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

# КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Галузь знань 12 – Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

на тему «Метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних»

КвРКІ. 190149.22.02.39 ПЗ

Виконала: студентка 2 курсу, група КІ2м-22-2 Підпис Вячеслав КАРПОВИЧ Ім'я, прізвище

Керівник канд. техн. наук, доцент Науковий ступінь, вчене звання Підпис Катерина БЕРЕЗЬКА Ім'я, прізвище

До захисту допускаю:
Зав. кафедри КІІС, д.т.н., проф.
Тетяна ГОВОРУЩЕНКО
07 05 2024 р.

Хмельницький, 2024

# ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень МАГІСТР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма ОСВІТНЬО-НАУКОВА ПРОГРАМА «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ
Зав. кафедри КІІС
Тетяна ГОВОРУЩЕНКО
"01" 09 2023 р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Карпович Вячеслав Васильович
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних
Керівник проекту (роботи) к.т.н., доцент Березька К.М.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.01.2024р. № 1

2. Строк подання студентом проекту (роботи) на кафедру 03.05.2024 р.

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

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

аналіз відомих методів забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних;

архітектура застосунків з інтенсивним обсягом даних;

метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

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

6. Консультанти розділів дипломного проекту (роботи)

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормоконтроль | Лисенко С.М., професор кафедри КІІС | | |
| Антиплагіат | Нічепорук А.О., доцент кафедри КІІС | | |

7. Дата видачі завдання «_06_»_____09_____2023 р.

## КАЛЕНДАРНИЙ ПЛАН

| №з/п | Назва етапів (розділів) дипломного проекту (роботи) | Термін виконання етапів проекту (роботи) | Примітка |
|---|---|---|---|
| 1 | Вибір напряму дослідження та узгодження тематики ДРМ з керівником | 01.09.2022 | виконано |
| 2 | Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження | 05.12.2023 | виконано |
| 3 | Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі | 25.02.2024 | виконано |
| 4 | Робота над розділом 2 – розробка архітектури для вирішення поставленої задачі | 01.04.2024 | виконано |
| 5 | Робота над тезами | 05.03.2024 | виконано |
| 6 | Робота над розділом 3 – розробка методу для вирішення поставленої задачі | 15.04.2024 | виконано |
| 7 | Робота над розділом 4 – проектування та розробка засобів для вирішення поставленої задачі, експериментальна частина | 25.04.2024 | виконано |
| 8 | Оформлення пояснювальної записки згідно вимог | 30.04.2024 | виконано |
| 9 | Попередній захист ВКР | 01.05.2024 | виконано |
| 10 | Захист ВКР на засіданні ЕК | До 30.05.2024 | |

**Студент** _____ Вячеслав КАРПОВИЧ
Підпис Ініціали, прізвище

**Керівник роботи** _____ Катерина БЕРЕЗЬКА
Підпис Ініціали, прізвище

# РЕФЕРАТ

Тема кваліфікаційної роботи: «Метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних»

Автор роботи: Карпович Вячеслав Васильович

Керівник роботи: Березька К. М.

Пояснювальна записка: 97 с., 1 рисунок, 2 таблиці, 81 джерело.

ПЕРЕЛІК КЛЮЧОВИХ СЛІВ: диск; пропускна здатність; алгоритм; комп’ютерні системи.

Об’єктом дослідження є процес забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

Предметом дослідження є методи забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

Метою кваліфікаційої роботи є розробка методу забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

Для розв’язання поставлених задач використовувалися методи теорії комп’ютерних системи, теорії алгоритмів.

Наукова новизна отриманих результатів:

- розроблено новий метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

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

У першому розділі проведено аналіз відомих рішень щодо забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

У другому розділі здійснено дослідження предметної області та визначено автономні рішення на основі індексування для вирішення двох проблем «великих даних» у застосунках з інтенсивним використанням даних: приписування відсутніх значень; запит на просторове об'єднання. Розроблено два різні підходи, засновані

на растрових зображеннях, для приписування відсутніх значень у великих наборах даних. Використано растрові зображення для покращення масштабованості та поганих проблем балансування навантаження в існуючих алгоритмах просторового з'єднання. Також, розглянуто проблеми відмовостійкості в існуючих алгоритмах машинного навчання та інтелектуального аналізу даних. Розроблено алгоритм відмовостійкості на основі пам'яті, який порівняно з дисковими підходами, має особливості, які забезпечують методи контрольної точки та відновлення даних.

У третьому розділі здійснено розроблення методу для побудови системи відмовостійкості для підтримки алгоритму в паралельних системах. Запропоновано три алгоритми відмовостійкості: відмовостійкість на основі диску, відмовостійкість на основі синхронної пам'яті, асинхронна відмовостійкість на основі пам'яті.

У четвертому розділі використано набір алгоритмів інтелектуального аналізу даних і машинного навчання, які були реалізовані за допомогою бібліотеки інтерфейсу передачі повідомлень MPI. В результаті отримані розширені функції цільових алгоритмів та односторонній зв'язок MPI для створення швидких та ефективних відмовостійких систем для кількох алгоритмів. Для цього використовували неблокувальні функції MPI, щоб приховати додаткові витрати на зв'язок «точка-точка», необхідний для контрольно-пропускних пунктів і операцій відновлення.

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

Практична значимість отриманих результатів полягає у розроблених алгоритмах для забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

**ЗМІСТ**

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

SVM – метод опорних векторів

ОС - операційна система

ПЗ - програмне забезпечення

# ВСТУП

Розроблені застосунки з великим обсягом даних для вирішення різних проблем і обмежень, пов'язаних з поточним вибухом даних, використовуються для зберігання, аналізу та управління існуючими великими наборами даних. Одним з основних недоліків існуючих застосунків з великим обсягом даних є відносно повільний доступ до диска. Програми обробки на основі дисків не можуть запропонувати відповідний час відгуку через затримку доступу до дисків. Таким чином, ефективні застосування з інтенсивним використанням даних повинні підтримуватися обробкою на основі пам'яті, а не дисковою обробкою, щоб отримати більшу продуктивність від цих програм.

Метою роботи є покращення довготривалих програм, які потребують великих обсягів даних, шляхом розробки нового методу щодо пам'яті, а також вирішення нових проблем, пов'язаних із програмами обробки на основі пам'яті, таких як відмовостійкість і послідовність. Також, розглядатимемо обмеження відмовостійкості, щоб дозволити різним програмам, які потребують великого обсягу даних, продовжувати належну роботу в разі збою.

Незважаючи на те, що доступ до даних пам'яті є надзвичайно швидким у порівнянні з доступом до диску, наявність ефективної техніки індексування для підтримки швидких запитів допомагає уникнути інтенсивного сканування пам'яті. Використаємо ефективну техніку індексації, яка називається растровим індексуванням, щоб покращити дві складні операції, пов'язані з великими наборами даних: вилучення відсутніх значень; запит на просторове об'єднання Цього можна досягнути шляхом побудови підходів на основі пам'яті з підтримкою індексування, які можуть обробляти великі обсяги даних. Пріритетним напрямом є розроблення способу прискорення процесу приписування відсутніх значень за допомогою растрового індексування. Існуючі способи обробки відсутніх значень не можуть масштабуватися до більших наборів даних. Іншими словами, ця конкретна проблема достовірності, яка

стосується точності даних, була вирішена, але не в контексті вирішення проблеми обсягу «великих даних» та уникнення затримки диска. Техніка індексування растрових зображень використовується для прямого доступу до необхідних записів для методу вилучення. Також, техніка растрового індексування використовується для оцінки відсутніх значень за допомогою попередньо згенерованих векторів індексації растрових зображень без доступу до самого набору даних. Обидва підходи можуть бути оцінені з використанням різних реальних і штучних наборів даних, а також чотирьох загальних алгоритмів вилучення. Крім того, затримка дискового вводу/виводу негативно впливає на запити в просторовому середовищі. Великі просторові набори даних страждають від низької продуктивності та проблеми дисбалансу навантаження у випадку складних операцій запиту, таких як операція просторового об'єднання.

Таким чином, пропонується ефективний підхід до балансування навантаження операції паралельного просторового з'єднання у великих просторових наборах даних, який залежить від використання растрових зображень як зведеної структури просторових даних. Використовуємо растрові зображення для прискорення послідовної обробки в пам'яті. Конкретними алгоритмами, які є такі, де основна ідея полягає в ефективному балансуванні навантаження за допомогою растрових зображень, які використовує растрові зображення для ефективного кроку фільтрації просторового з'єднання.

Також, розглянемо обмеження відмовостійкості існуючих застосунків з великим обсягом даних, забезпечуючи відмовостійку підтримку на основі пам'яті для різних алгоритмів.

Тому, метою роботи є розробка методу забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

Актуальність роботи полягає в необхідності розробити метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

Поставлена мета досягається розв’язанням таких основних завдань:

- проаналізувати відомі методи забезпечення пропускної здатності дисків для застосунків;

- розробити новий метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних;

- здійснити реалізацію розробленого методу забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних;

- здійснити есперименталні дослідження згідно розроблених рішень.

Об’єктом дослідження є процес забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

Предметом дослідження є методи забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

Для розв’язання поставлених задач використовувалися методи теорії комп’ютерних систем, теорії алгоритмів.

Наукова новизна отриманих результатів:

- розроблено новий метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

Практична значимість отриманих результатів полягає у розроблених алгоритмах для забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

За темою кваліфікаційної роботи опубліковано одну публікацію [81] у Збірнику наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2023». (Хмельницький – 2023. – С.116-117).

# 1 АНАЛІЗ МЕТОДІВ І ТЕХНОЛОГІЙ ЗАБЕЗПЕЧЕННЯ ПРОПУСКНОЇ ЗДАТНОСТІ ДИСКІВ ДЛЯ ЗАСТОСУНКІВ З ІНТЕНСИВНИМ ОБСЯГОМ ДАНИХ

## 1.1 Аналіз предметної області

Концепція «великих даних» [1] виникла у зв'язку з нещодавнім вибухом обсягів даних та розбіжностями, пов'язаними з аналізом, управлінням та обробкою великих обсягів даних. Паралельні застосунки з інтенсивним використанням даних були спроектовані та розроблені в різних областях для роботи з великими обсягами даних, які існують в даний час. Однак ці програми мають ряд недоліків і обмежень [2,3], які негативно впливають на їх функціональність. Перспективними для дослідження є дві проблеми [4, 5] паралельних застосунків з інтенсивним використанням даних: повільний час читання диска в порівнянні з продуктивністю сучасних процесорів; більш висока частота відмов при великій кількості вузлів і тривалому часу виконання. При цьому ця робота спрямована на забезпечення двох основних вимог [6]: розробку нових методів усунення вузького місця дискового вводу/виводу; розробку ефективних відмовостійких алгоритмів для обробки апаратних несправностей у паралельних застосунках.

Розглянемо спочатку значення та мотивацію обробки на основі пам'яті та підтримки відмовостійкості в паралельних застосунках, що вимагають великого обсягу даних [6, 7]. Далі розглянемо дві основні концепції: індексування растрових зображень [8]; апаратні несправності [9]. А також здійсимо огляд робіт, включаючи припущення про відсутніх значень на основі растрових зображень, операцію просторового з'єднання на основі растрових зображень і відмовостійкі системи для генератору шаблонів [10-12].

Розглянемо завдання паралельних обчислень, що вимагають великих обсягів даних. Швидке зростання обсягів даних, що надходять з різних джерел, таких як соціальні медіа, ділові та наукові застосунки, а також публічна мережа,

підвищує потребу в розробці надійних застосунків для обробки цих великих обсягів даних. З цієї причини, а також з наявністю швидких і потужних обчислювальних ресурсів, починають з'являтися паралельні застосунки з інтенсивним використанням даних [12]. Останніми роками темпи зростання даних стають швидшими [13]. Наприклад, світова оцінка темпів зростання бізнес-даних у комерційних застосунках показує, що бізнес-дані подвоюються кожні 1,2 року [13, 14]. Незважаючи на те, що великий обсяг даних може бути джерелом важливої інформації, нова парадигма даних породжує кілька проблем і обмежень. Ці проблеми та обмеження повинні бути вирішені за допомогою інноваційних та новітніх методів, щоб довести продуктивність нових обчислювально дорогих застосунків. Існуючі дискові системи обробки більше не можуть забезпечити належний час відгуку через надмірну затримку жорстких дисків [15]. Значна залежність від диска для обробки та керування великими наборами даних обмежує функціональність та можливості існуючих застосунків, які потребують великих обсягів даних. Таким чином, ці програми повинні підтримуватися службами аналізу та управління даними з низькою затримкою та в режимі реального часу [16]. Позбувшись вузького місця дискового вводу/виводу та більше покладаючись на пам'ять, можна використовувати паралельні системи та підвищити загальну продуктивність, зберігаючи той самий рівень точності [17]. Ємність пам'яті подвоюється кожні три роки [18], а її вартість знижується на 10 відсотків кожні п'ять років [19]. Таким чином, методи обробки на основі пам'яті можуть бути ефективним рішенням у випадку великих наборів даних. Було проведено роботу з розробки нових методів передачі більшої кількості пам'яті для обробки великих наборів даних [12-18]. Однак розробка таких методів повинна бути більш точною, оскільки системи обробки в пам'яті можуть постраждати від інших джерел додаткових витрат [19, 20], які приховані нормальною затримкою вводу/виводу диска у випадку дискової обробки. Загалом, дослідження систем управління та обробки даних у пам'яті концентруються на кількох аспектах, таких як

використання індексування для покращення доступу до даних, покращення компонування даних, використання паралелізму, прискорення потоку даних, побудова відмовостійкої системи, покращення обробки запитів у пам'яті [19, 20].

Розглядатимемо паралелізм, індексацію та відмовостійкість для покращення обробки в пам'яті з великими наборами даних. Використовуватимемо особливості існуючої структури індексування, яка називається растровим індексуванням, щоб розробити нові методи для підтримки двох різних операцій у великих наборах даних: приписування відсутніх значень і запит на просторове об'єднання. Запропоновані методи спрямовані на використання пам'яті, а не диску, для підвищення загальної продуктивності обох операцій з великими обсягами даних. Також, розглядатимемо в цьому контексті проблему відмовостійкості в системах, пропонуючи різні рішення для існуючих застосунків, що вимагають великого обсягу даних. Щоб показати вплив апаратних збоїв в системах, розробники апаратного забезпечення оцінюють частоту апаратних відмов існуючих систем, проводячи серію експериментів [21]. У цих експериментах були використані великомасштабні довготривалі симуляції. Таким чином, системні збої повинні бути усунені для підвищення загальної продуктивності паралельного виконання шляхом створення відповідних відмовостійких паралельних застосунків. Довготривалі програми, що вимагають великого обсягу даних, зазвичай є програмами інтерфейсу передачі повідомлень [22].

Таким чином, будемо розглядати проблему відмовостійкості в застосунках на основі MPI. Для цієї стратегії застосовуватимемо алгоритми машинного навчання та інтелектуального аналізу даних. Застосунки MPI з інтенсивним використанням даних були підписані та реалізовані з механізмом контрольних точок/відновлення [7, 23, 25, 40, 42, 70]. Дисковий механізм контрольної точки/відновлення має бути простішим у проектуванні та реалізації. Однак головною метою є усунення місця дискового вводу/виводу, покладаючись на пам'ять, а не на диск, і уникаючи великих додаткових витрат.

Здійснимо огляд двох основних концепцій: індексування растрових зображень і апаратних несправностей в паралельних застосунках. Індексування широко досліджувалося і використовувалося в контексті реляційних баз даних, де ефективний доступ до даних може бути забезпечений шляхом розробки ефективної схеми індексації. Однак більшість існуючих методів індексації вимагають реорганізації записів даних, що створює складні проблеми, особливо у випадку великих наборів даних. Ще одним обмеженням цих схем індексації є те, що вони не здатні обробляти динамічні, зашумлені та неоднорідні дані, пов'язані з великомасштабними наборами даних. Крім того, ці схеми індексації не можуть ефективно підтримувати запити з використанням великої кількості атрибутів, що буде очікуваним випадком використання, коли це великі обсяги даних. Вимоги до застосунків, наприклад, підтримка гетерогенних даних, потокових даних, а також підтримка хмарних технологій можуть бути легко задоволені за допомогою схеми індексації на основі бітових векторів [24-26] У цьому типі індексації двійкові значення можуть бути використані для представлення записів даних у стисненому форматі, а двійкові операції можуть бути використані для виконання легких побітових логічних операцій над новим представленням даних на основі бітових векторів.

В основному, дослідницька робота з індексування растрових зображень була зосереджена на розробці ефективної обробки запитів для різних великих комерційних і наукових наборів даних і розробці ефективних алгоритмів стиснення растрових зображень для зменшення займаного простору растровими векторами [27]. Незважаючи на те, що індексування растрових зображень сьогодні вважається перспективним напрямом досліджень, все ще залишаються деякі дослідницькі проблеми, такі як: розробка растрової структури для обробки відсутніх значень у великих наборах даних і розробка систем запитів на основі растрових зображень для складних операцій запитів, таких як операція об'єднання. Більш детально про індексування растрових зображень, таких як: структура бітових векторів, методи стиснення і підтримувана система запитів

подано в роботі [28].

Існуючі суперкомп'ютерні системи відіграють важливу роль у розробці масових застосунків для об'єму даних. Ці системи виросли від петамасштабних обчислень до екзамасштабних обчислень, щоб задовольнити потреби поточних і майбутніх застосунків з інтенсивним використанням даних. Однак високий рівень відмов цих систем розглядається як одна з основних перешкод для досягнення екзамасштабу [28]. Тому, будемо розглядати апаратни збої як ще однин ризик паралельних застосунків, що вимагають великого обсягу даних. Масивні масштабні системи страждають від численних типів несправностей: постійних, перехідних і переривчастих. Постійна несправність зазвичай вимагає заміни пристрою. Будемо розглядати відмовостійкість до незворотних технологічних дефектів. У багатьох випадках несправності в одному або декількох системних пристроях, таких як блок живлення, вентилятори, диски, проявляються як постійні технологічні несправності. Як тільки процес сприймається як несправний, то його вважатимемо неіснуючим для решти обчислень.

Оскільки постійні несправності вузлів є звичайним явищем у великомасштабних системах, кілька дослідників запропонували методи усунення цих несправностей [29]. Як правило, використовуються методології, засновані на контрольній точці-перезапуску [30]. Методи, незалежні від застосування, перевіряють простір застосування шини на постійному диску, однак, було показано, що вони масштабуються лише на системах малих розмірів [31]. Методи, що залежать від алгоритмів відомі як методи відмовостійкості на основі алгоритмів, що зменшують ці додаткові витрати, періодично вибірково перевіряючи важливі структури даних. Однак, залежно від характеристик застосування, контрольні точки критичних структур даних все одно можуть вимагати доступу до диска.

Розглянемо стратегію прискорення інтелектуального аналізу даних на неповних наборах даних шляхом припущення відсутніх значень на основі

растрових зображень. Розглянемо, відповідно, спочатку стратегії вилучення відсутніх значень у великих наборах даних. Як відомо, якість оброблюваних даних є життєво важливою для гарантії всіх алгоритмів інтелектуального аналізу даних. Проблема відсутніх значень є одним із поширених факторів, що впливають на упередженість процесу аналізу даних. Щоб вирішити проблему вилучення відсутніх значень у масивних наборах даних, пропонується структура, заснована на растровому індексі, яка може допомогти прискорити техніку вилучення, а також скоротити час читання диска. Різні алгоритми вилучення відсутніх значень, наприклад, метод глобальної відповідності, найбільш поширене значення, найбільш поширене значення та середнє значення атрибута можуть бути прискорені за допомогою техніки растрового індексування.

Новий метод може базуватись на стратегії застосування растрового індексування для пошуку відповідних записів, і, таким чином, дасть змогу уникати повного сканування всього набору даних, який називають методом вилучення прямого доступу. Цей метод, заснований на пам'яті, щоб використовувати растрове індексування як підсумок усього набору даних і взагалі не звертатися до набору даних, який отримується методом растрового вилучення.

Балансування навантаження та прискорення операцій паралельного просторового з'єднання за допомогою растрового індексування необхідно, щоб охопити великі просторові набори даних. Пропонується здійснити масштабованість загальної та складної операції просторового запиту, тобто операції просторового з'єднання. Зокрема, цей тип запиту спрямований на пошук пар просторових об'єктів із двох заданих просторових наборів даних, які задовольняють певному просторовому співвідношенню такі як перетин тощо [32]. Використовуючи переваги індексування растрових зображень, такі як швидкі побітові логічні операції та стиснене представлення, операція просторового об'єднання може бути прискорена за допомогою растрових

зображень. Растрове індексування використовується двома різними способами. Растрові зображення використовуються для ефективного розбиття просторового простору, щоб мати збалансоване робоче навантаження у випадку паралельної обробки. Крім того, растрові зображення використовуються для представлення просторових об'єктів під час операції об'єднання. Растрові зображення прискорюють операцію об'єднання, групуючи набір значень точок об'єктів в одне значення, щоб зменшити загальну кількість порівнянь в операціях з'єднання. Використання растрових зображень для покращення процесу розбиття масивних просторових наборів даних є новим підходом, який називається підходом просторового з'єднання на основі розбиття растрових зображень. Просторові об'єкти розподіляються порівну на різні розділи динамічного балансування навантаження для підвищення загальної продуктивності паралельної обробки. На основі растрових зображень у пам'яті можна прискорити операцію просторового з'єднання в пам'яті шляхом зменшення загальної кількості необхідних порівнянь та шляхом групування набору значень точок об'єктів в одне значення з використанням різних значень точності.

Розглянемо відмовостійкий алгоритм зростання з використанням розширених функцій MPI. Розглянемо спочатку різні способи боротьби з апаратними збоями в паралельних застосунках, що вимагають великого обсягу даних. В останньому десятилітті розробляються системи екстремального масштабу для обчислювально дорогих застосувань [33, 34]. Загальним конструктивним обмеженням цих систем є підвищена частота відмов апаратних компонентів [35, 36]. Було проведено багато робіт з усунення обмеження відмов апаратного забезпечення шляхом розробки відмовостійких алгоритмів, моделей програмування та систем виконання [36, 37]. Розглядаємо проблему відмовостійкості в застосунках інтерфейсу передачі повідомлень з різними алгоритмами. Механізм відмовостійкості на основі алгоритмів для кожного алгоритму визначає критичну структуру даних, яку слід зберегти на випадок

будь-якого майбутнього збою, який може статися. Ця структура даних періодично зберігається на безпечному просторі зберігання. Більші структури даних споживають більше переміщення даних для контрольних точок, що стає фактором обмеження.

Суть пропонованого алгоритму полягає у генератору патернів, який потребує двох проходів над даними для досягнення цільових результатів [38]. Дерево визначаємо як критичну структуру даних. Використовуємо переваги як алгоритму, так і одностороннього зв'язку MPI [18] для виконання повних операцій контрольної точки та відновлення цієї структури даних з набагато меншими витратами на виконання. Існуючі моделі програмування і алгоритми реалізовуватимемо за допомогою MPI. Зокрема, використовуватимемо механізм MPI для асинхронної перевірки критичних структур даних у пам'яті. В результаті алгоритм буде характеризуватись контрольною точкою у пам'яті для великомасштабних систем. Запропонований алгоритм використовує перекриття зв'язку з фазою побудови дерева таким чином, що додаткові витрати на контрольні точки зводяться до мінімуму.

Також, потребують аналізу та детального розгляду три різні механізми відмовостійкості для паралельного алгоритму зростання: відмовостійке зростання на основі диска за замовчуванням; відмовостійке зростання на основі синхронної пам'яті; відмовостійке зростання на основі асинхронної пам'яті [39].

Розглянемо вплив апаратних збоїв на поширену класифікацію. У алгоритмі KNN основна мета полягає в тому, щоб знайти K найближчих вибірок, щоб предикатувати правильний клас певної вибірки в даному наборі даних. Використовуємо ту саму стратегію для побудови відмовостійкого алгоритму KNN. Як відомо, визначення критичної структури даних є важливим для побудови надійних алгоритмів перевірки та відновлення. У паралельних системах набір даних розділений на набір фрагментів, де кожен процес володіє одним фрагментом. Кожен процес спрямований на пошук K найближчих зразків для кожної локальної вибірки, яку він має. Таким чином, кожен процес повинен

обчислити відстань між кожною локальною вибіркою і всіма зразками, розподіленими різними порціями. Для кожної локальної вибірки найближчі зразки K зберігаються в структурі даних пріоритетної черги, яка оновлюється при знаходженні нової ближчої вибірки. Доступ до зразків інших процесів може здійснюватися за допомогою набору операцій зв'язку між різними вузлами. Кожен процес зберігає у своїй пам'яті набір пріоритетних черг для всіх локальних вибірок. Щоб побудувати надійний відмовостійкий алгоритм KNN, потрібно зберігати структуру даних у надійному місці, щоб її можна було відновити у разі збою. У запропонованих підходах для кожного процесу використовуємо пам'ять іншого процесу для перевірки точок. Пропонується два різні алгоритми відновлення, щоб показати, як різні варіанти відновлення можуть вплинути на загальну продуктивність відмовостійкого алгоритму KNN. Крім того, використовуємо технологію одностороннього зв'язку MPI, щоб максимально перекрити операції з контрольно-пропускного пункту та відновлення. Зокрема, застосуємо односторонній комунікаційний підхід MPI до черг пріоритету контрольних точок.

Для відновлення будемо використовувати два різні алгоритми. В першому алгоритмі для відновлення та обробки зразків процесу, що зазнав невдачі, використовується лише один процес. Для порівняння, алгоритм використовує всі активні процеси, що залишилися, для обробки робочого навантаження невдалого процесу. Для здійснення поглибленої оцінки запропонованих алгоритмів контрольних точок і відновлення використаємо великі набори даних, і порівняємо їх з підходом на основі двосторонньої комунікації MPI.

Розглянемо можливість застосування в пропонованій стратегії відмовостійких опорних векторних машин (SVM). Тут опрацьовуємо апаратні збої за допомогою алгоритму машинного навчання SVM. Алгоритм SVM — це алгоритм контрольованого навчання, який будує математичну модель, яка максимізує межу поділу між позитивними та негативними вибірками у бінарній навчальній множині. Розглянемо підхід послідовної мінімальної оптимізації,

який зазвичай використовується для реалізації алгоритму SVM [40]. При невеликому розмірі наборів даних виконання алгоритму з високопродуктивними системами, як відомо, займає кілька годин [41], що збільшує середній час відмови.

Таким чином, проведено аналіз предметної області для дослідження. В результаті встановлено, що досягнення відмовостійкості потребує удосконалення за рахунок методів інтелектуального аналізу даних для застосунків з великим обсягом даних.

## 1.2 Аналіз методів вилучення пропущених значень на основі пам'яті

Розглянемо розробку методів на основі пам'яті для операції вилучення пропущених значень. Будемо використовувати техніку індексування растрових зображень для прискорення операції вилучення у великих наборах даних, щоб отримати більшу продуктивність і зменшити загальні додаткові витрати. Зокрема, зазначимо, що існуючі методи обробки відсутніх значень не можуть масштабуватися до більших наборів даних. Іншими словами, ця конкретна проблема достовірності, тобто точності даних, була вирішена, але не в контексті також проблеми обсягу і, можливо, швидкості, великих даних. Розглянемо вирішення проблеми відсутніх значень за допомогою методу індексування растрових зображень. Растрове індексування використовується для прямого доступу до необхідних записів для методу вилучення. Техніка растрового індексування використовується для оцінки відсутніх значень за допомогою попередньо згенерованих векторів індексації растрових зображень без доступу до самого набору даних. Обидва рішення були оцінені з використанням різних реальних і штучних наборів даних, а також чотирьох загальних алгоритмів вилучення. Методи, засновані на растрових зображеннях, можуть прискорити інтелектуальний аналіз даних, класифікацію неповних даних, зберігаючи при цьому точність [41-44].

Одне з найпопулярніших визначень великих даних включає в себе чотири складні аспекти роботи з даними: обсяг; швидкість; різноманітність; достовірність. Останніми роками було проведено багато роботи над вирішенням цих проблем, причому найбільший інтерес викликала проблема обсягу і, меншою мірою, проблема швидкості [41-43]. Робота над фреймворком для обробки масивних даних, а також робота з обробки потокових даних та аналізу даних на місці були темами багатьох досліджень [41-44]..

Проблема достовірності, яка стосується точності даних, не отримала стільки уваги. Традиційно добре відомо, що проблеми, пов'язані з якістю даних, такі як неповні, надлишкові, неузгоджені та зашумлені дані [45], становлять серйозну проблему для інтелектуального аналізу даних та їх аналізу взагалі. Фактично, одним із найважливіших етапів інтелектуального аналізу даних вважається етап підготовки даних, який є процесом забезпечення якості даних шляхом зміни вихідних даних у відповідний формат для процесу аналізу. Більша половина часу виділеного на інтелектуальний аналіз даних витрачається на процес підготовки даних порівняно з фактичним часом у процесі інтелектуального аналізу даних [46].

Власне, точності великих даних не приділялося стільки уваги. Нещодавня робота [46] над фреймворками не наголошувала на складнощах попередньої обробки даних і, зокрема, на складності застосування відомих методів для великомасштабних даних. Розглянемо проблему, пов'язану з обробкою відсутніх значень. Неповний набір даних або відсутні значення можуть вплинути на завдання аналізу даних. Щоб уникнути їх негативного впливу, популярним підходом є заповнення відсутніх значень розрахунковими значеннями, які можна обчислити на основі повних записів того ж набору даних [47]. Кілька досліджень проілюстрували різні методи роботи з відсутніми значеннями, тобто використання повного набору записів для приписування відсутніх значень у реальних наборах даних. Простий алгоритм [48], тобто алгоритм для приписування пропущених значень за допомогою найбільш поширеного

значення одного і того ж атрибута [49]. Модифікована версія цього методу є найбільш поширеним значенням атрибута, обмеженим міткою або концепцією [50]. Інше дослідження [51] запропонувало замінити відсутнє значення всіма можливими значеннями, які з'являються в одному атрибуті у всіх записах набору даних. Також був запропонований інший підхід [52], де відсутнє значення в конкретному записі береться з відповідних значень із запису, найбільш схожого на цей запис у всьому наборі даних. Крім того, у різних дослідженнях було розглянуто та порівняно кілька підходів до відсутніх значень [53-56].

Загальною темою всіх робіт у цій галузі є те, що алгоритми приписування втраченого значення не можуть масштабуватися до більших наборів даних через затримку доступу до дисків. Іншими словами, ця конкретна проблема достовірності була розглянута, але не в контексті проблеми обсягу і, можливо, швидкості великих даних. Ця робота усуває цей недолік існуючої роботи та зосереджується на покращенні масштабованості методів за допомогою методів обробки в пам'яті. Використовуючи індексацію для прискорення приписування відсутніх значень використовуємо растрову індексацію, яка може служити підсумком наборів даних. Ключова перевага цієї методики полягає в тому, що для обробки можна використовувати недорогі побітові операції.

Різні методи вилучення відсутніх значень можуть бути прискорені за допомогою обробки в пам'яті, що підтримується технікою індексування растрових зображень, на двох рівнях. Як і будь-яка техніка індексування, растрове індексування можна використовувати для пошуку відповідних записів, і, таким чином, можна уникнути повного сканування всього набору даних. Можна використовувати растрові зображення як підсумок всього набору даних і взагалі не отримувати доступ до набору даних.

Розглянемо існуючі алгоритми роботи з відсутніми значеннями, тобто конкретно методи вилучення та індексацію на основі растрових зображень.

Мета будь-якого алгоритму вилучення полягає в тому, щоб оцінити відсутнє значення в конкретному записі, використовуючи інші повні записи в

тому ж наборі даних.

Наведемо огляд чотирьох існуючих алгоритмів [57-60].

Глобальний метод найближчої посадки подано в роботах [57-59]. Основна ідея цього методу полягає в тому, щоб замінити відсутнє значення на значення атрибута з одного запису, який є найбільш схожим записом на запис з відсутнім значенням. Найбільш схожим записом є той, що має найменшу відстань від поточного запису, причому відстань розраховується із рівняння, що є загальною формою для обчислення різниці між будь-якими двома записами як підсумовування відстані між значеннями в кожному атрибуті для обох записів. Рівняння може бути використане для обчислення відстані між двома значеннями одного і того ж атрибута в записах з різницею між максимальним і мінімальним значеннями цього атрибута для всього набору даних. Це відповідає використанню того, на що посилається загальновживана метрика Манхеттенська відстань [61].

Метод, який базовано на найбільш поширеному значенні, подано в роботах [57-59]. Це метод, при якому відсутнє значення може бути замінено найпоширенішим значенням в атрибуті. Просканувавши весь набір даних, можемо знайти єдине значення атрибута, яке зустрічається найчастіше [59, 62].

Аналогічно, метод, який базовано на найбільш поширеному значенні на основі концепції, подано в роботах [59-61]. Метод на основі концепції є окремим випадком методу найбільш поширених значень. Цей метод передбачає, що записи мають пов'язані з ними мітки. Використовуючи цю інформацію, відсутнє значення замінюється найбільш поширеним значенням для конкретного атрибута, серед безлічі записів з такою ж міткою [61].

Метод середнього значення атрибута на основі концепції подано в роботах [60-62]. Відсутнє значення атрибута замінюється середнім арифметичним значень того ж атрибута серед записів, що мають ту саму концепцію. Подальшим узагальненням цього методу є метод середніх значень атрибутів, який використовується для приписування відсутнього значення як середнього

значення всіх значень в одному атрибуті, тобто ігнорування будь-яких можливих міток [61, 63].

Метод індексування растрових зображень подано в роботах [63-65]. Реалізація кожного з наведених методів подано в роботах [57-65].передбачає невеликий набір даних, який вміщується в пам'ять і включає просте сканування всіх записів для пошуку значень для заміни відсутніх значень. Ці реалізації явно мають обмеження при роботі з великими наборами даних.

Таким чином, розглянемо використання індексації як механізму прискорення методів вилучення. Є кілька причин для розгляду такого підходу. Індексація вже реалізована в більшості сховищ даних, які обробляють великомасштабні дані. Всі або більшість описаних методів можна прискорити за допомогою підтримки індексації.

Обраним методом індексації є растрове індексування [65]. Розглянемо короткий огляд індексування растрових зображень і причин його вибору спеціально для підвищення продуктивності існуючих технологій приписування відсутніх значень. У найпростішому вигляді генерується один бітовий вектор для кожного окремого значення в кожному атрибуті. Довжина бітового вектора дорівнює кількості записів даних у наборі даних. Якщо значення конкретного атрибута в записі збігається зі значенням, якому відповідає цей бітовий вектор, то біт встановлюється в 1, а в іншому випадку дорівнює 0. Однак велика кількість різних значень атрибута негативно впливає на продуктивність індексування растрових зображень, оскільки для покриття всіх цих різних значень необхідно згенерувати велику кількість бітових векторів. Типовим рішенням цієї проблеми є використання процесу. У процесі значення атрибутів бінуються таким чином, що кожен бін представляє діапазон значень [66]. Бітвектори в первісному вигляді можуть стати надзвичайно компактними.

Таким чином, для вирішення проблеми простору було запропоновано кілька методів стиснення. Основна ідея цих методів полягає в тому, щоб закодувати можливу велику серію суміжних 0 або 1, хоча вони менш ймовірні,

перед їх зберіганням.

Використовуючи індексування растрових зображень, типовий запит на підмножину може бути оброблений шляхом вилучення набору бітових векторів в залежності від умов запиту, а потім результат може бути експортований шляхом виконання побітових операцій І і АБО над цими бітовими векторами, які зазвичай дуже ефективно підтримуються в апаратному забезпеченні [67].

Багато дослідників [68-76] запропонували відмовостійкі алгоритми для наукових застосувань. Однак не відомо про будь-які підходи до відмовостійкості на основі алгоритмів для машин з опорними векторами. Нещодавно запропоновані SVM для уникнення зв'язку [77] не враховують відмовостійкість і не надають гарантій точного рішення. Алгоритми розроблені з використанням моделей функціонального програмування [78]. Моделі функціонального програмування спрощують специфікацію паралелізму, дозволяючи користувачеві задавати карту і зменшувати завдання і записувати кожну мутацію кожного варіанту. Наприклад, у SVM кожне оновлення градієнта буде записуватися та відтворюватися під час відновлення у функціональній моделі програмування. Однак це не сприяє ефективному результату, оскільки додаткові витрати на збереження кожної мутації високі, тоді як запропонований алгоритм уникнення комунікації повністю усуває етап контрольної точки.

Інші відмовостійкі моделі програмування включають контрольні точки [79] для інтерфейсу передачі повідомлень MPI [78]. Крім того, комунікаційні підсистеми враховували відмовостійкість. У той же час як ці моделі програмування та середовища виконання надають інструменти для безперервного виконання під час помилок, все-таки залишається завдання реалізації алгоритму відновлення. Також були запропоновані підходи до автоматичного орпацювання контрольних точок [80]. Однак вони обмежені розміром знімка контрольних точок, і, як відомо, масштабуються лише на невеликих системах.

Таким чином, відмовостійкі моделі потрібно обов’язково застосовувати

для забезпечення автоматичного орпацювання контрольних точок.

## 1.3 Висновки до першого розділу

В результаті проведеного дослідження предметної області було встановлено недоліки відомих рішень і виділено їх з метою розробки рішень, проведено аналіз проблемних завдань.

Застосунки, що вимагають великих обсягів даних, були розроблені для вирішення ряду обмежень і проблем, пов'язаних з існуючими великими обсягами даних. Тому важливим питанням стало удосконалення та розширення можливостей цих програм. Було розглянуто два обмеження існуючих застосунків з інтенсивним використанням даних: уповільнення доступу до диска; апаратні збої. Додаткові витрати на затримку диску в існуючих програмах обробки дисків обмежують його функціональність, особливо у випадку великих наборів даних. Тому, пам'ять стає більш підходящою альтернативою для прискорення обробки та управління великими наборами даних. Однак великі набори даних не можуть бути безпосередньо оброблені та підтягнуті в пам'ять одним обсягом. Таким чином, програми обробки на основі пам'яті повинні бути підкріплені методами управління операціями введення-виведення пам'яті. Одним із перспективних напрямів розв’язання такого завдання є використання індексування для швидкого доступу до окремих записів.

## 1.4. Постановка задачі дослідження

Поставлена мета досягається розв’язанням таких основних завдань:

- проаналізувати відомі методи забезпечення пропускної здатності дисків для застосунків;

- розробити новий метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних;

- здійснити реалізацію розробленого методу забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних;

- здійснити есперименталні дослідження згідно розроблених рішень.

# 2 МОДЕЛЬ ПРИСКОРЕННЯ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ НА НЕПОВНИХ НАБОРАХ ДАНИХ

## 2.1 Дослідження предметної області

Прискорення інтелектуального аналізу даних на неповних наборах даних будемо розглядати в контексті припущення відсутніх значень на основі растрових зображень. Існує кілька причин вибору растрового зображення для покращення процесу видалення. Растрове зображення розглядає кожен атрибут окремо, і, таким чином, швидше за все буде підтримувати ефективний індекс для неповних наборів даних. Для порівняння, інші популярні методи індексації, такі як B-дерево, B+-дерево та R-дерево призначені для підтримки запитів за одним або двома атрибутами. Растрові індекси можуть бути побудовані поверх існуючого сховища даних, тобто вони не вимагають реорганізації даних під індекс. Така реорганізація даних займає надзвичайно багато часу і може бути невиправданою, якщо єдиною або головною метою індексації може бути підтримка приписування відсутніх значень. Операція видалення відсутніх значень є приблизним процесом.

Таким чином, алгоритми видалення відсутніх значень на основі растрових зображень можуть бути прискорені за допомогою методів апроксимації растрових зображень, таких як бінування без втрати точності. Під час інтелектуального аналізу даних напрями роботи орієнтовані на набори даних, які не оновлюються, і растрові зображення добре підходять для таких наборів даних.

Щоб прискорити різні алгоритми видалення за допомогою растрових індексів є два різних способи, за допомогою яких растрові зображення можуть бути використані для процесу видалення, а саме: з компромісом між точністю; швидкістю обробки. Пошук записів, схожих на запис із відсутнім значенням або записи з певною міткою можна прискорити за допомогою растрових зображень. Таким чином, сканування всього набору даних може бути замінено переглядом конкретних частин набору даних. Можна використовувати бітові вектори як

приблизний підсумок всього набору даних і давати приблизні відповіді, використовуючи цю інформацію. Першу стратегію забезчує метод видалення прямого доступу, тоді як другу стратегію метод точкового видалення. Перш ніж розглядати підходи, відзначимо, що растрові зображення використовувалися для підтримки обробки запитів в декількох системах. Таким чином, якщо растрові зображення вже створюються для підтримки обробки запитів, реалізація методу видалення з їх використанням навіть не вимагатиме додаткових витрат на генерацію індексів.

Кілька алгоритмів видалення можна прискорити, якщо зможемо безпосередньо отримати доступ до необхідних записів для введення кожного відсутнього значення без повного сканування бази даних. У цьому полягає ідея підходу. Розглянемо, як застосовувати метод у випадку трьох алгоритмів видалення: глобальної найближчої відповідності; концептуального найбільш поширеного значення атрибута; концептуального середнього значення атрибута.

Розглянемо основний метод найближчого сусіда. Він приписує відсутнє значення з записів, які знаходяться найближче до запису з відсутнім значенням, в контексті відстані між двома записами. Метод перебору для реалізації цього методу передбачає отримання відсутнього запису та обчислення відстані між цим записом та всіма іншими записами, тобто метод повного сканування набору даних. Використовуючи растрові зображення, можемо отримати лише той набір записів, які мають значення атрибутів, близькі до цільових значень записів.

Виконуючи повне сканування набору даних, відносно легко вибрати найближчий збіг щодо інших атрибутів. Однак використання методу індексації для пошуку найближчого збігу вимагає певних труднощів. Потрібно продовжувати визначати сусідні значення кожного атрибута, щоб знайти найближчий збіг або добре наближення. Для того, щоб уможливити процес, використовуємо поняття впливу сусіднього значення атрибута на відстань. Вплив наступного значення для кожного атрибута можна обчислити як поточне значення, що використовується для атрибута, що враховує різницю між

максимальним і мінімальним значенням атрибута. Ідея полягає в тому, що сусідні значення кожного атрибута можуть по-різному впливати на відстань. Записи, які мають сусіднє значення з найменшим впливом, потрібно шукати в першу чергу, оскільки вони матимуть меншу відстань. Обчислення представляє набір записів з будь-якого набору даних, який має відсутнє значення. Припустімо, що немає записів, які мають таке саме значення для всіх інших атрибутів, як запис із відсутнім значенням. Другий процес пошуку здійснюється шляхом вибору наступного найближчого значення, який записаного у файлі метаданих, для кожного атрибута та вибору атрибута і його значення з найменшим впливом. Наприклад, він має найменший вплив. Далі потрібно отримати записи, де відповідне значення менше або дорівнює цьому конкретному значенню для обраного атрибута і більше, ніж старе значення, при цьому має те саме значення для решти атрибутів, які були використані для попереднього пошуку. Якщо відповідних записів не виявлено, то процес вибору нового атрибута з використанням значення впливу повторюється. Після того, як результуючий бітвектор отримує набір записів, наприклад, ці записи повинні бути перевірені з метою отримання найбільш близького запису до запису з відсутнім значенням. Реалізація такого підходу відбуватиметься в контексті конкретної системи управління даними на основі растрових зображень. Ця система підтримує підмножину запитів, подібну до бази даних над даними в наукових форматах даних. Модуль обробки запитів перетворює запит користувача в набір умов запиту, де кожна умова може бути представлена одним бітовим вектором, тобто умовою bitvector.

Завдання полягає в тому, як побудувати запит, який отримує записи, які є найбільш близькими до запису, з відсутнім значенням на кожному кроці. Наприклад, потрібно отримати всі записи, які точно збігаються із записом з відсутнім значенням. У цьому випадку кожна окрема умова запиту на атрибут буде побудована таким чином, щоб охопити всі записи в діапазоні, де максимальне та мінімальне значення дорівнюють відповідному значенню

атрибута в записі про відсутнє значення. Далі, потрібно збільшити внутрішній діапазон, щоб покрити записи, які можуть бути використані в процесі видалення. Отже, запит повинен бути побудований так, щоб мати максимальне і мінімальне значення, пов'язані з ним. Алгоритм операції побудови запитів формально описується і приймає на вхід запис з відсутнім значенням. Різні значення для кожного атрибута зчитуються з файлу метаданих у вектор. Він ініціює масив запиту зі значеннями відсутніх значень запису. За допомогою алгоритму для кожного атрибута всі записи, які мають значення між значеннями як значення запиту, так і значення запису, повинні бути отримані з нього. Ідея полягає в тому, щоб адаптувати значення масиву запиту до тих пір, поки результуючий вектор не отримає будь-яку кількість записів. Для окремого значення кожного атрибута обчислюється вплив зміни старого значення у значеннях запиту з цим значенням, щоб визначити значення кожного атрибута, який зазнає найменших змін у значенні впливу. Масив використовується для визначення атрибута, який має найменше значення впливу. Значення цього атрибута замінює старе значення в масиві запиту. Це значення видаляється з вектора, щоб гарантувати, що наступного разу буде вибрано нове значення. Запит був сформований з використанням запису відсутнього значення та масиву запиту. Якщо растровий вектор не має записів, потрібна наступна ітерація. Ця ж стратегія може бути застосована, якщо використовується процес. Операція зміни власного значення атрибута в залежності від ступеня впливу на відстань може бути застосована до об'єднаних значень, вибравши відповідний діапазон значень, а не одне значення. У цьому випадку ступінь впливу можна обчислити, використовуючи середнє значення поточного діапазону і середнє значення діапазону кандидатів.

Розглянемо метод найбільш поширених значень на основі концепцій. Метод може бути реалізований за допомогою растрових зображень наступним чином. Після отримання всіх відсутніх значень для кожного запису, який містить відсутнє значення, будується один запит запиту. Мета запиту полягатиме в тому, щоб отримати всі записи, які містять ту саму мітку, що й запис. Потім ці

отримані записи можна перевірити, щоб визначити найпоширеніше значення атрибута, де запис має відсутнє значення. Ці операції тривіально виконувати, якщо не використовується пошук. Однак у разі пошуку для кожного пропущеного значення отриманими записами будуть записи, які містять мітку в тому самому діапазоні, що й мітка, пов'язана із записом. На останньому кроці можемо визначити найпоширеніший контейнер, а потім приступити до пошуку за найпоширенішим значенням. Це може призвести до іншого значення, порівняно зі значенням, отриманим алгоритмом перебору. Це пов'язано з тим, що найпопулярніший масив значень може не містити значення, яке зустрічається найчастіше. Реалізація найбільш поширеного значення також дуже схожа. Тоді, зосереджуємося на знаходженні найбільш поширеного значення атрибута, задіяного у всіх записах.

Розглянемо метод значень середнього атрибута на основі концепції. Подібно до попереднього методу, один запит запиту будується для отримання всіх записів, які мають ту саму мітку, що й запис, який має відсутнє значення. До цих записів можна звертатися безпосередньо, і всі значення атрибута, де він має відсутнє значення, можуть бути отримані в пам'ять. Оціночна вартість це середнє арифметичне всіх отриманих значень. У разі пошуку потрібні певні розрахунки. Файл метаданих атрибуту слід перевірити, щоб визначити об'єднаний діапазон, який містить мітку цільового запису, тобто запис із відсутнім значенням. Потім будується простий запит для отримання всіх записів у цьому бінованому діапазоні. Ці записи перевіряються ще раз, щоб визначити записи з тією ж міткою, що й запис про відсутнє значення. Введене значення буде розглядатися як середнє арифметичне всіх значень відсутнього атрибута.

Розглянемо метод видалення на основі растрових зображень. Цей підхід передбачає введення відсутніх значень безпосередньо за допомогою растрових векторів, тобто без доступу до самого набору даних. Використання підходу в трьох алгоритмах припущення. В цьому випадку використовується набір даних із трьома атрибутами та міткою. У цьому методі ключовим моментом є

визначення кількості записів для кожного окремого значення цільового атрибута, тобто атрибута з відсутнім значенням. Значення з найбільшою кількістю записів є найпоширенішим значенням. Растрові бітові вектори можуть бути використані для визначення найбільш поширеного значення наступним чином. Різні значення атрибутів можна отримати з файлу метаданих атрибуту відсутнього значення. Для кожного окремого значення пов'язаний растровий вектор витягується з індексного файлу. Легко підрахувати кількість записів для кожного окремого значення, підрахувавши кількість записів на кожному отриманому бітовому векторі. Бітовий вектор з найбільшою кількістю одиниць представляє собою бітовий вектор найпоширенішого значення в цільовому атрибуті. Розглянемо отримання в бітовому векторі кожного окремого значення. Бітовий вектор з найбільшою кількістю одиниць відображає найбільш поширене значення. Однак цей підхід не може бути використаний без апроксимації у випадку пошуку, оскільки кожен бітовий вектор представляє набір значень, а не одне значення.

Проаналізуємо метод найбільш поширених значень на основі концепцій. Спочатку будується бітовий вектор умови запиту для отримання всіх записів з тією ж міткою, що і мітка відсутнього запису. Для кожного окремого значення у файлі метаданих цільового об'єкта створюється запит, який передається в систему растрових запитів для отримання бітового вектора, що представляє всі записи, що дорівнюють цьому значенню. Буде виконано логічну операцію між бітовими векторами всіх різних значень і бітовим вектором мітки. Обчислюється кількість одиниць на бітовий вектор кожного окремого значення, що представляє кількість записів з однаковим значенням і міткою. Бітовий вектор значення з найбільшою кількістю одиниць може бути використаний як приписане значення для відсутнього значення. Аналогічно для найбільш поширеного методу значень ця стратегія стає менш точною в разі використання пошуку при побудові растрового індексу. Використання підходу з алгоритмом на основі концепції покращує пошук.

Проаналізуємо метод значень середнього атрибута на основі концепції. Подібно до попереднього методу, бітовий вектор з однією умовою буде побудований для повторного отримання всіх записів, які мають збіги мітки з міткою відсутнього запису значення. Для атрибуту введене значення є середнім арифметичним усіх значень цільових записів цього атрибута. У випадку з пошуком техніка змінюється. Потрібно перевірити файл метаданих атрибуту, щоб визначити об'єднаний набір, який містить мітку пропущеного запису значення. Будуємо умову запиту для отримання всіх записів у цьому наборі. Ці записи можна перевірити, щоб визначити записи, які мають таку саму мітку, як і запис із відсутнім значенням. Введене значення буде середнім значенням для всіх записів з однаковою відсутньою міткою запису. У разі пошуку техніка змінюється. Потрібно перевірити файл метаданих атрибуту, щоб визначити об'єднаний набір, який містить мітку пропущеного запису значення. Будуємо умову запиту для отримання всіх записів у цьому наборі. Ці записи можна перевірити, щоб визначити записи, які мають таку саму мітку, як і запис із відсутнім значенням. Введене значення буде середнім значенням для всіх записів з однаковою відсутньою міткою запису.

Представлені концепції для проведення серії експериментів, проведення оцінки запропонованих підходів до видалення на основі растрових зображень. Зокрема, запропоновані підходи до видалення порівнюватимуться з двома способами: прискорення переслідування; точність введених значень. У порівнянні точності, щоб зосередитися на впливі різних підходів до видалення відсутніх значень на процес інтелектуального аналізу даних, використовуватимемо алгоритм класифікації для класифікації записів набору даних після того, як процес видалення відсутніх значень застосовується за допомогою різних рецепторів, а потім коефіцієнт шуму набору даних оцінюється в кожному випадку. Всі підходи можна реалізувати на C-подібній мові. Експерименти зі швидкістю виконання можуть бути проведені з використанням двох різних наборів даних. Перший набір даних є набором даних про

температуру. Цей набір даних складається з семи атрибутів і майже записів, які можуть вміститися в пам'яті, і тим самим допомогти оцінити ефективність запропонованих підходів, коли набори даних поміщаються в пам'ять. Використовуватимемо штучний набір даних, який був згенерований з десятьма числовими атрибутами та міткою, і великою кількістю записів. Цей набір даних використовується як приклад великого набору даних, допомагаючи оцінити продуктивність запропонованих методів видалення на резидентних дискових наборах даних.

Для реалізації алгоритму видалення використовувалися тільки методи, оскільки він не може бути застосований без них. Цей експеримент націлений на обидва набори даних (наприклад, на Берклі та штучний набір даних). Порівняння швидкості виконання при обробці відсутніх значень наборів даних за методом з різною кількістю відсутніх значень можна здійснити одночасно з визначенням продуктивності. Кількість відсутніх значень у наборі даних варіюється і вони розподіляються за різними атрибутами. Порівняння припускає, що індекс генерується спеціально для процесу видалення. Однак на практиці індекс може бути згенерований заздалегідь з іншої причини, наприклад, потреба в підтримці підмножин запитів, і час генерації індексу може бути змінено. Для оброблення результатів експерименту можна використати два показники, які корелюють між собою: різна кількість відсутніх значень в обох наборах даних; нормалізований час виконання, який обчислюється як частка найдовшого часу виконання. Загальний час виконання методу, навіть включаючи необхідний час для генерації файлів для цільового набору даних, менший, ніж загальний час виконання методу перебору для обох наборів даних. Зі збільшенням кількості відсутніх значень перевага пропонованого підходу стає ще більш значною, якщо порівнювати з методом стандартного пошуку. Відносна продуктивність методу ще краща у випадку резидентних наборів даних на диску. Причина полягає в наступному: час видалення для одного значення є сумою часу для знаходження найближчих записів до запису відсутнього значення і часу, необхідного для

отримання цих записів, незалежно від того чи з основної пам'яті або з диска. У випадку з набором даних весь набір даних вміщується в пам'ять, а значить, загальний час виконання залежить в першу чергу від часу на пошук найбільш схожих записів. Однак для великого набору даних потрібні операції з диском для отримання записів з диска в пам'ять.

Проаналізуємо метод найбільш поширеного значення. Для попереднього алгоритму показаний нормалізований загальний час виконання алгоритму з використанням трьох методів. Оскільки всі відсутні значення одного атрибута будуть замінені найпоширенішим значенням цього атрибута, продуктивність цього алгоритму залежить від кількості атрибутів, що містять пропущені значення, а не від кількості відсутніх значень. Перший метод прискорює процес видалення в порівнянні з методом перебору. Другий метод працює тільки на попередньо згенерованих растрових векторах, тому збільшення розміру набору даних не впливають на працездатність методу на відміну від решти методів. Іншими словами, розрив у продуктивності між загальним часом виконання в них збільшується в міру того, як розглядаємо більший набір даних. Однак другий метод працбє гірше, коли збільшується кількість різних значень у цільовому атрибуті. У цьому випадку збільшується кількість згенерованих растрових векторів, що негативно позначається на продуктивності методу.

Аналіз методу найбільш поширеного значення на основі концепції подібний до поперднього методу. Метод на основі концепції може бути реалізований як за допомогою трьох попердніх методів їх комбінацією, так і поперднього методу на основі концепції. Нормалізований загальний час виконання методів з різною кількістю відсутніх значень в обох цільових наборах даних працює ефективно. Методи прискорюють процес видалення в порівнянні з методом пошуку і видалення. У загальному часі переважає час генерації індексу. Якщо індекс можна попередньо згенерувати, то метод може забезпечити дуже великі прискорення. У наборі даних цей метод задовольняє більш високу продуктивність в порівнянні з методом з більшою кількістю відсутніх значень.

Однак у штучному наборі даних цей метод дає вищу продуктивність. Причина такої поведінки полягає в тому, що він залежить від кількості різних значень у цільових атрибутах, тобто, в даному випадку, атрибута відсутнього значення та атрибута label, тоді як другий метод залежить від кількості відповідних записів для кожного відсутнього значення. У наборі даних кількість різних значень у цільових атрибутах є великою порівняно з штучним набором даних, які негативно впливають на продуктивність ВВІ.

Метод середнього значення атрибута на основі концепції демонструє ефективність при реалізації алгоритму, який включає два попердні методи. На обох цільових наборах даних методи є швидшими з різною кількістю відсутніх значень. Так само, як і у випадку з другим методом у наборі даних він швидший за нього через більші різні значення в цільових атрибутах. Однак у штучному наборі даних приріст продуктивності від методу є більш значним, оскільки метод не вимагає обробки введення-виведення в порівнянні з попереднім методом.

Розглянемо вплив кількості різних значень. Для кожного окремого значення кожного атрибута будується один вектор растрових зображень, якщо не використовується пошук. Збільшення кількості різних значень у кожному атрибуті впливає як на час генерації індексу, так і на запропоновані методи видалення на основі растрових векторів. Щоб побачити цей вплив, оцінюємо продуктивність кожного алгоритму видалення, використовуючи методи з різною кількістю різних значень у кожному атрибуті. Експеримент передбачатиме, що пошук не використовується. У цьому експерименті генеруємо п'ять штучних наборів даних з однаковою кількістю записів і атрибутів, але з різною кількістю різних значень у кожному атрибуті. Продуктивність методів з різною кількістю різних значень показуватиме, що час виконання збільшується зі збільшенням числа різних значень. Більш значущим спостереженням є те, що перший метод стає гіршим при дуже великій кількості різних значень. Другий метод ґрунтується на скануванні растрових векторів для оцінки відсутнього значення,

а збільшення кількості різних значень збільшує кількість згенерованих растрових векторів, що уповільнює процес видалення. Загалом, буде краще використовувати пошук, коли кількість різних значень велика.

Розглянемо концепцію експерименту для оцінка масштабованості запропонованих методів при введенні відсутніх значень у набори даних зі збільшенням розмірів. У експерименті використовується певна кількість штучних наборів даних. Ці набори даних генерувалися випадковим чином двома різними способами: шляхом збільшення кількості атрибутів; збільшення кількості записів. Кожен набір даних також був згенерований, щоб включити відсутні значення, розподілені за різними атрибутами. Оскільки метою є оцінка масштабованості середовища виконання запропонованих методів, то зафіксовуватимемо лише загальний час виконання кожного алгоритму видалення, не включаючи час генерації індексу. Час виконання збільшується лише лінійно зі збільшенням розміру набору даних або за рахунок збільшення кількості записів, або за рахунок збільшення кількості атрибутів у кожному записі. Цей експеримент також показуже інше спостереження. Оскільки метод повинен аналізувати файли індексів, щоб ввести відсутні значення, збільшуючи кількість записів, то негативно впливає на ефективність першого методу порівняно з другим методом як в алгоритмах припущення найбільш поширених значень на основі концепцій, так і в алгоритмах припущення середніх атрибутів на основі концепції.

Проаналізуємо вплив методу видалення на інтелектуальний аналіз даних. Оскільки порівняння точності штучних наборів даних не є дуже змістовним, а розмір набору даних не був дуже важливим, використовуватимемо еталонні наборів даних для порівняння точності. Набори даних узагальнені, тому показуємо різні характеристики кожного набору даних, такі як кількість записів, атрибутів і класів. Розглядатимемо всі атрибути як числові атрибути. У випадку символьних атрибутів конвертуватимемо їх у числовий формат на етапі попередньої обробки Всі методи видалення є евристиками. Таким чином,

кінцевою метою двох методів є не отримання тих же результатів, що і методи повного сканування, а підтримка

Процес інтелектуального аналізу даних оцінюємо за різними алгоритмами видалення по відношенню до кінцевих результатів конкретного методу класифікації. Зокрема, використовується алгоритм класифікації, заснований на правилах. Цей алгоритм ґрунтується на генеруванні набору прогностичних правил асоціації з позначених записів у певному наборі даних для класифікації немаркованих записів у тому самому наборі даних. У порівнянні з іншими класифікаторами, заснованими на правилах асоціацій, цей алгоритм має ряд переваг, таких як: уникнення генерації надлишкових правил за допомогою динамічного програмування; генерація невеликого набору правил асоціацій; досягнення більш високої точності. Оскільки головною метою є процес видалення, то використовуємо лише один алгоритм класифікації для порівняння точності різних запропонованих процесів видалення.

Видалення за допомогою різних методів застосовується до наборів даних перед класифікацією, а згодом отриманий класифікатор оцінюється за допомогою популярної метрики, якою є коефіцієнт шуму. Цей показник визначає рівень шуму в певному наборі даних, тобто відношення записів, визначених як шумні, до загальної кількості записів. У свою чергу, запис вважається зашумленим, якщо маркування його за допомогою алгоритму призводить до неправильної мітки. Ефективний алгоритм видалення повинен вводити значення, які зменшують коефіцієнт шуму. У цьому експерименті порівнюватимемо коефіцієнт шуму для введених наборів даних після застосування методів відповідно.

Значення обчислюється наступним чином. Кожна мітка позначається алгоритмом точніше, а найпоширенішою міткою серед найближчих записів є той, що перебуває в центрі вибірки. Якщо запис неправильно класифіковано, то значення шуму збільшується на одиницю.

Використовуватимемо стандартну техніку з алгоритмом для класифікації

даних записів набору даних, тобто повний набір даних після операції врахування відсутнього значення розбивається на частини. Кожна частина використовується для навчання моделі для прогнозування мітки для інших записів решти частин. Кожен запис позначений загальною назвою, що складається з інших дев'яти частин.

Встановимо різні параметри, які вимагає алгоритм, виходячи зі значень за замовчуванням, наведених у попередньому аналізі та розгляді методів. Для коефіцієнта шуму використовуватимемо додаткові параметри. Порівнюючи за допомогою коефіцієнта шуму, можна оцінити вплив запропонованих підходів на результати алгоритму класифікації.

Порівняння підходу із запропонованими підходами до видалення розглядатимемо зі стратегією пошуку та без неї. Для цього використовуватимемо декілька наборів даних, які є штучними. Для цих наборів даних значення коефіцієнта шуму для всіх трьох базових методів майже однакові для різних алгоритмів видалення. Це спостереження показує, що при використанні растрових зображень поставлені значення призводять до того ж класифікатора якості, що і метод перебору. Коефіцієнт шуму важливий, коли видалення не застосовується. Для наборів даних дуже значне зниження коефіцієнта шуму досягається застосуванням методів видалення. В цілому методи не тільки підвищують ефективність видалення, але і допомагають підвищити якість процесу інтелектуального аналізу даних після видалення.

Розглянемо корекції методів за рахунок приписування відсутніх значень і використання растрової техніки для індексації неповних наборів даних.

Підходи для уникнення негативного впливу пропущених показників у наборі даних під час інтелектуального аналізу даних є досить ефективними. Вони включають в себе оцінку пропущених значень або побудову іншого представлення даних, яке може бути використано замість неповного набору даних. Такий напрям відноситься в категорію приписування відсутньої цінності. Багато існуючих підходів передбачають операцію прямого видалення з повних

рядків одного і того ж набору даних для оцінки відсутніх значень. Окрім методів, на яких ґрунтується цей підхід, інші підходи базуються на методі найближчих сусідах та кластеризації середніх значень. Підхід максимальної правдоподібності також є популярним підходом до оцінки відсутніх значень. У цьому підході статистична модель будується на заданому неповному наборі даних, використовуючи підхід машинного навчання, і оцінюються модельні параметри. Наприклад, для оцінки відсутніх значень може бути використаний алгоритм пошуку максимального значення. Аналогічним чином, нейронні мережі також можуть бути використані для побудови моделі даних для заповнення відсутніх значень при неповному наборі даних.

Розглянемо в цьому контексті іншу проблему, що виникає в процесі опрацювання значень. Це поліпшення виконання запитів на неповних наборах даних шляхом модифікації методів індексації. Одна з таких спроб заснована на растровій індексації. Мета роботи зовсім інша, оскільки оцінюємо відсутні значення, готуючи множину значень до подальшого аналізу та видобутку набору даних.

Набори даних, що виникають у реальних застосунках, як правило, мають багато проблем невизначеності або неповноти, однією з яких є проблема відсутності значень. Існує безліч існуючих методів для вибору ймовірного значення для відсутнього значення, щоб мати можливість підтримувати інші запити або аналіз набору даних. Однак загальною темою в цій області є те, що алгоритми приписування відсутнього значення не можуть масштабуватися до великих наборів даних. Проаналізуємо цей недолік. Зокрема, растрові зображення можуть бути використані для прискорення приписування відсутніх значень. Розроблені два підходи, у тому числі один, у якому растрові індекси можуть допомогти зменшити кількість записів, які потрібно отримати, і другий, де лише растрові зображення використовуються для вибору значення для заміни відсутнього значення.

Таким чином, завдяки концепції оцінювання отримано покращення

продуктивності. Навіть врахувавши час генерації індексу, розроблені методи є швидшими. Однак, якщо індекс вже створено для підтримки інших функцій, наприклад, підтримки обробки запитів, то покращення методів суттєві. Крім того, результат процесу інтелектуального аналізу даних залишається приблизно таким же, як і при використанні стандартних методів.

## 2.2 Балансування навантаження за допомогою індексування растрових зображень

Запропонувані два різних рішення проблеми видалення відсутніх значень у великих наборах даних за допомогою індексації растрових зображень надають можливість використовувати переваги індексування растрових зображень, використовуючи його для покращення однієї з найбільш трудомістких операцій у просторових наборах даних, тобто для операції просторового з'єднання.

Просторові дані виникають у різних контекстах, включаючи географічні інформаційні системи, наукове моделювання та медичні застосування. Зі збільшенням розміру просторових даних все більше уваги приділяється ефективному розпаралелюванню алгоритмів просторового з'єднання. Існуючі алгоритми паралельного просторового з'єднання страждають від проблеми дисбалансу навантаження. Підхід до балансування навантаження базується на використанні растрових зображень як зведеної структури цільових даних. Використаємо растрові зображення для прискорення послідовної обробки в пам'яті. Конкретні алгоритми, де основна ідея полягає в ефективному балансуванні навантаження за допомогою растрових зображень, та пам’яті, яка використовує растрові зображення для ефективного кроку фільтрації просторового з'єднання, порівнюватимемо з існуючими методами.

З поширенням просторових даних у різних сферах застосування, таких як географічні інформаційні системи, наукове моделювання та аналіз медичних даних, розробка надійних систем просторових запитів стала необхідною.

Підтримка низького часу відповіді на запити при одночасному збільшенні розмірів наборів даних зараз є проблемою в цих областях. Найбільш поширеними просторовими запитами, які часто використовуються в просторових застосунках, є найбільш витратні з них з обчислювальної точки зору.

Розглянемо питання масштабованості запитів на просторове з'єднання. Зокрема, цей тип запиту спрямований на пошук пар просторових об'єктів із двох заданих просторових наборів даних, які задовольняють певному просторовому відношенню, тобто перетинаються, містять, торкаються, примикають тощо. Цей тип запиту на об'єднання називається стандартним просторовим запитом на об'єднання. У цьому випадку система запитів повинна об'єднувати різні набори даних щодо просторового відношення перетину. Як ще один приклад, у системах з максимальною кількістю транспортних засобів вона має вирішальне значення для збалансування трафіку. У цьому випадку регіони та набори даних маршрутизації транспортних засобів об'єднуються відносно відношення для генерації результату запиту. Цей тип запиту на об'єднання називається запитом на просторове об'єднання. В останнє десятиліття було запропоновано кілька алгоритмів для прискорення просторового об'єднання над просторовими наборами даних. Існуючі методи можна розділити на дві основні категорії: методи внутрішньої пам'яті, де операція об'єднання виконується повністю в основній пам'яті; методи зовнішньої пам'яті, при яких просторові дані зберігаються в основній пам'яті лише частково. Прикладами методів внутрішньої пам'яті є вкладений цикл. Ці методи вимагають, щоб обидва об'єднані набори даних знаходилися в пам'яті під час обробки. Для порівняння приклади методів просторового з'єднання зовнішньої пам'яті більше підходять для великих просторових наборів даних, які не можуть поміститися в основну пам'ять.

Індексування зазвичай використовується для підвищення продуктивності операцій просторового з'єднання. Існуючі методи просторового з'єднання на

основі індексування відрізняються залежно від того, чи вимагають вони індексації обох або одного з двох заданих наборів даних. Наприклад, якщо обидва просторові набори даних індексуються, то метод об'єднання може бути використаний з різними методами індексування, щоб прискорити операції пошуку даних. Якщо індексується лише один набір даних, то для виконання операції об'єднання може бути використане об'єднання. Розбиття методами зовнішньої пам'яті також є основою для розпаралелювання. Паралельна версія методу просторового з'єднання на основі дерева полягає в наступному. Методи розбиття сітки також були розпаралелені. Просторове з'єднання є паралельною версією алгоритму і, як було показано, перевершує інші методи.

Використовуватимемо растрове індексування для подолання обмежень як внутрішніх, так і зовнішніх методів просторового з'єднання і досягнення розпаралелювання, яке краще збалансоване навантаженням. Зокрема, пропонуємо два методи просторового з'єднання на основі растрових зображень, а саме: метод зовнішньої пам'яті; метод внутрішньої пам'яті.

Існує кілька причин для вибору растрового індексування для прискорення операцій просторового з'єднання. При використанні растрових зображень можна скористатися швидкими побітовими логічними операціями. Растрові індекси містять інформацію, необхідну для виконання етапу фільтрації просторових з'єднань, без необхідності доступу до вихідних даних. Іншими словами, растрові зображення є не тільки структурою індексування, але також можуть служити компактним узагальненням даних. Це може зменшити потребу в отриманні великомасштабних даних. Растрові індекси можуть бути побудовані поверх існуючого сховища даних, тобто вони не вимагають реорганізації даних під індекс. Така реорганізація даних є надзвичайно трудомісткою і може бути невиправданою, якщо єдиною або основною метою індексації може бути підтримка обробки запитів. Просторові набори даних не оновлюються, оскільки робота з регулярно оновлюваними наборами даних є добре відомим обмеженням растрового підходу. Растрове індексування використовувалося для прискорення

операції просторового з'єднання. Наприклад, растрове індексування використовується для відстеження просторових блоків, які генеруються за допомогою структури дерева фільтрів для призначення просторових об'єктів різним розділам просторового простору. Однак растрове зображення використовується двома різними способами: розбиття просторового простору; представлення просторових об'єктів під час операції об'єднання.

У методі растрове індексування використовується для підтримки ефективної стратегії розбиття на розділи. Кожен розділ містить просторові дані з обох цільових наборів даних, які можуть бути оброблені за допомогою будь-якого методу об'єднання внутрішньої пам'яті. Балансування навантаження, яке є основною проблемою більшості існуючих алгоритмів, може бути досягнуто шляхом сканування бітових файлів індексів і визначення найкращої точки розбиття в кожному просторовому вимірі. Для отримання змоги уникнути зайвого зберігання результатів просторового об'єднання з різних розділів у методі продуктивність підвищується шляхом групування набору значень точок об'єктів в єдине значення, щоб зменшити загальну кількість порівнянь в операціях з'єднання. Крім того, масштабованість покращується шляхом виконання операції просторового з'єднання з використанням згенерованих індексів замість сам набір даних. Для оцінки обох запропонованих алгоритмів просторового з'єднання використовується набір даних і використовуються штучні набори даних.

Операція просторового з'єднання об'єднує два просторових набори даних для знаходження пар просторових об'єктів, які задовольняють певному просторовому співвідношенню. У просторовому просторі кожен об'єкт може бути представлений великою кількістю точок. Наприклад, багатокутники представлені сотнями або тисячами точок у просторовому просторі. Щоб порівняти кожну пару обох заданих просторових множин, потрібно порівняти кожну пару точок з обох множин. Таким чином, операція просторового об'єднання розглядається як трудомістка операція, особливо у випадку великого

розміру набору даних.

Існуючі методи просторового з'єднання можна розділити на методи внутрішньої пам'яті і методів зовнішньої пам'яті. Більшість методів внутрішньої пам'яті виконують операцію просторового з'єднання в два етапи, щоб скоротити час, що витрачається на цю операцію: фільтрація; уточнення. На етапі фільтрування використовується приблизне представлення просторових об'єктів для початкового вибору набору відповідних пар кандидатів з обох просторових наборів даних. Головний завантажувальний сектор об'єкта це найменший прямокутник, який повністю охоплює цей об'єкт, такий, що сторони прямокутника також паралельні просторовим осям. Представлення просторового об'єкта лише з двома значеннями в кожному вимірі зменшує простір пам'яті та час обчислення операції об'єднання. Крок фільтрування дає лише початковий результат для операції просторового з'єднання, оскільки використовується наближене представлення просторових об'єктів. Цей початковий набір результатів може бути потім бути обробленим для отримання остаточних результатів, використовуючи етап уточнення. На цьому кроці для точності використовується повне представлення просторових пар кандидатів.

Поточні зусилля з розпаралелювання операції просторового з'єднання будуються на основі методів зовнішньої пам'яті, де ключовим етапом є розбиття, тобто поділ просторового простору на множину розділів. Кожен розділ містить невеликий набір просторових об'єктів з обох наборів даних, які можуть бути об'єднані за допомогою будь-якого існуючого методу просторового об'єднання в пам'яті. Просторові об'єкти можуть бути призначені одному розділу або декільком розділам. Алгоритми розбиття фокусуються на поділі простору, що не обов'язково призводить до рівномірного поділу об'єктів. Як наслідок, для викривлених наборів даних алгоритм має поганий баланс навантаження. Щоб проілюструвати це, можна провести експеримент з 64 ядрами для оцінки робочого навантаження на кожному ядрі за допомогою алгоритму, що використовувався як алгоритм об'єднання в пам'яті для кожного розділу. Два випадково згенеровані

набори даних використовувалися як розподілені. Цей алгоритм страждає від поганого балансування навантаження при спотворенні даних.

Розглянемо розбиття та алгоритми на основі растрових зображень. Розглянемо спочатку два різні алгоритми просторового з'єднання на основі растрових зображень: просторове з'єднання на основі розбиття растрових зображень; просторове з'єднання на основі растрових зображень у пам'яті. Він використовує растрові зображення для ефективного та результативного поділу просторового простору. На відміну від багатьох з існуючих алгоритмів, цей алгоритм дозволяє динамічно балансувати навантаження для підвищення паралельної продуктивності. Крім того, в цьому алгоритмі генерується набір растрових векторів для представлення значень для обох просторових наборів даних. Основна ідея полягає в тому, щоб просто отримати доступ до растрових зображень, а не до всього набору даних під час етапу фільтрації. Використовуємо стиснені растрові вектори для обох запропонованих алгоритмів, що ще більше підвищує ефективність пам'яті.

Просторове об'єднання на основі растрових розділів базується на розробленому алгоритмі розбиття простору, який називається алгоритмом просторового об'єднання на основі розбиття растрових зображень, що використовує растрове індексування для динамічного поділу просторового простору. Припустимо, що наявні два набори просторових даних, кожен з яких містить набір просторових об'єктів. Кожен головний завантажувальний сектор представлений двома значеннями в кожному вимірі. У кожному вимірі алгоритм генерує набір растрових зображень для представлення атрибутів. Мета полягає в тому, щоб ефективно сканувати просторовий простір на наявність відповідних точок, щоб розділити простір у кожному вимірі. Процес поділу спрямований на призначення достатньої кількості об'єктів з обох наборів даних кожному розділу, щоб забезпечити майже однакове навантаження для кожного розділу. Зокрема, нам потрібно визначити початок і кінець кожного розділення в кожному вимірі, таким чином, щоб кількість елементів між ними дорівнювала або трохи

перевищувала цільове значення принаймні одного набору даних. Кількість цільових об' єктів для кожного набору даних дорівнює загальній кількості просторових об'єктів, поділеній на кількість розбиття в кожному вимірі. У цьому випадку максимальне зв'язане значення вважається точкою і припускає просторовий простір з двома вимірами, однак його можна легко розширити, щоб охопити просторові об'єкти в будь-якій кількості просторових вимірів. У рядках списки розбиття вимірів генеруються за допомогою методу. У рядках для кожного розділу перетину між двома розбиттями в розмірах існуючі об'єкти витягуються шляхом виконання операції між цільовими бітовими векторами. Нарешті, виконується операція просторового з'єднання між отриманими об'єктами.

Розглянемо просторове з'єднання на основі растрових зображень у пам'яті. Метод паралельного просторового з'єднання подібний до того, який розроблено, вимагає ефективного послідовного методу обробки кожного розділу. Виявляється, растрові зображення можуть бути використані для прискорення операції просторового з'єднання в пам'яті. Тому, розглядатимемо новий алгоритм просторового з'єднання в пам'яті.

Нагадаємо, що двома кроками в типовій реалізації операції просторового об'єднання є: крок фільтрації, на якому операція об'єднання виконується з між обох наборів даних, і крок уточнення, який використовує повне представлення пар об'єктів, виведених фільтром кроку для отримання набору результатів. На етапі фільтрації висока вартість в першу чергу пов'язана з великою кількістю необхідних порівнянь, і відомі дослідження щодо покращення просторових з'єднань були зосереджені на цьому кроці. У розробленому алгоритмі просторові порівняння прискорюються за допомогою растрових зображень, а не вихідних даних, і отримують вигоду не тільки від меншого розміру, але й від апаратної підтримки швидких побітових операцій. Крім того, кількість порівнянь потенційно зменшується шляхом групування набору просторових значень для порівняння в одне значення. Переваги використання растрових індексів є

найбільшими для наборів даних з низькою кардинальністю, тобто там, де маємо невелику кількість різних значень у кожному стовпці. Однак більшість існуючих просторових наборів даних є наборами даних з високою кардинальністю. Щоб усунути це обмеження, використовуємо значення з фіксованою точністю або приблизні значення. Таким чином, можемо зменшити кількість різних значень у кожному стовпці просторових даних і покращити продуктивність. В представленні просторових об'єктів кожен об'єкт представлений двома значеннями в кожному вимірі. Метод використовує ідею фіксованої точності для представлення значень заданих наборів даних при побудові бітових векторів. Іншими словами, відбувається групування різних значень одного і того ж просторового атрибута в одне значення. Це зменшує кількість просторових порівнянь.

Розглянемо приклад запропонованої техніки апроксимації в алгоритмі, де потрібна операція з'єднання перетинів. Щоб оцінити перетин зосередившись лише на розмірності, зазвичай потрібно виконати шість операцій порівняння. Можна зменшити кількість порівняльних ізонів до чотирьох, представивши максимальне значення і максимальне значення, як єдине значення.

Таким чином, ще одним обмеженням існуючої техніки розбиття є надлишкові пари, що перетинаються, які можна отримати з різних розділів. Це обмеження можна вирішити, перевіривши контрольні точки.

## 2.3 Висновки до другого розділу

В результаті запропоновано автономні рішення на основі індексування для вирішення двох проблем «великих даних» у застосунках з інтенсивним використанням даних: приписування відсутніх значень; запит на просторове об'єднання. Розроблено два різні підходи, засновані на растрових зображеннях, для приписування відсутніх значень у великих наборах даних. Використано растрові зображення для покращення масштабованості та поганих проблем

балансування навантаження в існуючих алгоритмах просторового з'єднання. Також, розглянуто проблеми відмовостійкості в існуючих алгоритмах машинного навчання та інтелектуального аналізу даних. Розроблено алгоритм відмовостійкості на основі пам'яті, який порівняно з дисковими підходами, має особливості, які забезпечують методи контрольної точки та відновлення даних.

# 3 МЕТОД ЗАБЕЗПЕЧЕННЯ ПРОПУСКНОЇ ЗДАТНОСТІ ДИСКІВ ДЛЯ ЗАСТОСУНКІВ З ІНТЕНСИВНИМ ОБСЯГОМ ДАНИХ

## 3.1 Відмовостійкий підхід для великомасштабних систем

Обмеження пропускної здатності диска при обробці великого набору даних створює дві проблеми обчислень, що вимагають великих обсягів даних, тобто видалення відсутніх значень і операцію просторового з'єднання, і запропоновано рішення для обох проблем, використовуючи метод індексування растрових зображень для обробки даних у пам'яті та надання ефективних рішень для уникнення затримки вводу/виводу диску. Було розглянуто апаратні збої у великомасштабних системах як ще одне обмеження обробки застосунків, що вимагають великих обсягів даних.

Розглянемо вплив апаратних збоїв у великих системах. Однією з проблем, що виникають при виконанні на великомасштабних паралельних системах, є підвищена ймовірність і частота несправностей. Великомасштабні системи часто страждають від несправностей декількох типів у багатьох компонентах. Тому, деякі недавні моделі програмування, розглядають відмовостійкість як один з найважливіших напрямів при проектуванні. Він досягає відмовостійкості, використовуючи кілька реплік структур даних у постійному сховищі, що, можливо, призводить до значного обсягу введення-виведення на критичному шляху. Друга система усуває це обмеження, використовуючи стійкі розподілені набори даних, таким чином, що реплікація в пам'яті може бути використана для відмовостійкості. Однак для дуже великих наборів даних реплікація в пам'яті неможлива. У деяких випадках він розглядає диск як серверну частину для контрольних точок, що може значно уповільнити обчислення та збільшити рух даних. Аналогічно, третя система розглядається з відмовостійкою потоковою обробкою і використовує диск як внутрішню логіку для контрольних точок. Природно, перевагою використання відмовостійкої моделі програмування є той факт, що контрольна точка і відновлення автоматизовані. Однак, обмеження

продуктивності відмовостійкої моделі програмування через контрольні точки на основі диска або додаткові витрати на простір через контрольні точки в пам'яті є непривабливими для масштабування декількох алгоритмів при великому обсязі та масштабі обчислень. Отже, важливо враховувати відмовостійкість на основі логіки для алгоритмів, щоб можна було досягти масштабованої відмовостійкості з мінімальним обмеженням продуктивності.

Отже, будемо будувати ріщення на новому алгоритмі та використаємо властивості для гарантування складності простору для контрольних точок важливих структур даних. Представляємо нову техніку використання самого набору даних для контрольних точок дерева, тобто для стисненого дерева для збереження частих шаблонів та часткових наборів даних з інших процесів.

Набір даних – це сукупність спостережень. Кожне спостереження також називають зразком, вектором або транзакцією. Кожна вибірка може складатися з однієї або декількох ознак також змінних або атрибутів. Мічений набір даних складається з обґрунтованої істини для кожної вибірки. Наприклад, набір даних про транзакції може позначати кожну вибірку як шахрайську або безпечну. Алгоритми контрольованого навчання будують моделі на мічених наборах даних.

Зазвичай мічений набір даних розбивається на навчальні та тестові набори. Модель/класифікатор побудований на навчальній вибірці, а точність, як правило, вимірюється за допомогою тестового набору.

Частий генератор патернів базується на алгоритмі і знаходить елементи, які часто зустрічаються разом у транзакціях бази даних. Елемент визначається як частий, якщо його частота вища за визначений користувачем поріг. Запропоновано кілька алгоритмів для вирішення цього завдання. Алгоритм дуже популярний, оскільки він вимагає лише двох проходів у наборі даних, не вимагає генерації кандидатів і забезпечує стиснене представлення елементів частоти за допомогою дерева частих шаблонів. Розглядатимемо відмовостійкий алгоритм, завдяки його ефективним властивостям.

Під час першого проходу алгоритм знаходить елементи, які часто зустрічаються. Під час другого проходу він створює дерево, яке є модифікованим. Перший прохід вимагає простого сканування заданого набору даних, щоб знайти всі окремі часті елементи.

Етап створення дерева, тобто другий прохід, є найбільш трудомісткою частиною загального розрахунку . Отже, будемо орієнтуватися на відмовостійкий крок створення дерева алгоритму, оскільки він має вищу ймовірність відмови.

Відмовостійкі моделі програмування останнім часом є популярними і спостерігається сплеск масштабних і відмовостійких моделей функціонального програмування. Функціональне програмування, в свою чергу, використовує концепцію єдиного присвоєння, коли кожна мутація змінної записується, зберігається на постійному сховищі/пам'яті іншого вузла і відтворюється при виникненні помилки. Як приклад, розглянемо вставку в існуюче дерево. Кожна зміна або мутація повинні бути записані локально, і такі записи можуть бути збережені в постійному сховищі.

Тепер розглянемо значення для такого алгоритму. У багатьох випадках етап збереження нової версії дерева на диску виконується в кінці фази. Для двофазного алгоритму, де більша частина часу витрачається на другу фазу, перевага не досягається. Іншою можливою реалізацією може бути поділ загального обчислення на кілька кроків. Контрольна точка може бути виконана в кінці кожної фази скорочення. Однак тепер загальний час виконання збільшиться, так як збереження нової версії буде пов'язано або з записом на диск або в пам'ять сусіда. Оскільки фаза зниження є фазою блокування, програма спостерігатиме значні додаткові витрати на контрольні точки, що погіршить загальну продуктивність. Природно, що масштабований алгоритм повинен використовувати найкращу можливу продуктивність за рахунок використання вбудованого виконання, при цьому мінімізуючи витрати на контрольні точки, використовуючи неблокуючі методи.

Тепер, розглядаючи альтернативну модель програмування, розглянемо інтерфейс передачі повідомлень MPI, який був легко доступний і широко використовувався на суперкомп' ютерах і кластерах, і починає знаходити своє місце в системах хмарних обчислень. Незважаючи на те, що MPI часто критикують за недостатню підтримку відмовостійкості, нещодавня література та імплементації вказують на те, що відмовостійкість добре вирішується при постійних несправностях процесу.

Представлені односторонні примітиви MPI надають необхідні інструменти для перекриття зв'язку з обчисленнями. З огляду на це спостереження, зосередимося на використанні MPI для розробки відмовостійкого алгоритму.

Односторонній зв'язок MPI має кілька примітивів. Зокрема, операції MPI можна класифікувати за трьома категоріями: створення вікон, операції; примітиви синхронізації. MPI використовує концепцію вікон для розкриття області пам'яті, таким чином, що інші процеси можуть асинхронно зчитувати/оновлювати її, використовуючи примітиви відповідно. Зазвичай пам'ять попадає в проміжок часу між створенням вікна і руйнуванням. Ще однією новою функцією, наданою останнім часом MPI, є динамічні вікна. Ця функціональність дозволяє процесу динамічно виділяти пам'ять на віддаленому процесі. Перевагою цієї функціональності є відмовостійкість.

Складні методи можуть забезпечити дуже низьку підтримку відмовостійкості для алгоритмів. Крім того, реалізація цих алгоритмів також вимагає ефективного використання розширених функцій MPI, що робить так, що простіші моделі програмування мають обмеження, коли справа доходить до досягнення як відмовостійкості, так і ефективності.

Розглядуваний алгоритм для побудови методу є добре відпрацьованим алгоритмом для генератору патернів. В останні роки ряд дослідників запропонували великомасштабні алгоритми з розподіленою пам'яттю. Нещодавно запропонували алгоритм для вирішення дисбалансу навантаження та

комунікаційних складнощів алгоритму.

Представляємо поглиблене дослідження алгоритму для відмовостійкості. Враховуючи його двопрохідні властивості запропоновано новий алгоритм, який вимагає складності простору для збереження критичних структур даних, тобто в пам'яті інших обчислювальних вузлів. Запропонований алгоритм поступово використовує пам'ять, виділену для стандартного алгоритму для контрольних точок дерев і, можливо, часткової копії транзакцій з інших обчислювальних вузлів, забезпечуючи додаткові витрати на простір запропонованих алгоритмів. Щоб ще більше мінімізувати витрати часу на контрольні точки, рішення не тільки використовує неблокуючі властивості, але й використовує віддалений доступ до пам'яті за допомогою MPI на додаток до мінімізації будь-якої участі віддаленого процесу для контрольних точок. Використовуючи MPI та суміжні структури даних для реалізації запропонованих алгоритмів, можемо ефективно використовувати віддалений прямий доступ до пам'яті. Запропоновані розширення можуть бути включені до існуючих рішень, де клас алгоритмів може повторно використовувати вже виділену пам'ять для контрольних точок і відновлення.

Таким чином, запропоновано алгоритм на основі контрольних точок у пам'яті для великомасштабних систем. Запропонований алгоритм використовує зв'язок, що перекривається, з фазою побудови таким чином, що додаткові витрати на контрольні точки зводяться до мінімуму.

Пропонуємо три різні механізми відмовостійкості для паралельного алгоритму: відмовостійкий за замовчуванням; синхронний відмовостійкий на основі синхронної пам'яті; асинхронний відмовостійкий.

Використовуємо односторонній механізм MPI для асинхронної перевірки критичних структур даних. Завдяки нещодавнім розробкам в області односторонньої відмовостійкості MPI стало можливим використовувати MPI для обробки несправностей, забезпечуючи при цьому нативну продуктивність.

Проводимо глибоку оцінку запропонованих підходів, використовуючи

транзакції до ядер. Оцінка продуктивності вказує на те, що запропоновані підходи здатні ефективно використовувати пам'ять набору даних для контрольних точок. Використовуючи транзакції на ядрах, додаткові витрати на контрольні точки становлять вартість відновлення після кількох збоїв, яка в свою чергу не залежить від кількості процесів. Розглянуто запропонований алгоритм як потенційний алгоритм для розгортання у великих масштабах з дуже великими наборами даних.

Ефективність пропонованої відмовостійкої реалізації перевершує реалізацію того ж алгоритму, забезпечуючи належну середню швидкість. Систематична оцінка реалізацій на основі MPI у контексті розробки підходів, орієнтованих на застосунки, є необхідною користувачам.

Алгоритм показує ключові кроки паралельного алгоритму, який використовували як базовий для розробки відмовостійких алгоритмів.

Першим кроком є розподіл транзакцій вхідної бази даних між її процесами. Кожен процес бере участь в обчисленні свого локального дерева. Кожен процес сканує локальні транзакції і записує частоту кожного елемента. Для збору глобальної частоти використовується скорочення «всі до всіх». Після скорочення «всі до всіх» елементи з частотою, що перевищує поріг підтримки, зберігаються, а інші елементи відкидаються. Потім кожне число генерує локальне дерево, використовуючи свої локальні транзакції, які мають принаймні один частий елемент. Пізніше, він об'єднує своє локальне дерево з рештою дерев з інших процесів, щоб створити глобальне дерево за допомогою кільцевого алгоритму зв'язку. Нарешті, часті набори елементів створюються за допомогою вихідного глобального дерева.

Запропоновані відмовостійкі алгоритми зростання базуються на кількох підходах до проектування відмовостійкого алгоритму. Базовий алгоритм використовує диск як безпечне сховище для збереження проміжних дерев, тоді як оптимізовані алгоритми використовують пам'ять, спочатку виділену для транзакцій бази даних, для перевірки проміжних дерев та транзакцій інших

процесів з високим перекриттям зв'язку з обчисленнями, що досягається за допомогою методології MPI.

Для розробки відмовостійкого алгоритму існує кілька варіантів дизайну. Оскільки розглядаємо модель, то важливо розуміти вибір дизайну між відродженням нового набору процесів на запасному вузлі та продовженням виконання з існуючими процесами та вузлами. Використовуємо тривале виконання, в першу чергу тому, що для більшості систем складно повторно відродитися, приєднати процеси/вузол до існуючого набору процесів і продовжити відновлення. Натомість, тривале виконання забезпечує простий механізм для проведення відновлення, без значної залежності від зовнішнього програмного забезпечення. Алгоритм є основою для інших підходів.

В алгоритмі є дві критичні структури даних, які необхідні в процесі відновлення. Це самі транзакти бази даних і проміжні дерева, згенеровані процесами. При підході проміжні дерева, згенеровані кожним процесом, періодично зберігаються на диску. У багатьох суперкомп'ютерів диски розташовані віддалено, наприклад, віддалене сховище. В інших випадках також можна використовувати локально доступні накопичувачі. Транзакції бази даних вже є резидентними на диску. Отже, немає необхідності перевіряти транзакти бази даних.

Розглянемо рівномірний розподіл транзакцій бази даних між процесами, в яких транзакції доступні на кожному процесі. Нехай є певна кількість контрольних точок, які виконуються застосунком. Кількість контрольних точок виводиться як функція таким чином, що вартість контрольних точок може бути первірена на етапі створення дерева. Алгоритм також повинен зберегти файл метаданих, пов'язаний з деревами, який може використовуватися під час відновлення. Просторова складність файлу метаданих є незначною, оскільки потрібно зберегти лише кілька цілих чисел.

Нехай алгоритм представляє середній розмір дерева, створеного кожним процесом. Часова складність для контрольних точок проміжних дерев становить

мінімум з усіх значень.

Однак фактичний час до контрольної точки може збільшитися через суперечки між декількома процесами, які записують файл контрольної точки одночасно. Складність простору, що виникає при кожному процесі, дорівнює значенню найдовшого з них, який можна ще більше скоротити шляхом переробки існуючих контрольно-пропускних пунктів.

У підході відновлення ініціюється майстром. Він зчитує файл метаданих, пов'язаний з несправним процесом, який надає необхідну інформацію для проведення відновлення. Вибирається процес відновлення, який зчитує з диска контрольну точку дерева і об'єднує контрольне дерево з його деревом, в той час як лічильник зчитує транзакції процесу з диска і перерозподіляє їх між процесами, що залишилися.

Таким чином, часова складність алгоритму відновлення є функцією зчитування часткового набору даних і виконання алгоритму відновлення. У гіршому випадку потрібно заново виконувати всі транзакції несправного процесу. Отже, найгіршим випадком складності часу є середня вартість злиття транзакції в існуючому дереві. У гіршому випадку всі транзакції виконуються повторно.

## 3.2 Метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних

Кожен процес зберігає копію локального дерева в безпечному сховищі. Таким чином, реалізація пропнованого рішення залежить від перевірки локального дерева на диску. Цей файл, пов'язаний з іншим файлом метаданих, що описує контрольне дерево, зберігаючи набір значень опису, таких як: позначка часу контрольної точки; остання оброблена транзакція. Кожен процес асинхронно оновлює обидва файли, під час виконання. У разі невдачі операція відновлення виконується в два етапи: заздалегідь визначений процес

відновлення зчитує з диска останнє дерево несправного процесу; поєднує його з локальним деревом. У той же час майстер-процес зчитує файл метаданих, щоб прийняти рішення про набір транзакцій, які потрібно відновити з диска. Майстер-процес відновлює необроблені транзакції та перерозподіляє їх між рештою процесів.

Запропонований алгоритм значною мірою еквівалентний розробці відмовостійкого алгоритму з використанням програмних моделей. Однак перевагою є те, що він може конкретно скористатися перевагами вбудованого зв'язку за допомогою MPI, особливо коли доступні високопродуктивні міжмережеві з'єднання. Дисковий підхід змушує користувача відійти від ряду обмежень. До них відносяться непомірно високі витрати на введення-виведення для контрольних точок/відновлення локальних дерев і відновлення необроблених транзакцій, а також централізоване вузьке місце основного процесу в разі неможливості повторного зчитування необроблених транзакцій з диска.

Синхронна відмовостійкість на основі пам'яті зростання є основним обмеженням підходу бо те, що він використовує перевірку та відновлення на основі диска, є непосильним для масштабування алгоритму. Тому важливо враховувати відмовостійкий алгоритм на основі пам'яті. Оскільки доступний розмір пам'яті відносно малий у порівнянні з розміром диска, також непривабливо створювати додаткове ускладнення простору для контрольних точок у пам'яті дерев та транзакцій бази даних від інших процесів. Він передбачає метод контрольних точок, коли загальна просторова складність алгоритму залишається постійною. Крім того, перекриваємо контрольні точки дерев та транзакцій бази даних, використовуючи неблокуючі примітиви, що надаються односторонньою моделлю MPI. Представляємо контрольно-пропускний пункт і повторно методи з їх часово-просторовим аналізом складності.

Передумова постійної складності простору базується на двопрохідних

властивостях алгоритму. На етапі створення дерева, як тільки транзакція бази даних оброблена, пам'ять, зайнята транзакцією, може бути використана для контрольних точок. Використовуємо цю властивість алгоритму для перевірки дерев та транзакцій бази даних. Зокрема, як тільки транзакція оброблена, звільняємо пам'ять, споживану транзакцією, і виділяємо окреме вікно пам'яті, яке може бути використано іншими процесами для перевірки їх дерев та транзакцій бази даних. При використанні цієї методики загальна просторова складність алгоритму, крім оптимальної складності простору, є мінімізованою. Тимчасова складність є величиною для контрольних точок як дерев, так і транзакцій бази даних. Розглядаючи число контрольних точок при наївному алгоритмі кожен процес може перевірити точку його існуючого деревао до іншого процесу при кожному значенні контрольної точки, яке не є незначними, оскільки цей крок блокує завершення зв'язку, перш ніж продовжити обробку транзакцій, що залишилися, на кожному кроці перевірки і зі збільшенням розміру дерева додаткові витрати на блокування збільшуються. Отже, важливо враховувати неблокуючі методи контрольних точок, таким чином, щоб витрати на зв'язок з контрольними точками можна було перекрити з обчисленнями.

Алгоритм використовує односторонні неблокуючі методи MPI для контрольних точок. Зокрема, під час обробки транзакцій бази даних аналогічний обсяг пам'яті додається до вікна контрольної точки. Алгоритм використовує функцію динамічного розподілу в MPI, що дозволяє поступово збільшувати розмір простору пам'яті контрольних точок під час виконання. Однак ця техніка динамічного розподілу вимагає синхронізації між обома процесами, що співпрацюють, для виконання кожної окремої контрольної точки, що додає більше додаткових витрат до процесу контрольних точок. Додаткові витрати на контрольні точки надходять з різних джерел: час очікування до синхронізації; зв'язок, який є незначним на основі відомої моделі зв'язку; витрати на виділення та розподіл пам'яті.

Процес потребує контрольної точки в пам'яті процесу в кожен період часу,

тобто процес повторно ініціює простір контрольної точки, який може обробляти обробити точку локальним деревом. У цьому випадку процес може віддалено перевіряти своє локальне дерево на нове призначене місце без зв'язку з контрольною точкою процесу·

Припускаємо, що процес зазнає невдачі під час виконання фази дерева. На процес відновлення помилок у спрощеному випадку об'єднує контрольно-поточкове дерево, що зберігається в його пам'яті з локальним деревом. Якщо воно також зберіг частину транзакцій бази даних, то він перерозподіляє ці транзакції між іншими процесами, які все ще активні в обчисленнях. Відновлені транзакції можуть бути зібрані з пам'яті, якщо вони були перевірені до збою. У разі відновлення диска втрачені транзакції можуть бути зчитані з диска двома різними способами. Транзакції набору даних можуть зчитуватися з диска за допомогою головного процесу і рівномірно розподілятися між рештою процесів. Однак в цьому випадку доступ до диска буде найдорожчою частиною загального алгоритму відновлення. Отже, пропонуємо використовувати всі доступні процеси для паралельного зчитування зразків невдалих процесів з диску. При цьому кожен процес буде звертатися до диску тільки для читання транзакцій. Далі, оскільки не вдалося процесу провести перевірку даних попереднім процесом, то процес виконує критичний крок і контрольну точка можна дослідити автоматично на процесі. У найпростішому випадку можна припустити, що процеси з'єднані у віртуальній кільцевій топології. Використовуючи цю методологію, завжди існує принаймні одна копія дерева кожного процесу.

Основна перевага розробленого рішення полягає в тому, що він дозволяє уникнути читання/запису з диска. Природно, він досягає власної продуктивності за допомогою MPI і, як очікується, понесе низькі додаткові витрати на контрольні точки з неблокуючим одностороннім зв'язком MPI. Алгоритм відновлення використовує пам'ять для відновлення транзакцій бази даних, якщо це можливо. Розподіляючи транзакції невдалого процесу на інші активні

процеси, алгоритм здатний мінімізувати додаткові витрати на відновлення. У разі відновлення транзакцій на диску він використовує всі процеси для паралельного зчитування відновлених транзакцій з диска, щоб уникнути вузького місця основного процесу.

Підхід має два основних обмеження. Кожні два процеси повинні синхронізуватися у всіх контрольних точках, щоб поділитися адресою вектору контрольних точок та розміром транзакцій. Алгоритм вимагає розвантаження існуючого простору та виділення нового простору для вікна контрольних точок. Додаткові витрати на синхронізацію, дерозподіл і розподіл спостерігаються на етапі створення дерева.

У першому алгоритмі кожен процес виділяє три вектори пам'яті. Ці вектори використовуються для обробки контрольних точок з процесу, який включає набір параметрів для опису обох векторів контрольних точок. Ці вектори виділяються та виставляються для читання/оновлення кожним процесом за допомогою примітивів MPI.

Для контрольних точок у пам'яті він вимагає, щоб кожен процес вибирав інший процес для контрольної точки. Хоча сам підтримує будь-яку довільну топологію. У найпростішому випадку можна припустити, що процеси пов'язані у віртуальній кільцевій топології. Кожен процес використовує пам'ять сусіднього процесора для контрольних точок свого локального дерева та транзакцій. Таким чином, кожен процес повинен підготувати свої буфери контрольних точок та вектори контрольних точок транзакцій для обробки контрольних точок даних процесу, коли це необхідно під час відновлення.

Для виконання однієї контрольної точки кожній парі процесів необхідно виконати три операції. Він збільшує розмір метаданих і структури даних таким чином, що нова контрольна точка може бути оброблена. Операція визначення розміру локального дерева з контрольною точкою вимагає синхронізації ними. Зокрема, перший процес надсилає запит на контрольну точку, включаючи обсяг даних, які потрібно перевірити. А він використовує

динамічний механізм MPI для збільшення розміру контрольного простору. Нова віртуальна адреса передається процесу, який використовується для перевірки фактичних даних за допомогою операції MPI.

Процес також може перевіряти локальні транзакції, що залишилися в пам'яті, щоб уникнути зчитування його з диска в разі збою. Якщо помилка виникає до перевірки транзакцій, то решта транзакцій відновлюються з диска. Однак, якщо процес зазнає невдачі після того, як транзакції набору даних були перевірені, то вони можуть бути перерозподілені безпосередньо на інші доступні процеси. Контрольна точка транзакцій може бути виконана аналогічно контрольній точці на векторі цільового процесу.

Алгоритм показує алгоритми контрольної точки та відновлення. У процедурі ініціалізації кожен процес створює три вектори для обробки контрольних точок процесу. Ці вектори виділяються і експонується за допомогою технології MPI для полегшення дистанційного читання/оновлення. Обидві процедури ілюструють роботу контрольних точок як для локального дерева, так і для транзакцій відповідно. Процес синхронізується зі своїм вихідним процесом, отримуючи розмір контрольної точки та змінюючи розмір буфера контрольної точки для обробки даних. Процес завершує операцію синхронізації, надсилаючи нову векторну адресу контрольної точки вихідному процесу. Далі процес використовує функцію MPI для перевірки своїх даних і оновлює вектор метаданих у пам'яті цільового процесу.

Процедура показує алгоритм відновлення. Заздалегідь визначений процес відновлення використовується для відновлення невдалого процесу шляхом злиття локального дерева, яке воно має у своїй пам'яті з локальним деревом. Крім того, транзакції невдалих процесів можуть бути відновлені за допомогою вектору метаданих безпосередньо з пам'яті відновлення за наявності, або з диска, якщо ні. Відновлення на диску слід виконувати паралельно, щоб прискорити загальний час відновлення.

У цьому підході переваги використання контрольних точок у пам'яті та

транзакцій бази даних. Однак є кілька обмежень. Зокрема, пара процесів повинна синхронізуватися для розподілу пам'яті та обміну адресами, що знижує загальну ефективність моделі MPI.

Для усунення обмеження потрібно використовувати односторонній механізм контрольних точок, тобто асинхронну відмовостійкість на основі пам'яті. В рамках цього використовуємо пам'ять вже оброблених транзакцій для контрольних точок замість виділення нового простору.

Контрольна точка зберігається на процесі. Для того, щоб забезпечити дійсно односторонній механізм для контрольних точок, він повинен переконатися, що розмір його контрольної точки менший за розмір вже оброблених транзакцій. В цьому методі досягаємо цієї мети, використовуючи атомарні операції над змінними, виділеними за допомогою MPI, і піддаючи їх для зчитування/оновлення іншими процесами. Оригінальний паралельний алгоритм дещо модифікований для атомарного оновлення розміру доступного простору контрольних точок. Цей крок не вимагає зв'язку з будь-яким іншим процесом. Коли він вирішує поставити контрольну точку своєму дереву, то воно атомарно зчитує значення доступного контрольного простору. Ретельно спроектувавши інтервал контрольних точок, дуже ймовірно, що розмір доступного контрольного простору є більшим за розмір, який вимагається числом. У випадку двох процесів періодично зчитується ним наявний контрольний простір до тих пір, поки умова не буде виконано. На практиці такої ситуації не спостерігається. У загальному випадку процес просто ініціює контрольну точку за допомогою MPI. Окрім локального дерева, решта необроблені транзакції процесу також можуть бути позначені пам'яттю, якщо є достатньо місця. Контрольні транзакції це одноразова операція, яка покращує процес відновлення шляхом зчитування транзакцій невдалих процесів безпосередньо з місця пам'яті контрольних точок, а не з диску.

Ефективність алгоритму контрольних точок полягає в його простоті. На відміну від другого алгоритму, тут не потрібна синхронізація між будь-якою

парою процесів, а також не потрібне виділення пам'яті. Використовуючи MPI на високопродуктивних міжз'єднаннях, очікуємо, що буде майже оптимальним алгоритмом контрольних точок для розробки великомасштабного алгоритму. В іншому випадку всі доступні процеси паралельно відновлювали з диску необроблені транзакції збої процесу.

Найгірша часова складність підходу схожа на другий алгоритм. У найгіршому випадку всі транзакції зчитуються з диску паралельно, як зазначено в підході часової складності та переобчислюється за допомогою журналу| процесів.

У багатьох випадках, особливо коли помилка виникає на пізніх стадіях фази збирання дерева заповнення диску буде повністю уникнено, що призведе до набагато швидшого відновлення порівняно з найгіршим сценарієм.

Алгоритм ілюструє процедури контрольної точки та відновлення для алгоритму. Під час процедури ініціалізації кожен процес має свій вектор, який містить локальний набір транзакцій. Кожен процес створює єдиний вектор, тобто метадані, який представляє набір параметрів для опису стану.

Трансвекторні та контрольні дані вихідного процесу зберігаються в пам'яті. У цьому випадку використовується технологія MPI для спільного використання обох векторів з іншими процесами.

Кожен процес повинен прочитати метаданні на цільовому етапі процесу, щоб перевірити наявність вільного місця перед контрольною точкою. Решта транзакцій виконується лише один раз після наявності вільного місця.

Процедура показує алгоритм відновлення в підході. Подібно до алгоритму відновлення, процес відновлення використовується для відновлення шляхом злиття останнього дерева з контрольною точкою, яке воно має, з локальним деревом. Необроблені транзакції можуть бути відновлені з пам'яті процесу, якщо вона була позначена до відмови, або безпосередньо з диска.

Представляємо далі детальну оцінку продуктивності запропонованих відмовостійких алгоритмів.

Для кожного відмовостійкого алгоритму представляємо детальний аналіз продуктивності додаткових витрат на контрольну точку та відновлення.

Для оцінки продуктивності використовуємо оптимізацію компілятора за допомогою компілятора. Для оцінки різних запропонованих відмовостійких алгоритмів використовуємо генератор наборів даних для генерації великомасштабних штучних наборів даних. Генератор наборів даних використовується в ряді досліджень і точно відображає характер транзакцій в реальних наборах даних.

Проаналізуємо додаткові витрати на підтримку відмовостійкості зростання. Якщо отримуємо масштабування, тобто збереження загальної роботи на постійній основі та збільшення кількості процесів, то алгоритм добре масштабується для процесів, в яких спостерігаємо суперлінійне прискорення за рахунок кращого використання кешу. Симулярні прискорення спостерігаються для всіх алгоритмів відповідно. Оскільки поріг підтримки високий, то кількість частих номерів процесів відносно невелика. Отже, загальний час обчислень становить менше допустимого значення. Природно, що уповільнення, яке спостерігається високе. Поточні реалізації MPI не завжди оптимізовані для масової передачі даних. Порівняння продуктивності алгоритмів з використанням транзакцій і порогу підтримки дає змогу спостерігати аналогічну закономірність. Високі додаткові витрати для підходів. Для великих транзакцій на процес, розмір дерева більший. Оскільки час виконання MPI менш оптимізований для масової передачі, а уповільнення є меншим, але незначним.

Ефективність запропонованих підходів з трансакцій та порогом підтримки показує уповільнення порівняно з базовим паралельним алгоритмом, тоді як для розробленого методу спостерігається лише невеликий відсоток додаткових витрат.

Очевидно, що він перевершує інші підходи, особливо дисковий підхід без будь-яких додаткових складнощів. При сильному масштабуванні, яке зазвичай є проблемою для алгоритмів розподіленої пам'яті, відносні додаткові витрати

розробленого методу зменшується. Це пов'язано з неоптимізованими протоколами MPI для масової передачі даних. Очікується, що з подальшою оптимізацією, як очікується в найближчому майбутньому, ці додаткові витрати ще більше зменшаться. З огляду на складність простору і все ще прийнятні додаткові витрати на контрольні точки, очікуємо, що запропонований алгоритм буде використаний як основа для майбутніх досліджень і практичного розгортання.

Ефективність будь-якого механізму відмовостійкості пов'язана з усуненням збоїв, крім додаткових витрат на контрольні точки. Оцінюємо додаткові витрати на відновлення в разі відмови шляхом введення помилок в паралельне виконання. Для імітації несправностей вибираємо процес для збою і точку відмови. При досягненні точки відмови цей процес виключається з виконання. Припускаємо, що точка відмови після обробки транзакцій набору даних справедливо порівнюється з алгоритмом відновлення для підходів.

У разі збою алгоритм відновлення повинен відновити дерево помилкового процесу з диска, порівнюючи з підходами, де дерево відновлюється з пам'яті. У першій серії експериментів обчислювали прискорення, використовуючи підходи, порівняно з підходом до відновлення одного процесу відмови. Це негативно впливає на продуктивність підходу порівняно з двома іншими підходами.

Таким чином, маючи дані, порівняно з підходом, розроблений метод прискорює процес відновлення, тоді як відомий метод прискорює процес відновлення на порядок повільніше.

Додаткові витрати на синхронізацію невеликі в порівнянні з контрольними точками та часом відновлення. Таким чином, різниця в швидкості між розробленим і класичним підходами зменшується. Середня швидкість для цих двох підходів збільшується зі збільшенням набору даних. Основна причина цього полягає в тому, що дерева стають більшими, а підхід потребує більше часу для контрольної точки або відновлення її з диску.

Таким чином, здійснено розроблення методу для побудови системи відмовостійкості для підтримки алгоритму в паралельних системах. Запропоновано три алгоритми відмовостійкості: відмовостійкість на основі диску, відмовостійкість на основі синхронної пам'яті, асинхронна відмовостійкість на основі пам'яті. Алгоритм предстаявляє собою метод перебору для побудови системи відмовостійкості з використанням періодичних контрольних точок на диску. Однак два інші алгоритми, періодично виконують контрольні точки на пам'яті, а не на диску, щоб уникнути затримки введення-виведення.

Видалення транзакцій, які більше не потрібні, з пам'яті кожного вузла забезпечує достатній простір пам'яті для контрольних точок. В алгоритмі звужуємо простір оброблюваних транзакцій і виділяємо новий простір, до якого можуть віддалено отримати доступ інші процеси для виконання дерева та контрольної точки транзакцій. Цей алгоритм вимагає синхронізації між процесами перед будь-якою окремою контрольною точкою, що додає більше додаткових витрат на роботу контрольних точок. Однак, в алгоритмі використовуємо сам вектор транзакцій як простір контрольних точок, щоб уникнути будь-якого зв'язку між процесами під час операції контрольної точки. Запропоновані алгоритми гарантують складність простору для контрольних точок у пам'яті. Кожен процес використовує локальний простір пам'яті набору даних для перевірки дерева та часткових наборів даних іншого процесу. Також запропонували алгоритм адекватного відновлення, використовуючи контрольну точку транзакцій у пам'яті, яка дозволяє повторювати транзакції з пам'яті, а не з диску. Таким чином, у багатьох випадках повернення до нормального режиму рооти може бути завершено без доступу до диску.

У той час як алгоритм відновлення виконується тільки при несправностях, витрати несуть навіть при відсутності несправностей. Звичайно, дуже важливо мінімізувати час роботи на контрольно-пропускних пунктах, особливо коли рівень несправностей низький.

## 3.3 Висновки до третього розділу

Таким чином, здійснено розроблення методу для побудови системи відмовостійкості для підтримки алгоритму в паралельних системах. Запропоновано три алгоритми відмовостійкості: відмовостійкість на основі диску, відмовостійкість на основі синхронної пам'яті, асинхронна відмовостійкість на основі пам'яті.

# 4 ЗАБЕЗПЕЧЕННЯ ВІДМОВОСТІЙКОСТІ АЛГОРИТМІВ КЛАСИФІКАЦІЇ

## 4.1 Реалізація відмовостійкості алгоритму класифікації

Вплив апаратних збоїв у паралельних програмах, що вимагають великого обсягу даних, а також обмеження існуючих моделей паралельного програмування у вирішенні такої проблеми, є суттєвим. Крім того, технологія MPI була використана для розробки та реалізації відмовостійкої системи для загального алгоритму генератору шаблонів, тобто алгоритму генератору патернів. Запропоновані алгоритми відновлення контрольних точок використовують особливості технології одностороннього зв'язку MPI та паралельного алгоритму для побудови надійного алгоритму відмовостійкості.

Розглянемо проблему апаратних збоїв за допомогою іншого алгоритму інтелектуального аналізу даних. Запропоновано новий відмовостійкий алгоритм класифікації, який залежить від механізму відновлення контрольних точок. Використовуємо односторонній зв'язок MPI, щоб максимально перекрити контрольні точки, мінімізуючи додаткові витрати на розробку відмовостійкого алгоритму. Також проаналізуємо два різних алгоритми відновлення, які підходять для паралельного алгоритму

Алгоритм найближчих сусідів є широко використовуваним алгоритмом класифікації. Він відомий як найкращий алгоритм класифікації, якщо немає попередніх знань про розподіл даних. У ньому техніка контрольованого навчання використовується для вибору найближчих екземплярів даних для кожної вибірки. Кілька дослідників запропонували паралельні реалізації, щоб отримати переваги високопродуктивних обчислювальних систем.

Алгоритм показує базовий паралельний алгоритм для запропонованої відмовостійкої реалізації. Алгоритм має три входи: навчальний набір даних; тестовий набір даних; кількість найближчих сусідів. Алгоритм починається зі зчитування та розповсюдження навчальними та тестовими зразками з диска

майстер-вузлом. Для кожного тестового зразка визначається пріоритетна черга певної довжини. Для простоти припустимо кільцеву моду зв'язку між різними вузлами. Тепер виконується два основних етапи кожного процесу. Процес посилає власні навчальні зразки в наступний вузол кільця. Після цього процес обчислює відстань між локальною тестовою вибіркою та отриманою навчальною вибіркою. Ці два кроки повторюються певну кількість разів, так що кожен процес матиме вектор пріоритетних черг повного локального набору тестування. Крім того, узагальнено умовні позначення, які використовували для моделювання часової та просторової сумісності запропонованих відмовостійких алгоритмів. Запропонований алгоритм використовує функцію одностороннього зв'язку для приховування додаткових витрат на зв'язок шляхом перекриття зв'язку між різними процесами. Кожен процес зберігає пріоритети черг локальних вибірок даних до пам'яті іншого процесу, який потрібно отримати у разі збою. Проводимо поглиблену оцінку запропонованих підходів, використовуючи набір наборів даних машинного навчання в алгоритмі.

Розглядаємо запропоновані підходи як блок для проектування інших відмовостійких алгоритмів з огляду на ефективність запропонованих реалізацій. Оцінюємо запропоновані алгоритми відновлення контрольних точок з реалізаціями на основі ітерацій. Ефективність відмовостійкої імплементації перевершує реалізацію того ж алгоритму, забезпечуючи швидке прискорення. Систематична оцінка реалізацій у контексті розробки підходів, орієнтованих на застосунки, є необхідною. Запропонований відмовостійкий алгоритм є паралельним і відмовостійким алгоритмом. Ключова особливість такого підходу полягає в тому, що для виконання періодичних контрольних точок використовується односторонній зв'язок так що вартість контрольних точок повністю перекривається з обчисленнями. Виклик базового паралельного алгоритму з кожної вибірки з тестового набору даних має відповідний список найближчих сусідів у вигляді черги пріоритетів, який необхідно підтримувати протягом усього виконання. Алгоритм виконується для ітерацій, де вони також

задаються кількістю процесів, що беруть участь. У кожній ітерації черги пріоритетів, пов'язані з кожним тестовим зразком, оновлюються, коли знаходять сусіда. Отже, пріоритетні черги є критично важливою структурою даних, яка оновлюється і повинна бути збережена для ефективного відновлення. Використовуючи техніку MPI, кожен вузол зарезервував простір, який може обробляти контрольну точку для сусіднього вузла. Після обробки локальних тестових зразків повним контрольним пунктом виконується кожен з процесів. Кожен процес використовує односторонній зв'язок для зберігання своїх локальних пріоритетних черг на сусідньому вузлі.

Алгоритм ілюструє процедуру контрольної точки, а також відновлення. Під час процедури ініціалізації кожен процес має свій вектор, який містить локальний набір транзакцій. Кожен процес визначає один вектор, тобто метадані, які представляють набір параметрів для опису стану вектору і контрольних точок, що зберігаються в пам'яті. Вони використовуються для спільного використання обох векторів, тобто до інших процесів. Ці дії можуть бути позначені за допомогою процедури. Специфічно для цієї процедури, де кожен процес визначає свій процес. Для простоти припускаємо, що всі процеси спілкуються кільцево. Після цього процес віддалено перевіряє доступний простір на векторі, щоб вирішити, чи потрібно шукати лише контрольну точку свого набору.

Запропоновані два різні алгоритми відновлення. Як випливає з назви, в алгоритмі для відновлення та обробки зразків процесу, що зазнав невдачі, використовується лише один процес. Для порівняння, алгоритм використовує всі активні процеси, що залишилися, для обробки робочого навантаження невдалого процесу. Припускаючи, що збій відбувається з певним процесом майстер-процес зчитує зразки з диска і відправляє їх в процес відновлення. Процес резервного копіювання, який вже має пріоритетні черги відновлених зразків, призначає відновлені пріоритетні черги вилученим зразкам. У гіршому випадку складність часу відновлення пов'язана лише з вилученням зразків з

диска і обмежується.

В алгоритмі відновлення у процесі відновлення беруть участь усі активні вузли. У разі збою в процесі головний процес повторно зчитує зразки з диску і розподіляє їх між усіма доступними вузлами з супутньою вартістю. Об'єкт процесу, який має резервну копію пріоритетних черг, що генеруються в процесі, розподіляє пріоритетні черги між усіма доступними вузлами. При цьому всі активні вузли мають однакову кількість зразків для обробки. Процедура з алгоритму показує алгоритм відновлення. Припускаючи, що знову кільцевий зв'язок між процесами, то процес відновлення буде процесом поруч із мертвим процесом у кільці. Загалом, цей процес використовується для відновлення шляхом злиття останнього контрольного дерева, яке він має. Необроблені транзакції можуть бути відновлені з пам'яті процесу відновлення, якщо вона була перевірена до відмови або безпосередньо з диска. Відновлені транзакції перерозподіляються між доступними процесами, щоб додати їх до розподіляти відновлені пріоритетні черги між доступними вузлами. Хоча час відновлення алгоритму вищий, ніж алгоритму, то загальний час виконання алгоритму, що підтримується алгоритмом відновлення, є швидшим. Це пов'язано з тим, що алгоритм відновлення забезпечує балансування навантаження між різними вузлами навіть після відновлення.

Розглянемо оцінку ефективності. Для цього досліджуватимемо продуктивність запропонованих відмовостійких алгоритмів, включаючи їх контрольні точки та додаткові витрати на відновлення.

Для проведення експериментів використовуємо кластер, де кожен вузол має однакові технічні характеристики. Також використовуємо бібліотеку MPI, яка забезпечує високопродуктивну реалізацію MPI для оцінки продуктивності.

Для оцінки алгоритму використовували набір реальних наборів даних з репозиторію. У таблиці 4.1 наведено опис наборів даних.

Результати серії експериментів, призначених для оцінки продуктивності алгоритму є належними. Зокрема, оцінка так: додаткові витрати на контрольні

точки, коли використовується односторонній і двосторонній зв'язок; додаткові витрати на відновлення у випадку одностороннього і двостороннього зв'язку; відновлення одного процесу; відновлення більше одного процесу. Для оцінки використовуємо кілька наборів даних. Результати експериментів для кожного набору даних з використанням двох різних значень. Набори даних упорядковані за кількістю зразків у кожному наборі даних.

Таблиця 4.1 - Опис наборів даних

| Розмір тренувального набору | Розмір тестового набору |
|---|---|
| 2672 | 2347 |

Для більшості наборів даних продуктивність контрольних точок з використанням одностороннього зв'язку та двостороннього зв'язку є однаковою. У невеликих наборах даних, односторонній метод комунікації є швидшим, ніж двосторонній метод комунікації. Однак для великого набору даних обсяг контрольних точок даних приховує додаткові витрати на комунікацію при використанні двостороннього методу комунікації.

У випадках, коли відбувається збій пряме зчитування пріоритетних черг з пам'яті вузла відновлення при односторонньому зв'язку призводить до підвищення продуктивності. З меншими наборами даних спостерігається значне покращення. Зосереджуючись на алгоритмах відновлення показують величезне прискорення за допомогою методу. Обробка відновлених зразків лише одним процесом негативно впливає на баланс навантаження. Незважаючи на додаткові витрати на розподіл відновлених зразків між активними вузлами, він покращує загальний час виконання алгоритму.

Ефективність системи також можна показати, провівши ряд експериментів для порівняння її з реалізацією поданою в аналізі. У цих експериментах використовуємо два реальних набори даних для порівняння.

Продуктивність на основі MPI достатня. Час виконання алгоритму

контрольних точок у MPI набагато нижчий, ніж у реалізації з вбудованим механізмом контрольних точок. Різниця в часі виконання між алгоритмом на основі MPI і на основі стаціонарно вбудованого механізму контрольних точок збільшується зі збільшенням числа ядер. У разі збою час виконання нашого алгоритму відновлення зменшується зі збільшенням кількості ядер, тоді як аналізований алгоритм має меншу масштабованість.

Таким чином, створення системи відмовостійкості для підтримки алгоритмів інтелектуального аналізу даних є загальною темою для якої є використання розширених функцій MPI для забезпечення відмовостійкості. Розроблено по два різні методи для контрольних точок і відновлення. Ефективність відмовостійкої реалізації алгоритму перевершує реалізацію того ж алгоритму, забезпечуючи до багатократне прискорення.

## 4.2 Відмовостійкі опорні векторні машини

Було розглянуто проблему апаратних збоїв у двох поширених алгоритмах інтелектуального аналізу даних: алгоритмах класифікації за шаблоном зростання; алгоритмах класифікації найближчого сусіда. Використали технологію MPI для створення надійної системи відмовостійкості для обох алгоритмів.

Потрібно далі вирішити ту ж проблему, будуючи систему відмовостійкості для паралельного алгоритму опорних векторних машин (SVM). SVM — це клас алгоритмів контрольованого навчання, які максимізують межу відстані між спостереженнями, також відомими як вибірки, різних класів. SVM отримав широке поширення завдяки своїй високій точності та здатності обробляти набори даних з дуже великими розмірами. Реалізації SVM алгоритму по суті послідовної мінімальної оптимізації, який може використовувати багатоядерні системи та великомасштабні суперкомп'ютери, стали затребуваними і тому багато дослідників працюють в напрямі їх поліпшення.

Відомо, що набори даних з висококласними системами займають кілька годин. Паралельні системи, які можуть працювати на великих суперкомп'ютерах, стають звичним явищем. Однак ці суперкомп'ютери, розроблені в умовах обмежень руху даних, часто спостерігають збої в обчислювальних пристроях. Багато несправностей пристрою проявляються як постійні збої процесів чи вузлів.

Представимо кілька підходів до проектування відмовостійких алгоритмів. Здійснимо поглиблений аналіз для визначення критичних структур даних, які періодично перевіряються для побудови декількох базових алгоритмів. Далі розглянемо пропонований новий алгоритм, який не вимагає переміщення даних між вузлами для контрольних точок, SVM — це алгоритм контрольованого навчання, який будує класифікатор, що максимізує межу відстані між вибірками в навчальній множині. Розглядатимемо саме проблему бінарної класифікації в SVM, найпоширенішому випадку, коли набір даних має рівно два класи.

Основною метою SVM є генерація гіперплоскої функції між вибірками даних у класах. Завданням СВМ є знаходження оптимальної гіперплощини. Складовими є нормальний вектор, перпендикулярний до гіперплощини константа, та коефіцієнти зважування, які мають максимальну відстань з найближчим зразком. Тому, можна сформулювати оптимізаційну задачу. Використовуючи множники, ця задача може бути сформульована як задача оптимізації квадратичного програмування. Гіперпараметр, який визначеним користувачем, підтримує компроміс між точністю та загальністю класифікатора. Алгоритм SVM розв'язує задачу квадратичної оптимізації.

Задача квадратичної оптимізації може бути вирішена за допомогою різних методів, таких як: метод внутрішньої точки, метод активної множини або метод градієнта . Однак ці методи обмежені через їх складність простору та залежність від зовнішніх пакетів.

Для усунення цих обмежень запропоновано набір алгоритмів фрагментації. Зокрема, запропонований метод фрагментації, алгоритм

послідовної мінімальної оптимізації. Алгоритм SMO спрощує алгоритм навчання SVM, розбиваючи задачу SVM-QP на найменші оптимізаційні задачі, використовуючи рівно дві вибірки на кожному кроці. На кожній ітерації алгоритм вибирає дві вибірки, які можуть забезпечити швидку збіжність до рішення.

При цьому, кожна вибірка зберігає градієнт, який використовується для спуску до збіжності. На кожній ітерації градієнт вибірки оновлюється.

Вибір двох зразків, які використовуються в кожній ітерації, здійснюється за найменшим значенням з наявних. Зокрема, для оптимізації використовуються дві вибірки.

Зазвичай реалізація зберігає вибірки з найгіршими градієнтами в змінних, що відповідають максимальним і мінімальним значенням градієнта, відповідно.

Для аналізу розробки відмовостійких алгоритмів розглядаємо методики. Підходи, як правило, вимагають ідентифікації критичних структур даних і періодичного збереження їх на безпечному сховищі додатковій пам'яті на інших вузлах. У випадках, коли розмір критичних структур даних великий, обмежуючим фактором стає вартість переміщення даних для контрольних точок. Застосунок може або часто перевіряти критичні структури даних, що призводить до непомірного переміщення даних, або нечасто, тобто потенційно затримуючи час відновлення.

Пропонується кілька алгоритмів, які варіюються від періодичної перевірки критичних структур даних до алгоритму для контрольних точок, що повністю уникають комікування. Завдяки уникненню зв'язку запропонований алгоритм полегшує переміщення даних, одночасно досягаючи відмовостійкості, усуваючи основні конструктивні обмеження великомасштабних систем.

Спочатку застосовуємо базовий алгоритм, який визначає критичні структури даних і періодично перевіряє ці структури. Критичні структури даних вимагають просторової складності та переміщення даних для кожного кроку контрольної точки. Застосовуємо блокуючі та неблокуючі варіанти контрольних точок

базових алгоритмів.

Далі виконуємо відмовостійкий алгоритм для уникнення комунікації, який не вимагає міжвузлового переміщення даних для перевірки критичних структур даних. Запропонований алгоритм зберігає точність і значно покращує загальні вимоги до простору для контрольних точок. Проаналізовано алгоритм відновлення та показано, що він потребує вирішення проблеми часової складності. Відмовостійкий алгоритм підходить для великих наборів даних і великої кількості ядер.

Реалізуємо алгоритм уникнення комунікації, базові алгоритми та оцінюємо їх за допомогою кількох наборів даних класифікації за допомогою великомасштабного кластера. Порівнюємо ефективність таких підходів з обома системними методами перевірки точок.

Зроблена оцінка вказує на те, що загальний рух даних для контрольних точок у базовому алгоритмі може бути більшим за розмір набору даних, тоді як запропонований новий алгоритм є повністю вільним від контрольних точок. Також спостерігаємо, що додаткові витрати базового алгоритму відновлення сильно залежать від частоти спрацьовування контрольних точок, тоді як додаткові витрати алгоритму уникнення зв'язку є фіксованими. Низька частота контрольних точок потрібна для зменшення загального переміщення даних і призводить до середнього прискорення часу на ядрах для алгоритму уникнення зв'язку з різною кількістю контрольних точок.

Незважаючи на те, що рішення є специфічним для алгоритму, цей підхід є кроком у виконанні великомасштабних застосунків з інтенсивним використанням даних у системах екстремального масштабу.

Розглянемо послідовні і паралельні алгоритми, які не забезпечують відмовостійкості. Ці реалізації використовуються як основа для відмовостійких версій. Розглянемо умовні позначення, які використано для моделювання часової та просторової складності запропонованих алгоритмів

Алгоритм показує кроки послідовного алгоритму. Після того, як набір

даних завантажено в пам'ять, присвоюємо значення за замовчуванням градієнту кожного зразка. Індекси присвоюємо зразкам, які мають відповідно максимальне і мінімальне значення градієнта.

Ітераційний цикл оновлює значення, що відповідають їм. Градієнт для кожної вибірки оновлюється в результаті чого забезпечується часова складність на ітерації, і, нарешті, вибираються індекси для наступної ітерації, використовуючи оновлені значення. Алгоритм завершується, коли задовольняється критерій збіжності.

Другий алгоритм показує паралельну версію алгоритму, який використовували як базовий алгоритм для запропонованого відмовостійкого алгоритму. Первинна структура алгоритму аналогічна послідовному алгоритму, представленому раніше. Відмінності, які виникають у паралельному алгоритмі. На кожному кроці процес, які володіють зразками, що відповідають базовим, використовують MPI для трансляції цих зразків усім іншим процесам. Глобальні значення визначаються за допомогою MPI операцій.

Представимо кілька відмовостійких алгоритмів та їх застосування. Зокрема, спочатку визначаємо критичні структури даних, які оновлюються ітеративно, і використовуємо цю інформацію при розробці набору базових відмовостійких алгоритмів. Більш відмовостійкий алгоритм повністю уникає зв'язку для контрольних точок.

Розглядаємо початковий алгоритм як відправну точку для побудови базового відмовостійкого алгоритму. На кожній ітерації градієнт кожної вибірки оновлюється. Крім того, множники, що відповідають двом вибіркам, оновлюються на кожній ітерації. Множники інших зразків залишаються незмінними. Крім того, для кожної вибірки результат може змінюватися відповідно до умов. Інші структури даних, включаючи набір даних і мітку, доступні лише для читання та залишаються незмінними.

Розглянемо рівномірний розподіл зразків між процесами, такий, що кожен процес мітки неможливий. Під час відновлення вони можуть бути завантажені з

диска, і немає необхідності перевіряти їх. Однак, такі структури даних як градієнт тв множники є структурами даних читання-запису.

Базова відмовостійка версія вимагає перевіряти ці структури даних через регулярні проміжки часу. Звідси і просторова складність базового відмовостійкого алгоритму іншого процесу. Хоча диск можна використовувати, але загальний рух даних може значно збільшитися, особливо якщо диск розташований на віддаленому сервері. Тому, пропонуємо полегшити цю проблему, використовуючи додаткову пам'ять, розміщену іншими процесами.

Наступний алгоритм показує кроки базового відмовостійкого алгоритму. У поєднанні з базовим алгоритмом додаткові виділені кроки також вказують на два варіанти контрольних точок, тобто на блокування та неблокування, щоб зменшити часову складність кроку контрольної точки. Природно, що при неблокуючому очікуємо, що загальний час пропуску буде незначним. Однак загальна складність базових алгоритмів і загальний рух даних залишається незмінним.

Удосконалення просторово-часової складності базового алгоритму для кожної вибірки можна обчислити за допомогою базового рівняння. У зв'язку з цим, потрібно перерахувати значення під час відновлення, що зменшує вартість, вказуючи на складність простору та обсяг комунікації для градієнта та множників для кожного процесу.

Базовий алгоритм відновлення відмовостійкого алгоритму може допускати збій процесу під час виконання алгоритму. Розглянемо майстер-процес у реалізації процесу із рангом MPI за замовчуванням. Тут може бути невдалий або вимкнений процес і тіньовий процес, який володіє збереженими контрольними точками. Для відновлення знадобиться кілька структур даних таких як набір даних тільки для читання з диска і раніше збережені копії критичних структур даних.

Після виявлення помилки процесу слід відновити пов'язані з ним структури даних. Критичні структури даних розподіляються тіньовим процесом

між іншими процесами. Перерозподіл критичних структур даних вимагає трансляції операції з рештою процесів, яка має часову складність. Крім того, зразки наборів даних можуть бути зчитані з диска двома різними способами. Зразки наборів даних можуть бути зчитані з диска головним процесом і рівномірно розподілені між іншими процесами. Однак у цьому випадку доступ до диска буде найдорожчою частиною загального алгоритму відновлення, який очікує зайняти час. Отже, він відмовостійкий для уникнення комунікації.

Кроки методу:

1) здійснити паралельний запит кожним процесом даних із диску;

2) визначити час відправки запиту та час відповіді на нього;

3) встановити затримки при роботі з великими даними;

4) розподілити виконання завдань через відповідні процеси;

5) отримати збережені копії результатів;

6) виконати п.1) повторно;

7) прийняти рышення про завершення.

Приклад опрацювання процесів за цим алгоритмом подано на рис. 4.1.

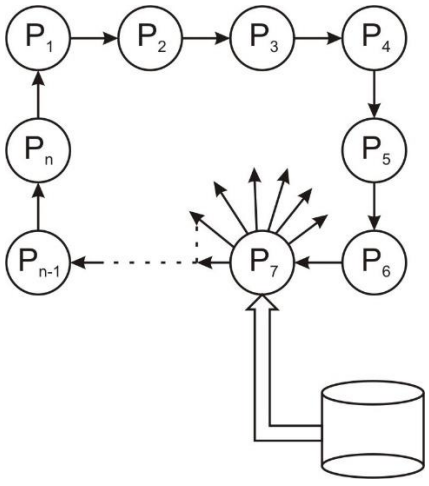



Рисунок 4.1 - Приклад опрацювання процесів

За результатами реалізації методу отримані часові значення для виконання процесів без застосування методу та із застосуванням методу. Вони подані в таблиці 4.2.

Таблиця 4.2 – Результати застсоування методу щодо часових затримок

| Номер процесу | Час для звичайного режиму, нс | Час при застосуванні методу, нс |
|---|---|---|
| 1 | 23 | 18 |
| 2 | 34 | 25 |
| 3 | 109 | 27 |
| 4 | 167 | 23 |
| 5 | 37 | 11 |
| Середнє значення: | 74 | 20,8 |

Таким чином, застосування розробленого методу забезпечує роботу застосунків з великими даними приблизно у 2,5 рази швидше.

Запропонований алгоритм для уникнення зв'язку має вплив на здатність проводити аналіз великих обсягів даних на великомасштабних системах в умовах обмежень переміщення даних і несправностей.

## 4.3 Висновки до четвертого розділу

Програми обробки на основі пам'яті повинні підтримуватися ефективною відмовостійкою системою, щоб отримати більшу продуктивність від цих програм. Було використано набір алгоритмів інтелектуального аналізу даних і машинного навчання, які були реалізовані за допомогою бібліотеки інтерфейсу передачі повідомлень MPI. В результаті отримані розширені функції цільових алгоритмів та односторонній зв'язок MPI для створення швидких та ефективних відмовостійких систем для кількох алгоритмів. Для цього використовували неблокувальні функції MPI, щоб приховати додаткові витрати на зв'язок «точка-точка», необхідний для контрольно-пропускних пунктів і операцій відновлення.

# ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень розроблено новий метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних та отримано такі результати.

1. Проаналізовано відомі методи забезпечення пропускної здатності дисків для застосунків та промлемні завдання, які виникають при їх реалізації.

2. Розроблено новий метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних. Використано растрові зображення для покращення масштабованості та поганих проблем балансування навантаження в існуючих алгоритмах просторового з'єднання. Також, розглянуто проблеми відмовостійкості в існуючих алгоритмах машинного навчання та інтелектуального аналізу даних. Розроблено алгоритм відмовостійкості на основі пам'яті, який порівняно з дисковими підходами, має особливості, які забезпечують методи контрольної точки та відновлення даних.

3. Здійснено реалізацію розробленого методу забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних. Існуючі алгоритми просторового об'єднання поділяються на два основних типи: алгоритми просторового об'єднання для роботи з малими наборами даних; алгоритми розбиття для роботи з великими наборами даних. В результаті було запропоновано підхід для прискорення процесу запиту просторового об'єднання в пам'яті, надавши техніку апроксимації для кроку базового фільтра за допомогою індексування растрових зображень.

4. Здійснено есперименталні дослідження згідно розроблених рішень.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Antoine E., Ramamohanarao K., Shao J., Zhang R. Accelerating spa- tial join operations using bit-indices. In *Proceedings of the Twenty-Second Australasian Database conference*-Australian Computer Society, Inc., 2011. Volume 115, P. 123–132.

2. Aparício G., Blanquer I., Hernández V. A parallel implementation of the k nearest neighbours classifier in three levels: Threads, mpi processes and the grid. In *High Performance Computing for Computational Science-VECPAR 2006*, Springer, 2007. P. 225–235.

3. Berkeley earth. http://www.berkeleyearth.org. [Online; accessed 12-January-2015].

4. Bitmap index vs. b-tree index: Which and when? http://www.oracle.com/ technetwork/articles/sharma-indexes-093638.html. [Online; accessed 1- January-2015].

5. ExaScale Software Study: Software Challenges in Extreme Scale Systems . In *DARPA Re- view*, 2009.

6. Agrawal R., Srikant R. Quest synthetic data generator. ibm almaden research center, sanjose, california, 2009.

7. Akidau T., Balikov A., Bekiroğlu K., Chernyak S., Haberman J., Lax R., McVeety S., Mills D., Nordstrom P., Whittle S. Millwheel: Fault-tolerant stream processing at internet scale. *Proc. VLDB Endow.*, 6(11):1033–1044, August 2013.

8. Arefin A. S., Riveros C., Berretta R., Moscato P. Gpu-fs-k nn:A software tool for fast and scalable k nn computation using gpus. 2012.

9. Batista G., Monard M. An analysis of four missing data treat-ment methods for supervised learning. *Applied Artificial Intelligence*, 2003. V. 17(5-6), P. 519–533.

10. Besta M., Hoefler T. Fault tolerance for remote memory access programmingmodels. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, ACM, 2014. P. 37–48.

11. J. Daily, A. Vishnu, B. Palmer, H. van Dam, and D. Kerbyson. On the suitability of mpi as a pgas runtime. In High Performance Computing (HiPC), 2014 21st International Conference on, pages 1–10, Dec 2014.

12. Bhokare P., Sharma R. A novel algorithm pda (parallel and distributed apriori) for frequent pattern mining. International Journal of Engineering, 3(8), 2014.

13. Borgelt C. An implementation of the fp-growth algorithm. In Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementa- tions, pages 1–5. ACM, 2005.

14. Bronevetsky G., Supinski B. Soft error vulnerability of iterative linear algebra methods. In Proceedings of the 22Nd Annual International Conference on Supercomputing, ICS ’08, 2008.

15. Bronevetsky G., Marques D., Pingali K., Stodghill P. Automated application-level checkpointing of mpi programs. ACM Sigplan Notices, 38(10):84–94, 2003.

16. Buehrer G., Parthasarathy S., Ghoting A. Out-of-core frequent pat- tern mining on a commodity pc. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 86–95. ACM, 2006.

17. Buehrer G., Parthasarathy S., Tatikonda S., Kurc T., Saltz J. Toward terabyte pattern mining: an architecture-conscious solution. In Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming, PPoPP ’07, pages 2–12, 2007.

18. Buehrer G., Parthasarathy S., Tatikonda S., Kurc T., Saltz J. Toward terabyte pattern mining: an architecture-conscious solution. In Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming, pages 2–12. ACM, 2007.

19. Canahuate G., Gibas M., Ferhatosmanoglu H. Indexing incomplete databases. In Advances in Database Technology-EDBT 2006, pages 884–901. Springer, 2006.

20. Cao L. J., Keerthi S. S., Ong C.-J., Zhang J. Q., Periyathamby U., Fu X. J.,

Lee H. P. Parallel sequential minimal optimization for the training of support vector machines. IEEE Transactions on Neural Networks, 17(4):1039–1049, July 2006.

21. Cappello F., Geist A., Gropp B., Kale L., Kramer B., Snir M. Towardexascale resilience. *International Journal of High Performance Computing Applications*, 2009.

22. Catanzaro B., Sundaram N., Keutzer K. Fast support vector machine training and classification on graphics processors. In *Proceedings of the 25th international conference on Machine Learning*, ICML '08, ACM, 2008. P. 104–111.

23. Chang E. Y., Zhu K., Wang H., Bai H., Li J., Qiu Z., Cui H. Psvm: Parallelizing support vector machines on distributed computers. In *NIPS*, 2007. Software available at http://code.google.com/p/psvm.

24. Chen Z. Algorithm-based recovery for iterative methods without checkpointing. In *Proceedings of the 20th international symposium on High performance distributed comput-ing*, ACM, 2011. P. 73–84.

25. Chou J., Wu K., Rubel O, Mark Howison, Ji Qiang, Brian Austin, E Wes Bethel, Rob D Ryne, Arie Shoshani, et al. Parallel index and query for large scale data analysis. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 InternationalConference for*, IEEE, 2011. P. 1–11.

26. Cios K., Kurgan L. Trends in data mining and knowledge discovery. In *Advanced techniques in knowledge discovery and data mining*, pages 1–26. Springer, 2005.

27. Coti C., Herault T. Pierre Lemarinier, Laurence Pilard, Ala Rezmerita, Eric Ro- driguez, and Franck Cappello. Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant mpi. In *Proceedings of the 2006 ACM/IEEE conference on Super-computing*, ACM, 2006. P. 127.

28. Cotter A., Srebro N., Keshet J. A GPU-tailored approach for training kernelized SVMs. In *Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining*, KDD '11, 2011. P. 805–813.

29. Cunningham D., Grove D., Herta B., Iyengar A., Kawachiya K., Murata Hiroki, Saraswat V., Takeuchi M., Tardieu O. Resilient x10: Efficient failure-aware

programming. *SIGPLAN Not.*, 49(8):67–80, February 2014.

30. Deliège F., Pedersen T. Position list word aligned hybrid: optimizing space and performance for compressed bitmaps. In *Proceedings of the 13th international conference on Extending Database Technology*, ACM, 2010. P. 228–239.

31. James D., Larkins D., Sadayappan P., Krishnamoorthy S., Nieplocha J. Scalable work stealing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ACM, 2009. P. 53.

32. Egwutuoha I., Levy D., Selic B., Chen S. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *The Journal of Supercomputing*, 2013. V. 65(3), P. 1302–1326.

33. Fang W., Lu M., Xiao X., He B., Luo Q. Frequent itemset mining on graphics processors. In *Proceedings of the Fifth International Workshop on Data Management on New Hardware*, DaMoN '09, 2009. P. 34–42.

34. Freund R. M. Solution methods for quadratic optimization. Technical report, Technicalreport, Massachusetts Institute of Technology, MA, 2004.

35. Gaur S., Dulawat M. A closest fit approach to missing attribute values in data mining. *International Journal of advances in Science and Technology*, 2011. V. 2(4). P. 18–24.

36. Gouda K., Zaki M. Genmax: An efficient algorithm for mining maximal frequent itemsets. *Data Mining and Knowledge Discovery*, 2005. V. 11(3). P. 223–242,

37. Grzymala-Busse J., Goodwin L., Grzymala-Busse W., Zheng X. Handling missing attribute values in preterm birth data sets. In *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, Springer, 2005. P. 342–351.

38. Grzymala-Busse J., Hu M. A comparison of several approaches to missing attribute values in data mining. In *Rough sets and current trends in computing*, Springer, 2001. P. 378–385.

39. Hadjieleftheriou M., Hoel E., Tsotras V. J. Sail: A spatial index library for efficient application integration. *Geoinformatica*, 2005. V. 9(4). P. 367–389,

40. Han J., Pei J., Yin Y. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, ACM, 2000. P. 1–12.

41. Hargrove P., Duell J. Berkeley lab checkpoint/restart (blcr) for linux clusters. *Journal of Physics: Conference Series*, 2006. V. 46(1). P. 494.

42. He B., Hsiao H.-I., Liu Z., Huang Y., Chen Y. Efficient iceberg query evaluation using compressed bitmap index. *Knowledge and Data Engineering, IEEE Transactions on*, 2012. V. 24(9). P. 1570–1583.

43. Heinis T., Tauheed F., Ailamaki A. Spatial data management chal- lenges in the simulation sciences. In *Proceedings of the international conference on Extend-ing Database Technology*, number EPFL-CONF-190522, 2014.

44. Hoefler T., Lumsdaine A., Dongarra J. Towards efficient mapreduce using mpi. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Springer, 2009. P. 240–249.

45. Hsieh C.-J., Chang K.-W., Lin C.-J., Keerthi S., Sundararajan S. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 408–415, New York, NY, USA, 2008. ACM.

46. Hursey J., Squyres J., Mattox T., Lumsdaine A. The design and implementation of checkpoint/restart process fault tolerance for open mpi. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, IEEE, 2007. P. 1–8.

47. Zaharia M., Chowdhury M., Das T., Dave A., Ma J., Mc- Cauley M., Franklin M., Shenker S., Stoica I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, Berkeley, CA, USA, 2012. USENIX Association. P. 2–2.

48. Zhang C., Li F., Jestes J. Efficient parallel knn joins for large data in mapre-duce. In *Proceedings of the 15th International Conference on Extending Database Technol-ogy*, ACM, 2012. P. 38–49.

49. Zhang H., Chen G., Ooi B., Tan K.-L., Zhang M. In-memory big data management and processing: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 2015. V. 27(7). P. 1920–1948.

50. Zhang Q., Li C., He P., Li X., Zou H. Irregular partitioning method based k-nearest neighbor query algorithm using mapreduce. In *2015 International Symposium on Computers & Informatics*. Atlantis Press, 2015.

51. Zhang S., Han J., Liu Z., Wang K., Xu Z. Sjmr: Parallelizing spatial join with mapreduce on clusters. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE international conference on*, IEEE, 2009. P. 1–8.

52. Zhou L., Zhong Z., Chang J., Li J., Huang J.Z., Feng S. Balancedparallel fp-growth with mapreduce. In *Information Computing and Telecommunications (YC- ICT), 2010 IEEE Youth Conference on*, IEEE, 2010. P. 243–246.

53. Zhu M., Papadias D., Lee D. L., Zhang J. Top-k spatial joins. *Knowledgeand Data Engineering, IEEE Transactions on*, 2005. V. 17(4). P. 567–579.

54. Ian H. *An introduction to geographical information systems*. Pearson Education India, 2010.

55. Itkar S. A., Kulkarni U. V. Distributed algorithm for frequent pattern mining using hadoopmap reduce framework. *International Conference on Advances in Computer Science, AETACS*, 2013.

56. Jacox E. H., Samet H. Spatial join techniques. *ACM Transactions on DatabaseSystems (TODS)*, 2007. V. 32(1). P. 7.

57. Joachims T. Making large-scale support vector machine learning practical. In Bern- hard Schölkopf, Christopher J. C. Burges, Alexander J. Smola, editors, *Advances in kernel methods*, MIT Press, 1999. P. 169–184.

58. Keerthi S. S., Shevade S. K., Bhattacharyya C., Murthy K. R. K. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 2001. V. 13(3). P. 637–649.

59. Khalefa M. E., Mokbel M. F., Levandoski J. J. Skyline query process-ing for incomplete data. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International*

*Conference on*, IEEE, 2008. P. 556–565.

60. Koudas N. Space efficient bitmap indexing. In *Proceedings of the ninth international conference on Information and knowledge management*, ACM, 2000. P. 194–201.

61. Kumar K., Laxmaiah M., Kumar C. A compacted bitmap vector technique toevaluate iceberg queries efficiently. *International Journal*, 2013. V. 3(6). P. 412–418.

62. Li D., Deogun J., Spaulding W., Shuart B. Towards missing data imputation: A study of fuzzy k-means clustering method. In *Rough Sets and Current Trends in Computing*, Springer, 2004. P. 573–579.

63. Vishnu A., Agarwal K. Ieee cluster. chapter Large Scale Frequent Pattern Mining using MPI One-sided Model. 2015.

64. Haoyuan L., Wang Y., Zhang D., Zhang M., Chang E. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*, ACM, 2008. P. 107–114.

65. Yinan L., Patel J. Bitweaving: fast scans for main memory data processing. In*Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ACM, 2013. P. 289–300.

66. Lin W.-T., Chu C.-P. Determining the appropriate number of nodes for fast mining of frequent patterns in distributed computing environments. *International Journal ofParallel, Emergent and Distributed Systems*, (ahead-of-print):1–13, 2014.

67. Lisboa C., Argyrides C., Pradhan D., Carro L. Algorithm level fault tolerance: a technique to cope with long duration transient faults in matrix multiplication algorithms. In *VLSI Test Symposium, 2008. VTS 2008. 26th IEEE*, pages 363–370. IEEE, 2008.

68. Lei L. A review of missing data treatment methods. *International journal of intelligent information systems and Tech*, 2005. P. 412–419.

69. Lu L., Shi X., Zhou Y., Zhang X., Jin H., Pei C., He L., Geng Y. Lifetime-based memory management for distributed data processing sys- tems. *arXiv preprint*

*arXiv:1602.01959*, 2016.

70. Luengo J., García S., Herrera F. On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowledge and information systems*, 2012. V. 32(1). P. 77–108.

71. Luo G., Naughton J. F, Ellmann C. J. A non-blocking parallel spatial join algorithm. In *Data Engineering, 2002. Proceedings. 18th international conference on*, IEEE, 2002. P. 697–705.

72. Mamoulis N., Papadias D., Arkoumanis D. Complex spatial query process- ing. *GeoInformatica*, 8(4):311–346, 2004.

73. Manyika J., Chui M., Brown B., Bughin J., Dobbs R., Roxburgh C., Byers A. H. Big data: The next frontier for innovation, competition, and productivity. 2011.MaTEx. Machine Learning Toolkit for Extreme Scale. http://hpc.pnl.gov/matex.

74. Meng X., Bradley J., Yavuz B., Sparks E., Venkataraman S., Liu D., Freeman J., Tsai D., Amde M., Owen S. Mllib: Machine learning in apache spark. *arXiv preprint arXiv:1505.06807*, 2015.

75. Modgi P., Vaghela D. Mining distributed frequent itemset with hadoop. *International Journal of Computer Science & Information Technologies*, 5(3), 2014.

76. Moody A., Bronevetsky G., Mohror K., Supinski B. Detailed modeling, design, and evaluation of a scalable multi-level checkpointing system. *Lawrence Livermore National Laboratory (LLNL), Tech. Rep. LLNL-TR-440491*, 2010.

77. Moon L., Long D., Joshi S., Tripath V., Xiao B., Biros G. Parallel algorithms for clustering and nearest neighbor search problems in high dimensions, in 2011 acm. In *IEEE conference on Supercomputing, Poster Session, Piscataway, NJ, USA*, 2011.

78. Richard M. The office of science data-management challenge. report from the doe office of science data-management workshops. Technical report, Technical Report SLAC, 2004.

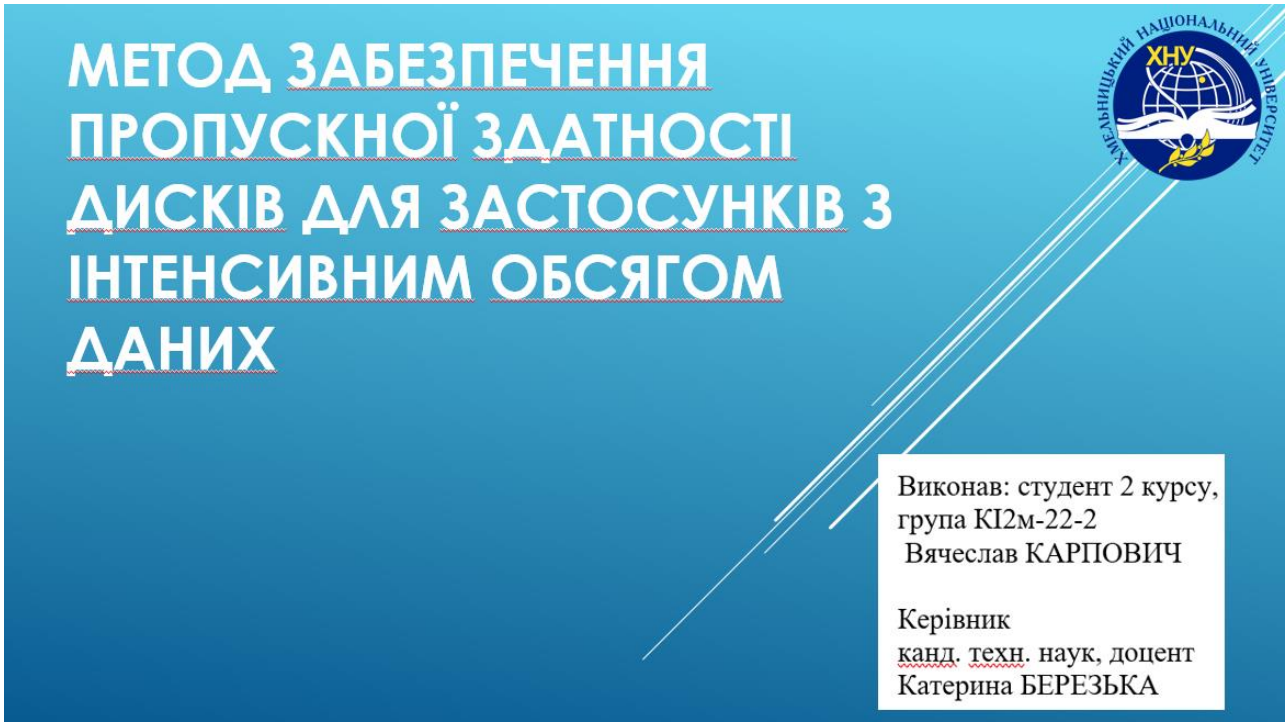
79. Nasser T., Tariq R. Big data challenges. *J Comput Eng Inf Technol 4: 3. doi: http://dx.doi. org/10.4172/2324*, 9307:2, 2015.

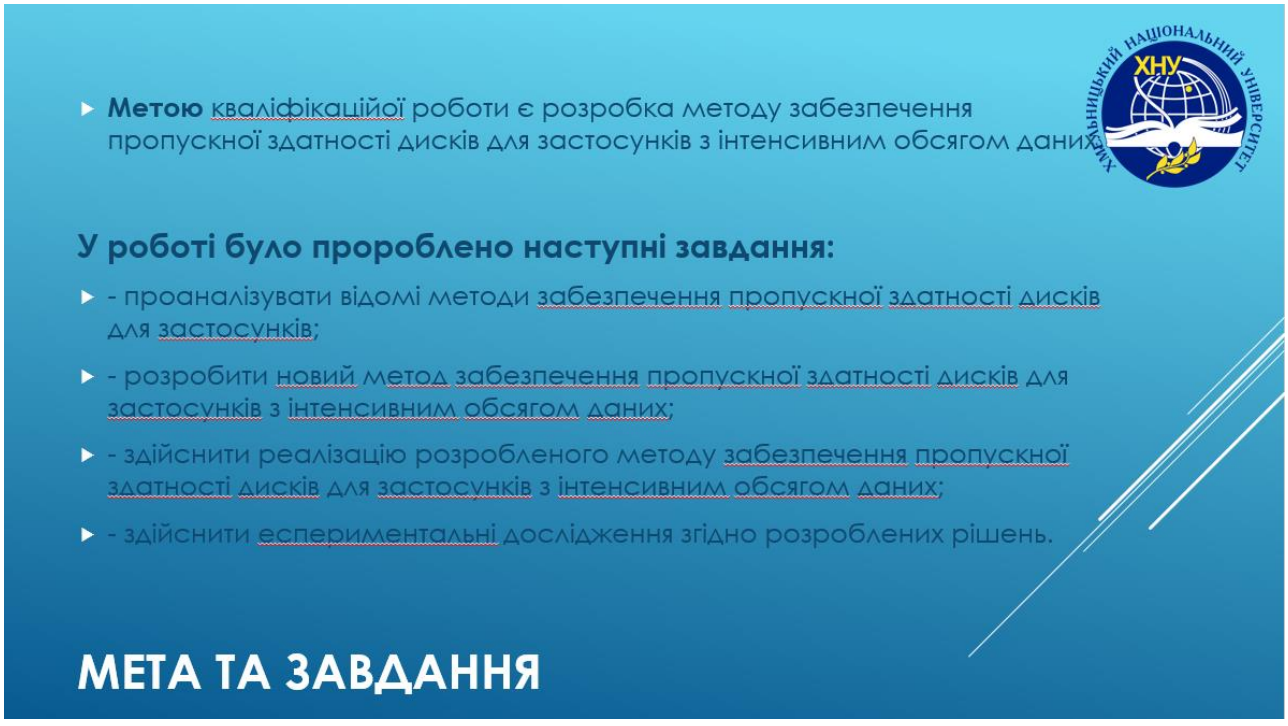
80. Nobari S., Tauheed F., Heinis T., Karras P., Bressan S. Ailamaki and

Anastasia. Touch: in-memory spatial join by hierarchical data-oriented partition- ing. In *Proceedings of the 2013 international conference on Management of data*, pages 701–712. ACM, 2013.

81. Карпович В.В., Дрозд А.І., Жуковський П.О., Мельник В.В. Методи вирішення проблем пропускної здатності дисків для застосунків з інтенсивним обсягом даних / Збірник наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2023». Хмельницький, 2023, С. 116-117. https://kn.khmnu.edu.ua/wp-content/uploads/sites/18/apkn-2023-corpuspaper.pdf

**ДОДАТОК А** Презентація роботи



- Об'єктом дослідження є процес забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.
- Предметом дослідження є методи забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.
- Наукова новизна отриманих результатів:

- розроблено новий метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

- Практична значимість отриманих результатів полягає у розроблених алгоритмах для забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних.

## ОБ'ЄКТ, ПРЕДМЕТ

Розроблені застосунки з великим обсягом даних для вирішення різних проблем і обмежень, пов'язаних з поточним вибухом даних, використовуються для зберігання, аналізу та управління існуючими великими наборами даних. Одним з основних недоліків існуючих застосунків з великим обсягом даних є відносно повільний доступ до диска. Програми обробки на основі дисків не можуть запропонувати відповідний час відгуку через затримку доступу до дисків. Таким чином, ефективні застосування з інтенсивним використанням даних повинні підтримуватися обробкою на основі пам'яті, а не дисковою обробкою, щоб отримати більшу продуктивність від цих програм.

## АКТУАЛЬНІСТЬ РОБОТИ

Існуючі алгоритми просторового об'єднання поділяються на два основних типи: алгоритми просторового об'єднання для роботи з малими наборами даних; алгоритми розбиття для роботи з великими наборами даних. В результаті було запропоновано підхід для прискорення процесу запиту просторового об'єднання в пам'яті, надавши техніку апроксимації для кроку базового фільтра за допомогою індексування растрових зображень. Растрові зображення групують набір різних значень точок об'єктів в одне значення, щоб зменшити загальну кількість порівнянь в операціях з'єднання. Існуючі підходи до просторового поділу страждають від поганого балансування навантаження у випадку паралельного процесу. Тому, було розроблено підхід на основі растрових зображень, який може автоматично рівномірно розподіляти просторові об'єкти в різних розділах.

## ПЕРЕВАГИ ТА НЕДОЛІКИ РІШЕНЬ

Концепція «великих даних» виникла у зв'язку з нещодавнім вибухом обсягів даних та розбіжностями, пов'язаними з аналізом, управлінням та обробкою великих обсягів даних. Паралельні застосунки інтенсивним використанням даних були спроектовані та розроблені в різних областях для роботи з великими обсягами даних, які існують в даний час. Однак ці програми мають ряд недоліків і обмежень, які негативно впливають на їх функціональність. Перспективними для дослідження є дві проблеми паралельних застосунків з інтенсивним використанням даних: повільний час читання диска в порівнянні з продуктивністю сучасних процесорів; більш висока частота відмов при великій кількості вузлів і тривалому часу виконання. При цьому ця робота спрямована на забезпечення двох основних вимог розробку нових методів усунення вузького місця дискового вводу/виводу; розробку ефективних відмовостійких алгоритмів для обробки апаратних несправностей у паралельних застосунках.

## АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Кроки методу:

1) здійснити паралельний запит кожним процесом даних із диску;

2) визначити час відправки запиту та час відповіді на нього;

3) встановити затримки при роботі з великими даними;

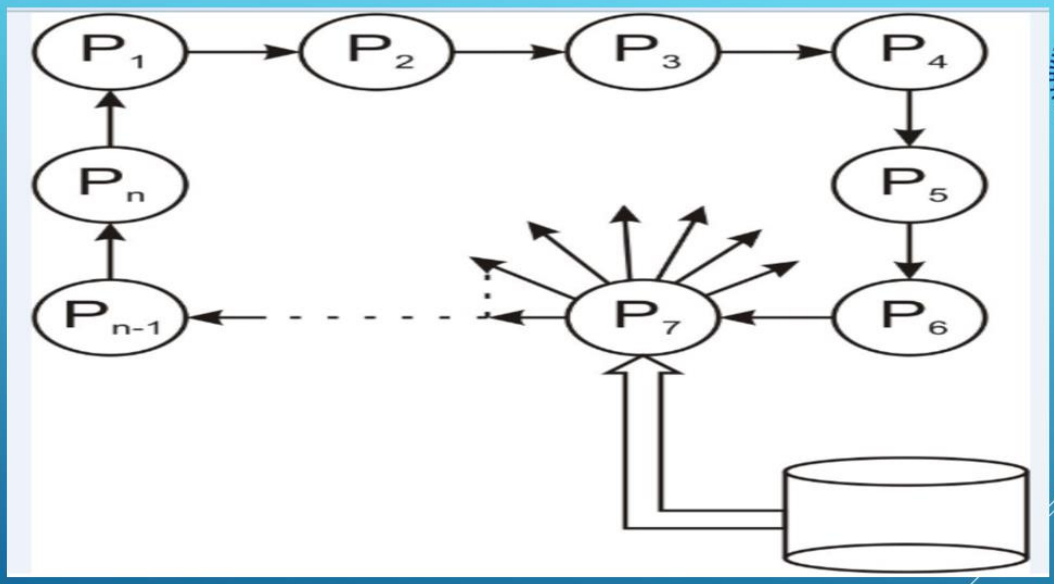
4) розподілити виконання завдань через відповідні процеси;

5) отримати збережені копії результатів;

6) виконати п.1) повторно;

7) прийняти рышення про завершення.

## МЕТОД




## ПРИКЛАД ОПРАЦЮВАННЯ ПРОЦЕСІВ

| Номер процесу | Час для звичайного режиму, нс | Час при застосуванні методу, нс |
|---|---|---|
| 1 | 23 | 18 |
| 2 | 34 | 25 |
| 3 | 109 | 27 |
| 4 | 167 | 23 |
| 5 | 37 | 11 |
| Середнє значення: | 74 | 20,8 |

## РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТУ

## ВИСНОВКИ

- У роботі за результатами виконаних теоретичних та практичних досліджень розроблено новий метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних та отримано такі результати.
- 1. Проаналізовано відомі методи забезпечення пропускної здатності дисків для застосунків та промлемні завдання, які виникають при їх реалізації.
- 2. Розроблено новий метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних. Використано растрові зображення для покращення масштабованості та поганих проблем балансування навантаження в існуючих алгоритмах просторового з'єднання. Також, розглянуто проблеми відмовостійкості в існуючих алгоритмах машинного навчання та інтелектуального аналізу даних. Розроблено алгоритм відмовостійкості на основі пам'яті, який порівняно з дисковими підходами, має особливості, які забезпечують методи контрольної точки та відновлення даних.
- 3. Здійснено реалізацію розробленого методу забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних. Існуючі алгоритми просторового об'єднання поділяються на два основних типи: алгоритми просторового об'єднання для роботи з малими наборами даних; алгоритми розбиття для роботи з великими наборами даних. В результаті було запропоновано підхід для прискорення процесу запиту просторового об'єднання в пам'яті, надавши техніку апроксимації для кроку базового фільтра за допомогою індексування растрових зображень.
- 4. Здійснено експериментальні дослідження згідно розроблених рішень.

- Карпович В.В., Дрозд А.І., Жуковський П.О., Мельник В.В. Методи вирішення проблем пропускної здатності дисків для застосунків з інтенсивним обсягом даних / Збірник наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2023». Хмельницький, 2023, С. 116-117. https://kn.khmnu.edu.ua/wp-content/uploads/sites/18/apkn-2023-corpuspaper.pdf

## ПУБЛІКАЦІЯ

**ДОДАТОК Б** Наукова праця здобувача



УДК 004.7

Карпович В.В., Дрозд А.І., Жуковський П.О., Мельник В.В.

*Хмельницький національний університет*

## МЕТОДИ ВИРІШЕННЯ ПРОБЛЕМ ПРОПУСКНОЇ ЗДАТНОСТІ ДИСКІВ ДЛЯ ЗАСТОСУНКІВ З ІНТЕНСИВНИМ ОБСЯГОМ ДАНИХ

*Розглянуто проблеми з пропускною здатністю дисків для застосунків з інтенсивним обсягом даних.*

*В роботі пропонується для підвищення ефективності таких операцій, що інтенсивно працюють із даними, розробити нові методи роботи в пам'яті, окрім вирішення нових проблем, пов'язаних із програмами обробки на основі пам'яті, такими як стійкість до відмов і узгодженість. Варіантом вирішення проблеми є обмеження відмовостійкості, щоб дозволити різним програмам із інтенсивним використанням даних продовжувати працювати належним чином у разі збою.*

*Addresses disk bandwidth issues for data-intensive applications.*

*The paper proposes to develop new in-memory techniques to improve the efficiency of such data-intensive operations, in addition to addressing new challenges associated with memory-based processing programs, such as fault tolerance and consistency. A workaround is to limit fault tolerance to allow various data-intensive applications to continue to function properly in the event of a failure.*

Сьогодні «великі дані» стають загальною темою в різних сферах. Таким чином, багато даних застосунків були розроблені для вирішення різних викликів і пов'язаних обмежень з поточним зростанням даних [1-3]. Ці програми використовуються для зберігання, аналізу та керування існуючими великими наборами даних. Одним із головних недоліків [4, 5] існуючих застосунків із інтенсивним використанням даних є відносно повільний доступ до диска. Дискові програми обробки не можуть запропонувати адекватний відповідний час відповіді через затримку доступу до дисків. Таким чином, ефективні програми з інтенсивним використанням даних повинні підтримуватися обробкою на основі пам'яті замість обробки на основі диска, щоб отримати більшу продуктивність цих програм.

Перспективним напрямом вирішення такої проблеми може бути покращення довгострокових програм, що інтенсивно працюють із даними, шляхом розробки нових методів роботи в пам'яті, окрім вирішення нових проблем, пов'язаних із програмами обробки на основі пам'яті, такими як стійкість до відмов і

узгодженість. Варіантом вирішення проблеми, також, є обмеження відмовостійкості, щоб дозволити різним програмам із інтенсивним використанням даних продовжувати працювати належним чином у разі збою.

Незважаючи на те, що доступ до даних пам'яті є надзвичайно швидким порівняно з доступом до диска, наявна ефективна техніка індексування для підтримки швидких запитів допомагає уникнути інтенсивного сканування пам'яті.

Для вирішення цього завдання розробимо метод індексування, щоб покращити дві складні операції, пов'язані з великими наборами даних – пропуск відсутніх значень і запит просторового з'єднання через створення підходів на основі пам'яті з підтримкою індексування, які можуть обробляти великі обсяги даних.

Застосування розробленого методу дозволить покращити роботу з інтенсивними даними при їх великих обсягах.

Підтвердження результатів роботи здійснено через імплементацію розробленого методу в застосунках і дослідження дискового простору, часу звернень і зміни обсягу.

**Перелік посилань**

1. Ifeanyi P Egwutuoha, David Levy, Bran Selic, and Shiping Chen. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. The Journal of Supercomputing, 65(3):1302–1326, 2013.
2. Thomas Heinis, Farhan Tauheed, and Anastasia Ailamaki. Spatial data management challenges in the simulation sciences. In Proceedings of the international conference on Extending Database Technology, number EPFL-CONF-190522, 2014.
3. Suhasini A Itkar and Uday V Kulkarni. Distributed algorithm for frequent pattern mining using hadoopmap reduce framework. International Conference on Advances in Computer Science, AETACS, 2013.
4. K Sunil Kumar, M Laxmaiah, and C Sunil Kumar. A compacted bitmap vector technique to evaluate iceberg queries efficiently. International Journal, 3(6):412–418, 2013.
5. Wei-Tee Lin and Chih-Ping Chu. Determining the appropriate number of nodes for fast mining of frequent patterns in distributed computing environments. International Journal of Parallel, Emergent and Distributed Systems, (ahead-of-print):1–13, 2014.

UNICHECK
by Turnitin

Ім'я користувача:
**Кафедра КІ**

ID перевірки:
**1016230136**

Дата перевірки:
**06.05.2024 08:55:39 EEST**

Тип перевірки:
**Doc vs Internet + Library**

Дата звіту:
**06.05.2024 08:57:24 EEST**

ID користувача:
**100005591**

---

Назва документа: **Карпович_Метод забезпечення пропускної здатності дисків для застосунків з інтенсивним...**

Кількість сторінок: **86** Кількість слів: **20851** Кількість символів: **163589** Розмір файлу: **263.75 KB** ID файлу: **1016010041**

---

# 11.8%
# Схожість

Найбільша схожість: **2.14%** з Інтернет-джерелом (https://mafiadoc.com/fault-tolerant-frequent-pattern-mining_5cc8bb...

| | | |
|---|---|---|
| 11.2% Джерела з Інтернету | 901 | Сторінка 88 |
| 1.59% Джерела з Бібліотеки | 61 | Сторінка 103 |

# 0.08% Цитат

| | | |
|---|---|---|
| Цитати | 2 | Сторінка 104 |
| Посилання | 1 | Сторінка 104 |

# 0%
# Вилучень

Немає вилучених джерел

# Anti-Plagiarism v-15.257

**Максимальне співпадіння з одним документом 1.0%**

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 6%**

| | Документ | | Сумарний збіг по Базі Даних | |
|---|---|---|---|---|
| ID: 125724<br>Назва: МКР Метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних<br>Додано в БД: 2024-05-06<br>Автора: Карпович В.<br>Керівники: Березька К.<br>Консультанти:<br>Опоненти: | Символи | Лексеми | Символи | Лексеми |
| | 150632 | 1230 | 1064 (1%) | 14 (1%) |

Джерело плагіату

| ID | Опис | Наявність плагіату в документі | |
|---|---|---|---|
| | | Символи | Лексеми |

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Тетяні ГОВОРУЩЕНКО

Вячеслав КАРПОВИЧ

ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-22-2

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 (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) 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.

31 березня 2024 року

В.Карп

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: Метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних

Автор: Вячеслав КАРПОВИЧ

Спеціальність: 123 – Компютерна інженерія

Освітня програма: освітньо-наукова

Науковий керівник: Березька Катерина Миколаївна, к.т.н, 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) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10-50 джерелами на один фрагмент речення;
2) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 11,8% і адресується до джерел з інтернету та бібліотеки, що, з урахуванням наведених обґрунтувань, відповідає характеру завдання і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Катерина БЕРЕЗЬКА

Гарант ОП _____ Олег САВЕНКО

Завідувач кафедри КІІС _____ Тетяна ГОВОРУЩЕНКО

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

# РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

Дипломник: Вячеслав КАРПОВИЧ

Тема: Метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень -; кількість сторінок записки 97

1. Короткий зміст роботи та прийнятих рішень У роботі розроблено метод забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних

2. Висновок про відповідність роботи дипломному завданню Кваліфікаційна робота відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: у вступі подано об'єкт та предмет дослідження, мету, наукову новизну та практичну цінність роботи, а також характеристику структури роботи. У першому розділі проведено аналіз відомих рішень щодо забезпечення пропускної здатності дисків для застосунків з інтенсивним обсягом даних. У другому розділі здійснено дослідження предметної області та визначено автономні рішення на основі індексування для вирішення двох проблем «великих даних» у застосунка. У третьому розділі здійснено розроблення методу для побудови системи відмовостійкості для підтримки алгоритму в паралельних системах. У четвертому розділі використано набір алгоритмів інтелектуального аналізу даних і машинного навчання, які були реалізовані за допомогою бібліотеки інтерфейсу передачі повідомлень MPI. У висновках підведено підсумки досягнення результатів з розв'язання завдань дослідження.

4. Позитивні сторони роботи:

5. Негативні сторони роботи: немає.

6. Оцінка графічного оформлення та пояснювальної записки роботи: __=____

7. Відгук про роботу в цілому: __Робота виконана на належному рівні.

8. Інші зауваження: ___=

9. Оцінка дипломної роботи:

Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи вважаю, що робота заслуговує оцінки «добре» 4,50 (B)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) __Мартинюк Валерій Володимирович, д.т.н., професор, завідувач кафедри АКІТР ХНУ

"_7_" __травня__2024р.