки програмного забезпечення, що вимагають розподіленого обчислення.

Таблиця 3.1 - Посилання на ресурси Microsoft

1	Microsoft Visual Studio	https://visualstudio.microsoft.com/whats-new-visual-studio-2019%3Fview%3Dvs-2019
2	msmpisetup.exe	https://www.microsoft.com/en-us/download/details.aspx?id=57467
3	msmpisdk.msi	

Після встановлення бажано перевірити, що на вашому комп'ютері встановлені всі компоненти MS-MPI, а саме:

- …\Microsoft MPI\Bin\
- …\Microsoft SDKs\MPI\Include\
- …\Microsoft SDKs\MPI\Lib\x86\ и/или …\Microsoft SDKs\MPI\Lib\x64\

Для цього:

- запустіть командний рядок від імені адміністратора;
- наберіть команду set MSMPI.

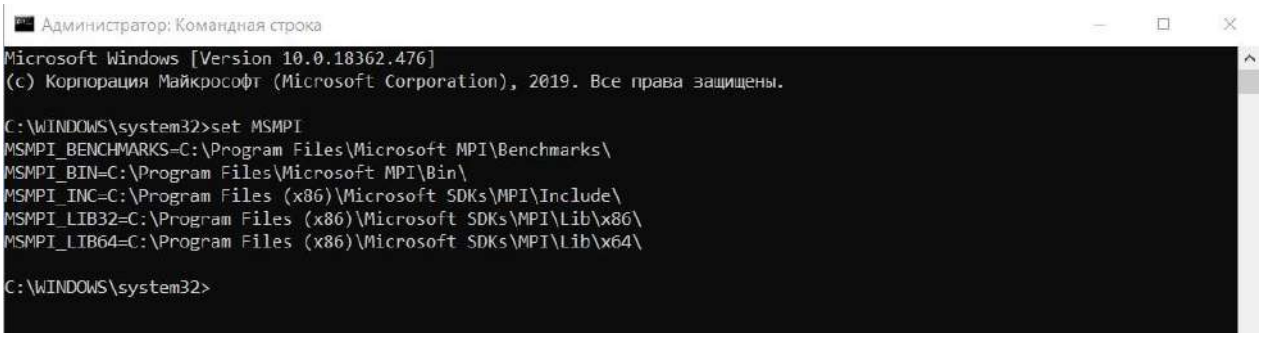


Рисунок 3.1 – Результат виконання команди

На зображенні показана командна строка Windows, де налаштовані змінні середовища для Microsoft MPI. Нижче наведено примітка: компонента:

…\Microsoft MPI\Benchmarks\ може бути відсутнім.

Далі буде детально розглядаться програмно-апаратна реалізація та тестування програмно-технічного засобу, призначеного для забезпечення функціонування проміжного програмного забезпечення розподілених систем зі спеціалізованим функціоналом. Проміжне програмне забезпечення виконує роль зв’язуючої ланки між різними компонентами розподіленої системи, забезпечуючи ефективну взаємодію та управління ресурсами.

Основна мета розробки такого засобу полягає в створенні інструменту, який може оптимізувати процеси обміну даними, балансування навантаження та забезпечення надійності системи. Впровадження такого програмно-технічного засобу дозволяє досягти високої продуктивності та масштабованості системи, що особливо важливо для обробки великих обсягів даних і підтримки складних обчислювальних задач.

Оптимізація процесів обміну даними дозволяє зменшити затримки та підвищити швидкість доступу до необхідної інформації, що є критично важливим для забезпечення ефективної роботи системи. За рахунок ефективного управління потоками даних можна досягти значного підвищення продуктивності, що, в свою чергу, дозволяє системі обробляти більше запитів у одиницю часу.

Балансування навантаження є ще одним ключовим аспектом, який забезпечує рівномірний розподіл ресурсів між різними компонентами системи. Це дозволяє уникнути перевантаження окремих компонентів та забезпечує стабільну роботу всієї системи. Завдяки цьому можна гарантувати, що система буде здатна обробляти підвищене навантаження без втрати продуктивності.

Надійність системи досягається за рахунок впровадження механізмів резервування та відмовостійкості. Це включає дублювання критично важливих компонентів, що дозволяє системі продовжувати роботу навіть у разі відмови одного з елементів. Такий підхід забезпечує високу доступність та надійність, що є важливим для систем, які працюють у режимі реального часу або обробляють критично важливу інформацію.

Масштабованість системи забезпечується завдяки можливості додавання нових компонентів або збільшення потужностей існуючих без значних змін у загальній архітектурі. Це дозволяє системі рости та адаптуватися до змінних вимог бізнесу, забезпечуючи можливість обробки все більших обсягів даних та виконання все складніших завдань.

Впровадження програмно-технічного засобу, який оптимізує процеси обміну даними, балансування навантаження та забезпечення надійності, є критичним для досягнення високої продуктивності та масштабованості системи. Це особливо важливо для сучасних інформаційних систем, що працюють з великими обсягами даних та складними обчислювальними задачами, де ефективність і надійність є ключовими факторами успіху.Процес створення програмно-технічного засобу на основі мови програмування C++ та бібліотеки MPI (Message Passing Interface) включає кілька важливих етапів. Кожен етап відіграє ключову роль у забезпеченні ефективності та продуктивності паралельних обчислень. Нижче наводиться детальний опис кожного з цих етапів та пояснення функцій, які використовуються в коді.

На початку роботи програми необхідно ініціалізувати середовище MPI. Це здійснюється за допомогою функції MPI_Init, яка приймає аргументи командного рядка та готує середовище для паралельного виконання програми (рис. 3.2).

```
MPI_Init(&argc, &argv);
```

Рисунок 3.2 - Ініціалізація середовища MPI

Ініціалізація здійснюється за допомогою функції MPI_Init, яка приймає аргументи командного рядка та запускає необхідні процеси для паралельного виконання програми.

Далі кожен процес отримує свій унікальний ідентифікатор (ранг) та загальну кількість процесів у комунікаторі за допомогою функцій MPI_Comm_rank та MPI_Comm_size. Це дозволяє процесам взаємодіяти між собою та виконувати свої завдання. (рис. 3.3).

```
int rank, size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

Рисунок 3.3 - Ініціалізація середовища MPI

Функція MPI_Comm_rank визначає ранг процесу у вказаному комунікаторі, а MPI_Comm_size визначає загальну кількість процесів у комунікаторі.

Для забезпечення унікальності випадкових чисел кожного процесу використовується поточний час як зерно для генератора випадкових чисел. До цього зерна додається ранг процесу, що гарантує різні випадкові числа для кожного процесу (рис. 3.4).

```
unsigned int seed = std::chrono::system_clock::now().time_since_epoch().count();
srand(seed + rank);
```

Рисунок 3.4 - Генерація випадкових даних

Використання std::chrono::system_clock::now().time_since_epoch().count() для отримання поточного часу забезпечує унікальність зерна на основі часу. Функція srand встановлює зерно для генератора випадкових чисел, а rand генерує випадкове число.

Для взаємодії між процесами було обрано схему, за якою головний процес (ранг 0) генерує дані та відправляє їх до всіх інших процесів. Головний процес використовує функцію MPI_Send для відправки даних, а робочі процеси отримують ці дані за допомогою функції MPI_Recv. Це дозволяє організувати чітку комунікацію між процесами та забезпечити правильний розподіл даних (рис. 3.5).

```
if (rank == 0) {
    for (int dest = 1; dest < size; ++dest) {
        int data = dest * 10;
        MPI_Send(&data, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
        std::cout << "The main process sent data " << data << " to the process " << dest << std::endl;
    }
}
```

Рисунок 3.5 - Передача повідомлень між процесами

Кожен робочий процес отримує дані від головного процесу та обробляє їх, генеруючи випадкове число. Для вимірювання часу обробки даних використовується функція MPI_Wtime, яка дозволяє точно визначити початок та кінець обчислень. Результати обробки, включаючи отримані дані, час обробки та згенеровані випадкові числа, записуються у текстовий файл (рис. 3.6).

```
auto start_time = MPI_Wtime();
int random_data = rand() % 100;
std::cout << "Process " << rank << " processed random data: " << random_data << std::endl;
auto end_time = MPI_Wtime();
double elapsed_time = end_time - start_time;
std::ofstream outfile("output.txt", std::ios_base::app);
```

Рисунок 3.6 - Обробка даних

Після обробки даних кожен процес зберігає результати у файл. Потім файл

відкривається для читання, і збережені дані виводяться на екран для перевірки коректності запису. Це дозволяє переконатися, що всі дані були правильно збережені та можуть бути використані для подальшого аналізу (рис. 3.7).

```
std::ifstream infile("output.txt");
if (infile.is_open()) {
    std::string line;
    while (std::getline(infile, line)) {
        std::cout << line << std::endl;
    }
}
```

Рисунок 3.7 - Обробка даних

Після виконання всіх обчислень та збереження результатів робота програми завершується викликом функції MPI_Finalize, яка закриває середовище MPI та звільняє всі ресурси, що були використані під час виконання програми. Це забезпечує коректне завершення роботи програми та запобігає можливим витокам пам'яті.

```
MPI_Finalize();
return 0;
```

Рисунок 3.8 – Завершення ініціалізації MPI

На рис. 3.9 зображено результат роботи програми роботи програми. Рисунок ілюструє основні етапи виконання паралельної програми, яка використовує MPI для обміну повідомленнями між процесами.

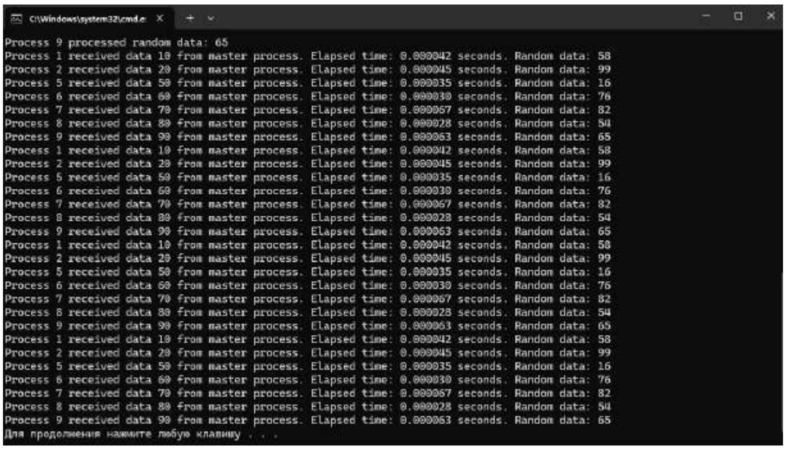


Рисунок 3.9 – Результат роботи програми

Зображення 3.10 показує вивід програми з декількома процесами, які отримують дані від головного процесу.

Кожен рядок виводу містить детальну інформацію про номер процесу, отримані дані, час виконання та випадкові дані. Це свідчить про коректну роботу програми та правильний запис у файл. У кожному рядку виводу чітко вказано номер процесу, що дозволяє легко ідентифікувати конкретний процес серед інших. Отримані дані є важливими показниками, що відображають результати роботи процесу, а час виконання допомагає оцінити ефективність та швидкодію програми. Додатково, включення випадкових даних виводу підтверджує правильність генерації та обробки інформації програмою, а також правильний запис у файл.

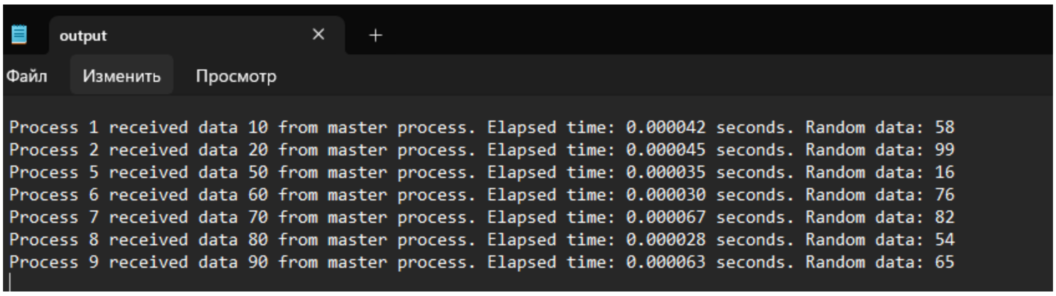


Рисунок 3.10 - Результат у текстовому файлі

Результати тестування програми показали її коректну та стабільну роботу в умовах паралельного обчислення. Всі етапи виконання, включаючи ініціалізацію середовища MPI, генерацію випадкових даних, передачу повідомлень між

процесами, обробку даних, збереження та зчитування результатів, були успішно виконані.

Передача даних між головним та робочими процесами здійснюється правильно. Головний процес успішно відправляє згенеровані дані до кожного робочого процесу, а робочі процеси правильно отримують та виводять ці дані. Вивід підтверджує, що дані, відправлені головним процесом, коректно досягають своїх цілей без втрат та спотворень.

Випадкові дані, згенеровані кожним процесом, є унікальними, що було забезпечено використанням поточного часу як зерна генератора випадкових чисел з додаванням рангу процесу. Це підтверджується виведеними результатами, де кожен процес демонструє різні випадкові числа. Такий підхід забезпечує необхідну різноманітність випадкових даних, що є важливим для паралельних обчислень.

Час обробки даних вимірювався за допомогою функції MPI_Wtime, і результати показали, що процеси завершують свої обчислення в очікуваних межах часу. Це підтверджує ефективність алгоритму та коректність його реалізації. Виведені значення часу виконання свідчать про стабільність та продуктивність програми при паралельному виконанні.

Збереження оброблених даних у текстовий файл відбулося успішно. Кожен рядок у файлі містить номер процесу, отримані дані, час виконання та випадкові дані, що підтверджує правильність роботи програми. Зчитування файлу також підтвердило, що всі дані були записані та збережені коректно, що дозволяє використовувати ці результати для подальшого аналізу.

Робота програми продемонструвала високу продуктивність, стабільність та коректність виконання. Передача даних, генерація випадкових чисел, вимірювання часу обробки та збереження результатів були здійснені правильно, що підтверджується виведеними результатами та аналізом збережених даних. Програма успішно виконує поставлені завдання в умовах паралельного обчислення, що свідчить про її ефективність та готовність до використання в реальних обчислювальних задачах.

Таким чином, створення програмно-технічного засобу включало ініціалізацію середовища MPI, генерацію випадкових даних, передачу повідомлень між процесами, обробку даних, збереження та зчитування результатів, а також завершення роботи середовища MPI. Кожен етап був ретельно спланований та реалізований з урахуванням специфіки паралельного програмування, що дозволило досягти високої продуктивності та ефективності обчислень.

У цьому розділі детально розглядається програмно-апаратна реалізація та тестування програмно-технічного засобу, призначеного для забезпечення функціонування проміжного програмного забезпечення розподілених систем зі спеціалізованим функціоналом. Проміжне програмне забезпечення виконує роль зв'язуючої ланки між різними компонентами розподіленої системи, забезпечуючи ефективну взаємодію та управління ресурсами.

Основна мета розробки такого засобу полягає в створенні інструменту, який може оптимізувати процеси обміну даними, балансування навантаження та забезпечення надійності системи. Впровадження такого програмно-технічного засобу дозволяє досягти високої продуктивності та масштабованості системи, що особливо важливо для обробки великих обсягів даних і підтримки складних обчислювальних задач. У процесі реалізації програмного засобу були використані сучасні технології та методи програмування, що дозволили забезпечити високу якість коду та зручність його використання. Апаратна частина системи включала використання високопродуктивних серверів та мережевих комунікацій, що забезпечували необхідну пропускну здатність та низькі затримки передачі даних. Тестування програмного засобу проводилось з метою перевірки його функціональності, продуктивності та надійності. Для цього були розроблені спеціальні тестові сценарії, що імітували різні умови експлуатації системи. Результати тестування показали, що розроблений засіб здатен ефективно виконувати свої функції в реальних умовах експлуатації, забезпечуючи стабільну роботу розподіленої системи.

Таким чином, розроблений програмно-технічний засіб для проміжного

програмного забезпечення розподілених систем зі спеціалізованим функціоналом повністю відповідає поставленим вимогам і може бути рекомендований для використання в різних галузях, де необхідна обробка великих обсягів даних та підтримка складних обчислювальних процесів.

ВИСНОВКИ

У першому розділі "Дослідження предметної області та постановка задачі" проведено комплексний аналіз об'єкта дослідження. Початковий змістовний аналіз дозволив ретельно розібратися у внутрішній структурі та основних функціях предметної області. Цей етап дослідження дозволив зрозуміти, які саме процеси, функції та задачі відіграють ключову роль, а також виявити можливі проблеми та обмеження, з якими може стикнутися система автоматизації. Після цього був проведений докладний аналіз наявного програмно-апаратного забезпечення в рамках предметної області. Це дозволило виявити як сильні сторони, так і потенційні недоліки чи обмеження існуючих рішень. Особлива увага приділялася визначенню тих аспектів, які вже вирішені з високою ефективністю, а також тих, які потребують подальшого удосконалення. На основі аналізу предметної області та існуючого програмно-апаратного забезпечення були сформульовані вимоги до системи автоматизації. Це включало в себе не лише технічні вимоги до функціональності системи, але й вимоги щодо її надійності, швидкодії, масштабованості та інших аспектів. На цій основі було розроблено технічне завдання, яке стало основою для подальшої розробки програмно-технічного засобу.

У другому розділі "Проектування програмно-технічного засобу" подробно розглядаються результати проектування системи автоматизації на основі вимог, що були визначені у попередньому розділі. Основною метою цього етапу є розробка концепції програмно-технічного засобу, яка відповідає потребам та вимогам користувачів. На першому етапі проектування розглядаються архітектурні рішення. Це означає вибір загальної структури системи, включаючи розподіл функцій між компонентами системи, взаємозв'язки між ними та способи комунікації. Наприклад, визначається, чи буде система монолітною, мікросервісною або заснованою на інших архітектурних підходах. Далі виконується вибір технологій та інструментів для реалізації системи. Це може включати вибір мов програмування, фреймворків, баз даних, інструментів для

версійного контролю та інших технологій, які найбільш відповідають вимогам проекту. Крім того, у цьому розділі описуються як функціональні, так і нефункціональні вимоги до програмно-технічного засобу. Функціональні вимоги описують конкретні функції та операції, які повинна виконувати система. Нефункціональні вимоги визначають якості системи, такі як продуктивність, безпека, надійність, масштабованість та інші аспекти, які не пов'язані безпосередньо з функціональністю, але впливають на її загальну ефективність та якість.

У третьому розділі "Програмно-апаратна реалізація та тестування програмно-технічного засобу" детально розглядаються результати фактичної реалізації проекту. Цей етап включає в себе не лише написання програмного коду, а й інтеграцію програмної частини з апаратним забезпеченням, якщо така є. Одним з ключових аспектів цього розділу є опис архітектури програмно-технічного засобу. Тут наводиться докладний опис структури системи, включаючи компоненти, модулі та їх взаємозв'язки. Описується, як саме програмні компоненти взаємодіють між собою та з апаратним забезпеченням для досягнення поставлених цілей та виконання функціональних вимог. Також важливим аспектом є процес розробки програмного коду. Тут розглядається вибір методології розробки, використані технології та інструменти для написання коду, а також організація робочих процесів та комунікації в команді розробників. Крім того, у цьому розділі проводиться тестування розробленого засобу з метою перевірки відповідності вимогам, визначеним на попередніх етапах проекту. Це може включати функціональне тестування для перевірки правильності реалізації функцій, а також навантажувальне тестування для перевірки продуктивності та стійкості системи за великими навантаженнями. Тестування також спрямоване на виявлення можливих помилок чи недоліків, які потребують виправлення.

Загальною метою проекту було створення ефективної та надійної системи автоматизації, яка відповідає потребам та вимогам користувачів у відповідності з результатами дослідження предметної області.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бойчук Н.Я., Орел В.В. Оптимізація управління бізнес – процесами на підприємствах України. Київ. 2016. 178 с.

2. Брила А.Ю., Антосяк П.П., Глебена М.І., Чупов С.В., Семйон І.В. Основи програмування у C#. Методичні вказівки до лабораторних робіт для студентів І-го курсу математичного факультету спеціальності прикладна математика. Ужгород, 2014. 60 с.

3. Бруцман Дон., Зіда Майкл. Обґрунтування проектування магістралі кіберпростору (CBone): матеріали 15-го семінару зі стандартів для DIS, Орландо, Флорида, 2021. 52 с.

4. Буров Є.В., Митник М.М., Комп'ютерні мережі. Том 1: навч. посіб. Львів: Магнолія 2006, 2019. 256 с.

5. Буров Є.В., Митник М.М., Комп'ютерні мережі. Том 2: навч. посіб. Львів: Магнолія 2006, 2019. 334 с.

6. Лебеденко Ю.О., Вакаров М.М., Крайнов В.Є. *V Всеукраїнська науковопрактичної конференції студентів, аспірантів та молодих вчених з автоматичного управління присвяченої дню космонавтики*. матеріали *міжнар. наук.-практ. конф*. 12 квітня 2017 р. м. Херсон, 2017. С. 27-30

7. Ватсен, Кент і Зіда, Майкл. Bamboo - портативна система для динамічно розширюваних, мережевих, віртуальних середовищ реального часу, у Proceedings of VRAIS 98*: матеріали міжнар. наук.-практ. конф., Атланта, Джорджія, 16 березня 1998 р. Атланта, Джорджія*, 1998. С. 252- 259.

8. Волк М.О., Бергер В.С., Ткаленко О.В., Саранча С.М. Методи та засобирозподіленого імітаційного моделювання інформаційних систем. *Проблемиінформатизації: матеріали дев'ятої міжнародної науково-технічної конференції. м. Черкаси, м. Баку, м. Бельсько-Бяла, м. Харків*, 18 листопада 2021 р. Харків, 2021. 82 с.

9. Гілл К., Д.Л. Левін, Д.К. Шмідт. Розробка та ефективність службипланування CORBA в реальному часі. Real-Time Systems, *The International Journal of Time-Critical Computing Systems*. 2001. 5 березня, № 20. С. 16-20

10. Грайворонський М.В. Сучасні підходи до забезпечення кібернетичної безпеки: матеріали *XVII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики». м. Київ, 18 травня 2015 р*. Київ: КПІ. 2015. С. 118-124.

11. Дегтярьова Л.М., Гроза П.М., Сомов С.В. Технології розробки програмного забезпечення: навч. посіб. Полтава: ПНТУ, 2017. 218 с.

12. Дорш М., Сковронський В. Модифікація RTI для активних мереж, матерівли весняного семінару з моделювання взаємодії. Орландо, Флорида. 12 травня 2001р. Орландо, 2001. 7 с.

13. Загородна Н. В. Обґрунтування вибору доступних програмно-апаратних засобів високопродуктивних обчислювальних систем для задач криптоаналізу. *Електроніка та системи управління*. 2015. 1 черв. (№1) С. 42–50.

14. Заікіна Т. В. Дослідження та розробка системи контролю вводу інформації при формуванні баз даних інформаційних систем. Наукові розробки молоді на сучасному етапі : тези доповідей *XVII Всеукраїнської наукової конференції молодих вчених та студентів* (м. Київ, 26 квітня 2018 р.). Київ 2018р. С. 273-274.

15. Про захист інформації в інформаційно-комунікаційних системах: Закон України від 16.12.2020 р. № 1089-IX. Дата оновлення 16.01.2024. URL: https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80#Text (дата звернення: 28.04.2024)

16. Про основні засади забезпечення кібербезпеки України: Закон України від 21.06.2018р. № 45. Дата оновлення 16.01.2024. URL: https://zakon.rada.gov.ua/laws/show/2163-19#Text (дата звернення: 28.04.2024)

17. Карпенко К. О. Програмне забезпечення для дослідницької лабораторії з функцією автоматизованого пошуку наукових заходів. *Наукові розробки молоді на сучасному етапі : тези доповідей XVII Всеукраїнської наукової конференції молодих вчених та студентів* (м. Київ, 26 квітня 2018 р.). Київ : КНУТД, 2018. С. 185-186.

18. Качинський А.Б. Безпека складних систем. Київ: Юстон, 2017. 498 с.

19. Каштальян А.С. Принцип синтезу мультикомп'ютерних систем з комбінованих антивірусних приманок і пасток та контролеру прийняття рішень для виявлення зловмисного програмного забезпеченнята комп'ютерних атак. *Вісник Хмельницького національного університету. Технічні науки.* 2023. (№ 327). С. 386-393.

20. Козіна О.А. Змішана несуперечність даних у багатохмарних системах. *Сучасні інформаційні системи. Харків* : ХПІ, 2018. Т.2. С. 23-29.

21. Security and Privacy in Cloud Computing: Technical Review. URL: https://www.mdpi.com/1999-5903/14/1/11 (дата звернення: 28.04.2024)

22. Креневич А.П. Алгоритми і структури даних: навч. посіб. Київ: ВПЦ Київський Університет, 2021. 200 с.

23. Лебеденко Ю.О. Автоматизована система віддаленого моніторінгу стану дощувальних машин. Київ, 2018. 115 с.

24. Ляшенко І.О. Європейські критерії безпеки інформаційних технологій. Харків, 2012. 84 с.

25. Микитишин А.Г., Митник М.М., Стухляк П.Д., Пасічник В.В. Комп'ютерні мережі. Книга 1. Львів: Магнолія 2006, 2013. 256 с.

26. Микитишин А.Г., Митник М.М., Стухляк П.Д., Пасічник В.В. Комп'ютерні мережі. Книга 2. Львів: Магнолія 2006, 2014. 312 с.

27. Митник М.М., Микитишин А.Г., Стухляк П.Д., Комплексна безпека інформаційних мережевих систем. навчальний посібник, Львів: Магнолія 2006, 2016. 261 с.

28. Мінухін С. В. Методи і моделі проектування на основі сучасних CASE–засобів. Навчальний посібник. Харків: ХНЕУ, 2008. 272 с.

29. Стеценко І.В. Моделювання систем: навч. посіб.; М-во освіти і науки України, Черкас. держ. технол. ун-т. Черкаси : ЧДТУ, 2010. 399 с.

30. Мухін В.Є., Волокита О.М. Розробка та реалізація політики безпеки в розподілених комп'ютерних системах. *Керуючі системи і машини*. Чернівці, 2010. С. 78-85.

31. On the Security of Today's Online Electronic Banking Systems вебсайт. URL: http://docseurope.electrocomponents.com/b8156853c.pdf (дата звернення: 28.04.2024)

32. Остапов С.Е. Технології захисту інформації: навчальний посібник. Харків: ХНЕУ, 2013. 476 с.

33. Пасічник В. В. Організація баз даних та знань. Київ: Видавнича група BHV, 2006. 384 с.

34. Резанова В. Г. Дослідження та розробка графічних програмних засобів для інтерактивного планування експерименту. *Тези доповідей IV Міжнародної науково-практичної конференції*, (м. Київ, 22 жовтня 2020 р.). Київ : КНУТД, 2020. – С. 135- 136.

35. Резанова В. Г. Розробка програмного забезпечення для малого автопідприємства, Київ : Освіта України ; ФОП Маслаков, 2020. С. 178-181.

36. Структурна модель інтелектуального агента для підтримки захищеної обробки даних в гетерогенних розподілених системах. URL: http://journals.dut.edu.ua/index.php/sciencenotes/article/view/589 (дата звернення: 28.04.2024)

37. Троелсен Є. Мова Програмування C# 2010 і платформа .NET 4.0. Миколаїв, 2011. 1392 с.

38. Трофименко О.Г., Козін О.Б., Задерейко О.В., Плачінда О.Є. Веб-технології та вебдизайн: навч. посібник. Одеса: Фенікс, 2019. 284 с.

39. Учасники проектів Вікімедіа. Проміжне програмне забезпечення. URL: https://uk.wikipedia.org/wiki/Проміжне_програмне_забезпечення (дата звернення: 28.04.2024)

40. Barrett S.F. Microchip AVR® Microcontroller Primer: Programming and Interfacing, Morgan & Claypool Publishers, 2019. 374 p.

41. Brutzman D. Zyda M., Watsen K., Macedonia M. Virtual reality transfer protocol (vrtp) Design Rationale. *Proceedings of the IEEE SixthInternational Workshop on Enabling Technologies: Infrastructure for CollaborativeEnterprises (WET ICE '97): proceedings of the Workshop Aspects of a Distributed Virtual Reality Exchange System, (Cambridge, MA, June 18, 1997).* Cambridge, Massachusetts,

P. 179-186.

42. Joseph A. C# 7.0 in a Nutshell: The Definitive Reference. 2021. 1007 c.

43. Vidyarthi D. P., Sarker B. K., Tripathi A. K. and Yang L. T., Scheduling in Distributed Computing Systems: Analysis, Design & Models (A Research Monography), Springer Science+Business Media: LLC, USA, 2009, 302 p.

44. Hussain H., Malik S., Hameed A., Ullah Khan S., Bickler G., and other, Survey on Resource Allocation in High Performance Distributed Computing Systems‖, Parallel Computing, vol. 39, issue 11, 2013. 736 p.

45. Herasymenko O. Analytical Assessment of Security Level of Distributed and Scalable Computer Systems. *International Journal of Intelligent Systems and Applications. Hong Kong: MECS Publisher*, 2016. – Vol. 8. № 12. P. 57 – 64.

46. Herasymenko O. Distributed Computer System Resources Control Mechanism Based on Network-Centric Approach. *International Journal of Intelligent Systems and Applications (IJISA). Hong Kong: MECS Publisher*, 2017. Vol. 9. № 7. P. 41-51.

47. Herasymenko O. The scheduler for the grid-system based on the parameters monitoring of the computer components. *Journal of Enterprise Technologies. Information Technology*. 2017. № 1. P. 31-39.

48. Kravets A.G., Bolshakov A.A., Shcherbakov M.V. Cyber-Physical Systems: Industry 4.0 Challenges (Studies in Systems, Decision and Control, 260), Springer; 1st ed., 2020. 349 p.

49. Qureshi M. B., Dehnavi M. M., Min-Allah N., Qureshi M.S., and other. Survey on Grid Resource Allocation Mechanisms‖, *Journal of Grid Computing,* vol. 12, issue 2, 2014. P. 399-441

50. Rahman M., Ranjan R., Buyya R. Decentralization in Distributed Systems: Challenges, Technologies, and Opportunities‖, in Advancements in Distributed Computing and Internet Technologies: Trends and Issues, A.-S. K. Pathan, M. Pathan and H. Y. Lee, Eds. *Information Science Reference (an imprint of IGI Global)*, USA, 2012, p. 386-399.

51. Monk S. Programming Arduino Next Steps: Going Further with Sketches. McGraw-Hill Education TAB, 2018. 320 p.

52. Maximum Security: A Hacker's Guide to Protecting Your Internet Site and Network. URL: http://redwave.net/books/hackg/index/html (дата звернення: 28.04.2024)

53. Nisan N., Schocken S. The Elements of Computing Systems, second edition: Building a Modern Computer from First Principles 2nd Edition, The MIT Press, 2021. 344 p.

54. Östberg P.O., Espling D., Elmroth E. Decentralized Scalable Fairshare Scheduling‖, Future Generation Computer Systems, vol. 29, issue 1, 2013. P. 130-143.

55. Singh S. P., Sharma S. Ch. A Particle Swarm Optimization Approach for Energy Efficient Clustering in Wireless Sensor Networks, *International Journal of Intelligent Systems and Applications(IJISA)*. 2017. Vol.9. №.6. p. 66-74.

56. Sánchez M. et al. Basic features of a reflective middleware for intelligent learning environment in the cloud (IECL). *2015 Asia-Pacific Conference on Computer Aided System Engineering. – IEEE*, 2015. P. 1-6.

57. Understanding difference between Cyber Security & Information Security – CISO Platform, 2016. URL: http://www.cisoplatform.com/profiles/blogs/understanding-difference-betweencyber-security-information (дата звернення: 28.04.2024)

58. Wuerfel, Roger, Performance comparison of HLA and DIS in real time, paper 98S-SIW-042, *Proceedings of Spring Simulation InteroperabilityWorkshop, Orlando*, FL, 1998. 20 p.

59. Huang Y., Bessis, Norringtonb P., Kuonend P. and Hirsbrunner B., Exploring Decentralized Dynamic Scheduling for Grids and Clouds Using the Communityaware Scheduling Algorithm‖, Future. *Generation Computer Systems,* 2013. Vol. 29, issue 1. P. 402-415.

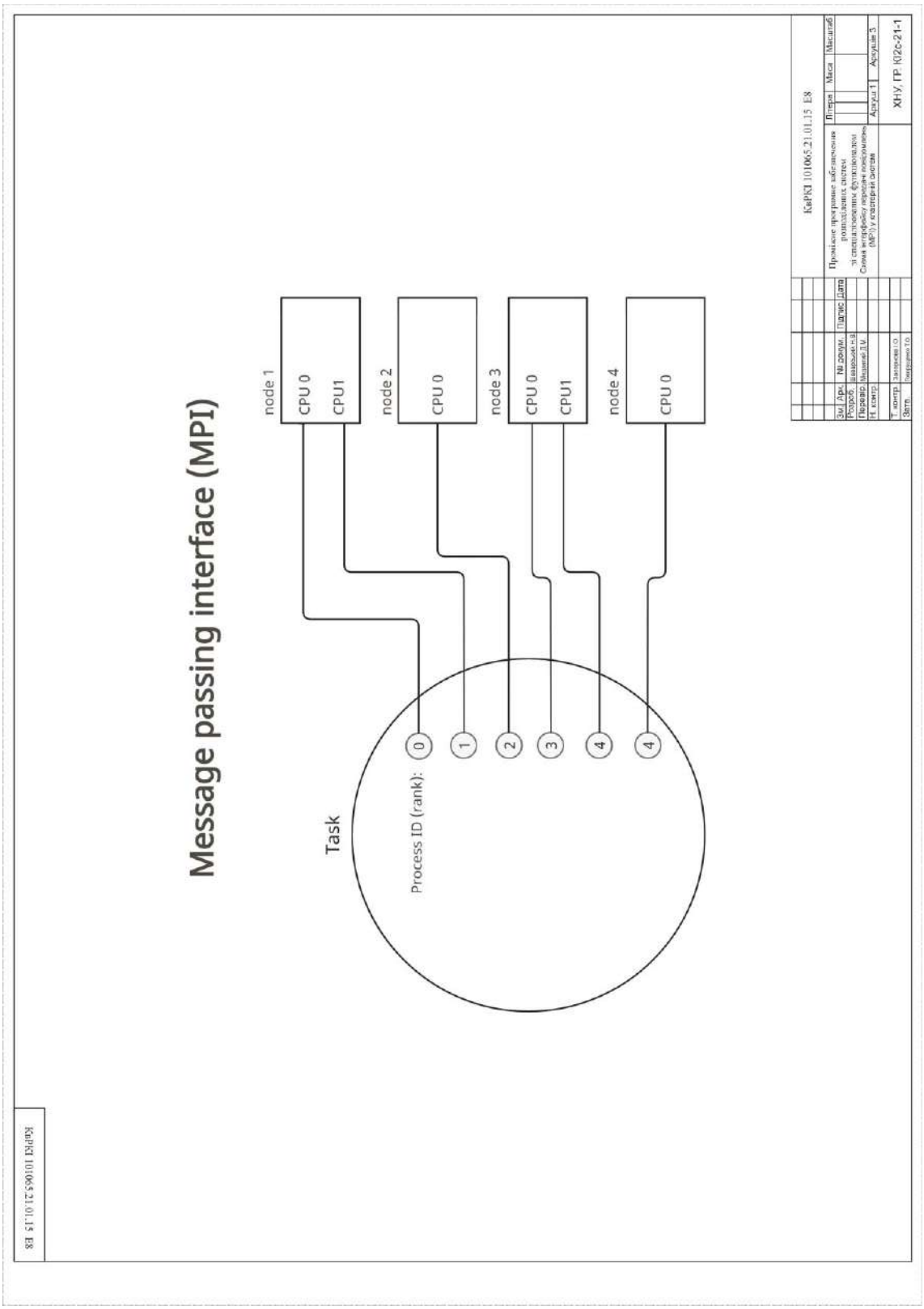
60. Zhu Y. and Ni L. M., A Survey on Grid Scheduling Systems [Online] , Technical Report SJTU_CS_TR_200309001, Department of Computer Science and Engineering. URL: http://www.cs.sjtu.edu.cn/~yzhu/reports/SJTU_CS_TR_200309001.pdf. (дата звернення: 28.04.2024)

61. Anderson J.M. Why we need a new definition of information security. Computers & Security, 2003. 308 с.

Додаток А

(обов’язковий)

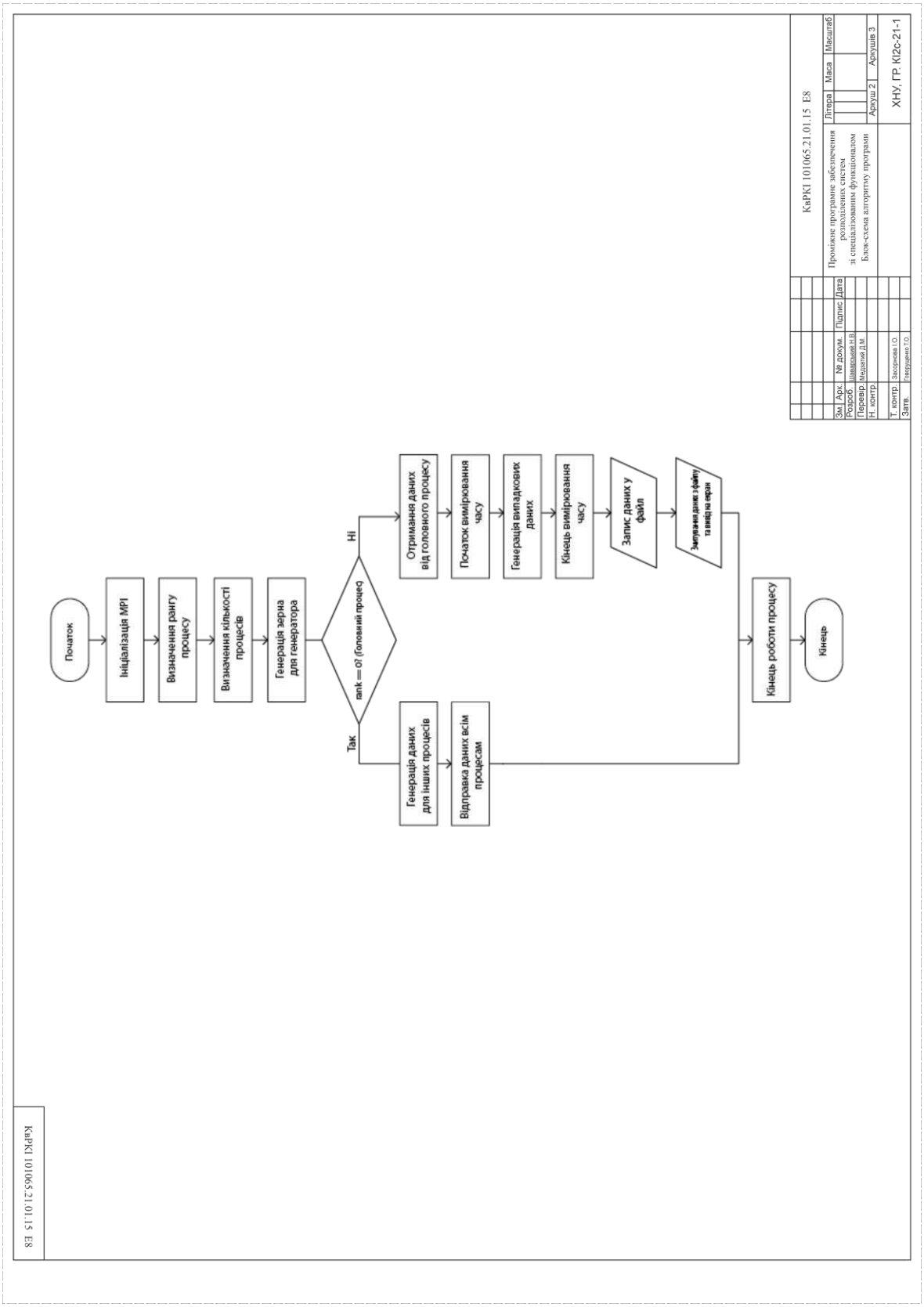
Копія креслення «Схема інтерфейсу передачі повідомлень (MPI) у кластерній системі»



Додаток Б

(обов’язковий)

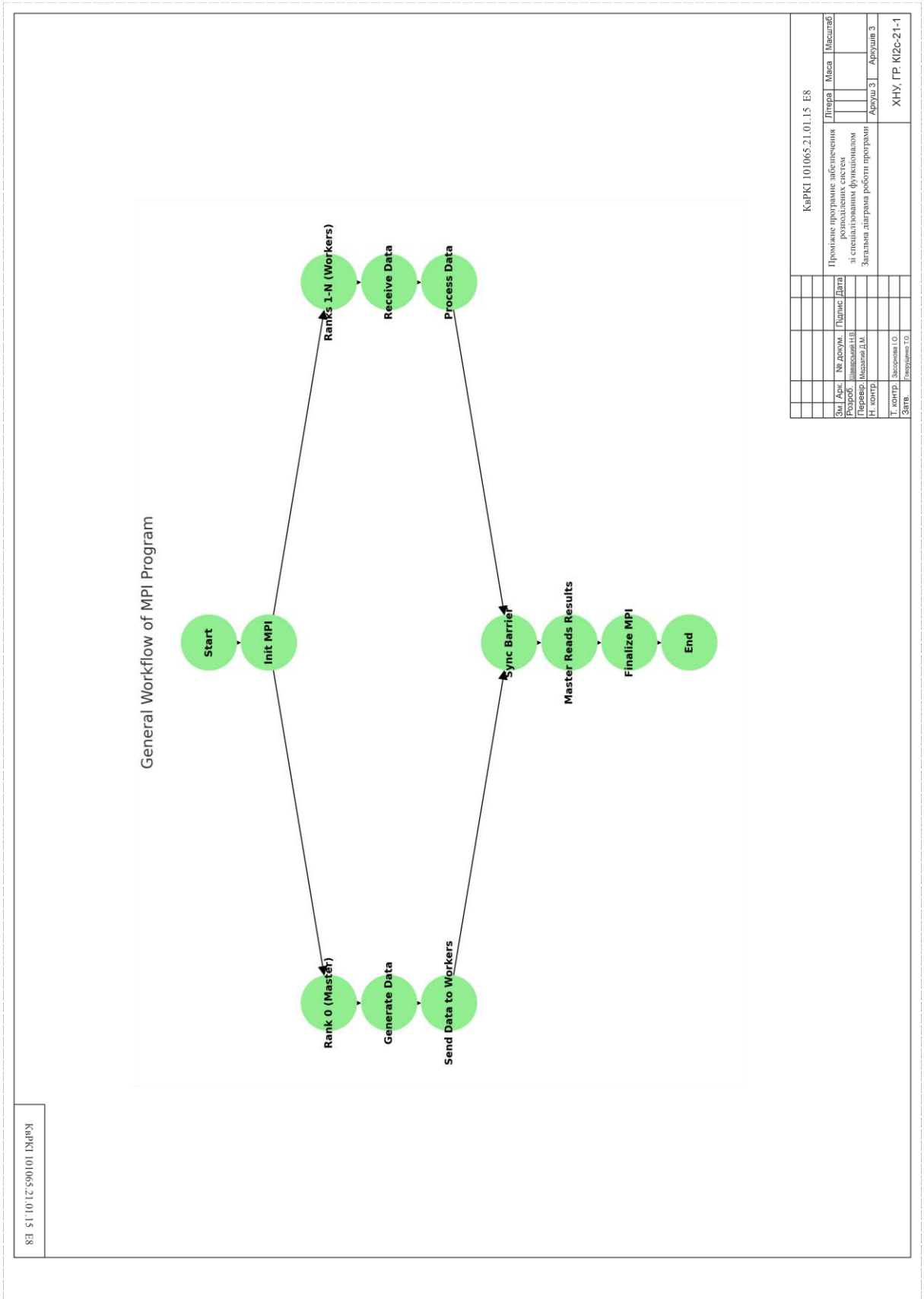
Копія креслення «Блок-схема алгоритму програми»



Додаток В

(обов’язковий)

Копія креслення «Загальна діаграма роботи програми»



Додаток Г

Код програми

```
#include <mpi.h>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <chrono>
#include <cstdlib>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Використання поточного часу як зерна для генератора випадкових чисел
    unsigned int seed = std::chrono::system_clock::now().time_since_epoch().count();
    srand(seed + rank); // Додаємо номер процесу для унікального зерна

    if (rank == 0) { // Головний вузол
        // Відправка повідомлення від головного процесу до всіх інших процесів
        for (int dest = 1; dest < size; ++dest) {
            int data = dest * 10; // Генеруємо дані для відправки
            MPI_Send(&data, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
```

```
            std::cout << "The main process sent data " << data << " to the process "
<< dest << std::endl;
        }
    }
    else { // Робочі вузли
        int received_data;
        MPI_Recv(&received_data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        std::cout << "Process " << rank << " received data " << received_data << "
from the main process" << std::endl;
        auto start_time = MPI_Wtime(); // Початок вимірювання часу

        // Обробка випадкових даних
        int random_data = rand() % 100; // Генеруємо випадкове число від 0 до
99
        std::cout << "Process " << rank << " processed random data: " <<
random_data << std::endl;
        auto end_time = MPI_Wtime(); // Закінчення вимірювання часу
        double elapsed_time = end_time - start_time; // Розрахунок часу обробки

        // Збереження отриманих даних, часу та випадкових даних в текстовий
файл
        std::ofstream outfile("output.txt", std::ios_base::app);
        if (outfile.is_open()) {
            outfile << std::fixed << std::setprecision(6); // Форматування виведення
даних з фіксованою кількістю знаків після коми
            outfile << "Process " << rank << " received data " << received_data << "
from master process. Elapsed time: " << elapsed_time << " seconds. Random data: " <<
random_data << "\n";        outfile.close();
```

```
        }
        else {
            std::cerr << "Failed to open file for writing" << std::endl;
        }

        // Відкриття файлу після збереження
        std::ifstream infile("output.txt");
        if (infile.is_open()) {
            std::string line;
            while (std::getline(infile, line)) {
                std::cout << line << std::endl;
            }
            infile.close();
        }
        else {
            std::cerr << "Failed to open file for reading" << std::endl;
        }
    }

    MPI_Finalize();
    return 0;
}
```

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016333750

Дата перевірки:
07.06.2024 22:49:16 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
07.06.2024 22:54:02 EEST

ID користувача:
100005591

Назва документа: Шаварський_Проміжне програмне забезпечення розподілених систем зі спеціалізованим ф..

Кількість сторінок: 66 **Кількість слів:** 11030 **Кількість символів:** 86633 **Розмір файлу:** 3.54 MB **ID файлу:** 1016133948

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

16.9%
Схожість

Найбільша схожість: 3.66% з Інтернет-джерелом (https://er.knutd.edu.ua/bitstream/123456789/24305/1/Dyplom151_Stru.

16.2% Джерела з Інтернету	508	Сторінка 68
3.87% Джерела з Бібліотеки	111	Сторінка 72

0.18% Цитат

Цитати	2	Сторінка 73
Посилання	1	Сторінка 73

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	214
Підозріле форматування	15 сторінок

Fri Jun 07 21:08:58 EEST 2024, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

	Документ		Сумарний збіг по Базі Даних	
ID: 129136 Назва: БКР Проміжне програмне забезпечення розподілених систем зі спеціалізованим функціоналом Додано в БД: 2024-06-07 Автора: Н. В. Шаварський Керівники: Д. М. Медзатий Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	76128	587	4292 (6%)	48 (8%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Дипломник Шаварський Назар Віталійович

Тема Проміжне програмне забезпечення розподілених систем зі спеціалізованим функціоналом

Спеціальність 123 – Комп'ютерна інженерія

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 3; кількість сторінок записки 58

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі проведено дослідження та розробка проміжного програмного забезпечення (middleware) для розподілених систем, яке має спеціалізований функціонал.

2. Висновок про відповідність кваліфікаційної роботи завданню Кваліфікаційна робота у повній мірі відповідає поставленому завданню як в теоретичній, так і в практичній частині роботи

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі подана загальна характеристика поставленої задачі, сформульована актуальність. Визначені задачі, які необхідно вирішити для досягнення поставленої мети, У першому розділі проведено огляд та аналіз сучасних технологій реалізації проміжного програмного забезпечення розподілених систем, виконана постановка задачі. В другому розділі було визначаено конкретні технічні рішення та архітектурні підходи, які будуть використовуватися для створення програмного продукту. В третьому розділі описується написання програмного коду, розгортання необхідних середовищ для виконання програми та налаштування апаратних засобів, що будуть використовуватися для роботи програми.

4. Позитивні сторони роботи Кваліфікаційна робота має практичну цінність. Практична цінність результатів дослідження полягає в високій портативність програми дозволяє її легко переносити між різними обчислювальними платформами, що робить її універсальною. Тестування показало стабільність та надійність роботи програми, підтвердивши її готовність до використання у реальних умовах..

5. Негативні сторони робот: недостатня кількість опрацьованих літературних джерел

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення виконане відповідно до теми кваліфікаційної роботи з дотриманням стандартів. В загальному графічне оформлення виконане якісно, пояснювальна записка відповідає нормам щодо її оформлення.

7. Відгук про роботу в цілому В загальному кваліфікаційна робота заслуговує позитивної оцінки. Весь матеріал кваліфікаційної роботи структурований, чіткий та послідовний. Усі розділи роботи послідовні та логічні, що дозволяє чітко розуміти викладений матеріал в рамках тематики кваліфікаційної роботи. Графічний матеріал дозволяє наочно побачити доцільність та ефективність рішень, які були прийняті за основу для досягнення поставленої мети.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Враховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінку «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи) доцент кафедри АКІТР ХНУ Федуна Микола Васильович

«13» червня 2024р. (підпис)

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Шаварського Назара Віталійовича
ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи КІ2с-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

3.06.2024
дата

підпис

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Проміжне програмне забезпечення розподілених систем зі спеціалізованим функціоналом

Автор: Шаварський Назар Віталійович

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Медзатий Дмитро Миколайович, канд. техн. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) більшість запозичень є джерела з інтернету, кількість запозичень становить 17.9%;
4) запозичення з друкованих ресурсів та ресурсів бібліотеки становить 3.87%;
5) усі зафіксовані системою ознаки модифікації тексту стосуються комбінування латинських символів з україномовними скороченнями, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 17,6% і адресується до 508 першоджерела, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Д. М. Медзатий

Гарант ОП _____ С.М. Лисенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко