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 «Методи перевірки трансформації та моделювання кешу комп'ютера»

КвРКІП. 180295.17.01.01 ПЗ

Виконав: студент 2 курсу, група КІ2м-21-1 Островський Д.О.
Підпис Ініціали, прізвище

Керівник к.т.н., доцент Березька К.М.
Науковий ступінь, вчене звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІІС, д.т.н., проф.
Т.О. Говорущенко
17 05 2023 р.

Хмельницький, 2023

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»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

" 01 " 09 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Островському Денису Олексійовичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Методи перевірки трансформації та моделювання кешу комп'ютера

Керівник проекту (роботи) к.т.н., доцент Березька К.М

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

Затверджена наказом ректора університету від 09.01.2023р. № 1

2. Строк подання студентом проекту (роботи) на кафедру 03.05.2023 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

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

Аналіз відомих стратегій, підходів та методів перевірки трансформації та моделювання кешу комп'ютера

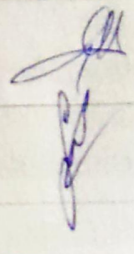
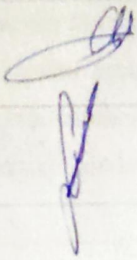
Сховище масивів з компактним інтегрованим індексом

Метод прискорення об'єднання масивів з інтегрованим індексом значень

Система для аналітики для гетерогенних систем

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

6. Консультанти розділів дипломного проекту (роботи)

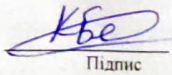
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання « 06 » 09 2022р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики КвРМ з керівником	01.09.2022	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	05.10.2022	виконано
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	05.11.2022	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	05.12.2022	виконано
5	Робота над науковою статтею	05.01.2023	виконано
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	15.02.2022	виконано
7	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	05.04.2023	виконано
8	Оформлення пояснювальної записки згідно вимог	15.04.2023	виконано
9	Попередній захист ДРМ	15.04.2023	виконано
10	Захист ДРМ на засіданні ЕК	До 10.05.2023	

Студент  Підпис Островський Д.О. Ініціали, прізвище

Керівник роботи  Підпис Березька К.М. Ініціали, прізвище

РЕФЕРАТ

Тема дипломної роботи: Методи перевірки трансформації та моделювання кешу комп’ютера

Автор роботи: Островський Денис Олексійович

Керівник роботи: Березька К.М.

Пояснювальна записка: 70 с., 2 рис., 93 джерела.

КЕШ, РАСТРОВІ ЗОБРАЖЕННЯ, СИСТЕМИ, ІНДЕКСОВАНІ МАСИВИ .

Об’єктом дослідження є процес перевірки трансформації та моделювання кешу комп’ютера.

Предметом дослідження є методи перевірки трансформації та моделювання кешу комп’ютера.

Метою дипломної роботи є розробка методів перевірки трансформації та моделювання кешу комп’ютера.

Для розв’язання поставлених задач використовувалися методи теорії комп’ютерних мереж, архітектури комп’ютерів, теорії множин, растрові зображення.

Наукова новизна отриманих результатів:

− удосконалено метод перевірки трансформації кешу комп’ютера за рахунок об’єднання індексованих масивів;

− удосконалено метод моделювання кешу комп’ютера за рахунок обробки растрових індексів графічним процесором;

На основі проведених досліджень розроблена система роботи з даними для перевірки трансформації та моделювання кешу комп’ютера.

Практична значимість отриманих результатів полягає у розробці перевірки трансформації та моделювання кешу комп’ютера

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

РЗ - растрові зображення

БД - база даних

ОС - операційна система

ПЗ - програмне забезпечення

ВСТУП

Із збільшенням кількості даних, що генеруються щодня, ефективне зберігання та запит таких даних, зазвичай багатовимірних, які можуть бути представлені за допомогою моделі даних масиву, стає все більш важливим. Тим часом, поряд з додаванням в систему все більш і більш потужних процесорів і прискорювачів, більшість сучасних обчислювальних систем містять все більш складний стек вводу-виводу, починаючи від традиційних дискових файлових систем і закінчуючи гетерогенними прискорювачами з індивідуальним простором пам'яті. Ефективний доступ до такого складного стека вводу-виводу при обробці масивів має важливе значення для використання великої обчислювальної потужності сучасних обчислювальних платформ. Одним із ключів до досягнення такої ефективності є визначення місця генерації або зберігання даних, а також відповідний вибір відповідних стратегій представлення та обробки. Ця робота зосереджена на оптимізації обробки масивів у таких складних стеках вводу-виводу шляхом дослідження двох фундаментальних питань: яке представлення даних слід використовувати, і де дані повинні зберігатися та оброблятися. Тому, розглядається проблема ефективної обробки даних масиву, представлено компактне сховище масивів для дискових даних, інтегруючи в нього індексацію на основі значень без втрат. Такі інтегральні індекси підвищують продуктивність операції фільтрації на основі значень за рахунок порядків для розміру сховища або точності. Потім розглянуто складні запити, такі як об'єднання масивів рівності та подібності, також можуть бути виконані на такому новому сховищі. Друга частина роботи зосереджена на даних, згенерованих моделюванням на прискорювачах без зберігання генерованих даних. Система генерує покращене растрове представлення на графічному процесорі, щоб зменшити вузьке місце пропускної здатності між хостом і прискорювачами, дозволяючи при цьому швидко обробляти набір складних запитів, таких як контрастний набір, так і прискорювачі. Оскільки велика кількість варіантів представлення та обробки даних надає безліч варіантів обробки масивів на місці, то розглядаємо детальне дослідження того, як такий

вибір може вплинути на аналітичну продуктивність, і застосовуємо моделювання витрат, методологію прогнозування оптимального розміщення і представлення для заданого аналітичного навантаження.

Актуальність роботи полягає в розробці методів та системи роботи з даними для перевірки трансформації та моделювання кешу комп'ютера.

Метою кваліфікаційної роботи є розробка методів перевірки трансформації та моделювання кешу комп'ютера.

Поставлена мета досягається розв'язанням таких основних задач:

− проаналізувати відомі методи для перевірки трансформації та моделювання кешу комп'ютера;

− удосконалити метод перевірки трансформації кешу комп'ютера за рахунок об'єднання індексованих масивів;

− удосконалити метод моделювання кешу комп'ютера, за рахунок використання графчних процессорів для обробки растрових індексів;

− реалізувати експериментальні дослідження розроблених рішень.

Об'єктом дослідження є процес перевірки трансформації та моделювання кешу комп'ютера.

Предметом дослідження є методи перевірки трансформації та моделювання кешу комп'ютера.

Наукова новизна отриманих результатів:

− удосконалено метод перевірки трансформації кешу комп'ютера за рахунок об'єднання індексованих масивів;

− удосконалено метод моделювання кешу комп'ютера за рахунок обробки растрових індексів графічним процесором;

Практична значимість отриманих результатів полягає у розробеній перевірки трансформації та моделювання кешу комп'ютера.

Для розв'язання поставлених задач використовувалися методи теорії комп'ютерних мереж, архітектури комп'ютерів, теорії множин, растрові зображення.

На основі проведених досліджень розроблена система роботи з даними для перевірки трансформації та моделювання кешу комп’ютера.

За темою кваліфікаційної роботи опублікована одна публікація у Збірнику наукових праць за матеріалами XIV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2022». Хмельницький – 2022 [92].

1 АНАЛІЗ ВІДОМИХ СТРАТЕГІЙ, ПІДХОДІВ ТА МЕТОДІВ ПЕРЕВІРКИ ТРАНСФОРМАЦІЇ ТА МОДЕЛЮВАННЯ КЕШУ КОМП'ЮТЕРА

1.1 Поняття про перевірку трансформації та моделювання кешу комп'ютера

Використання великих даних і глибоких нейронних мереж надало дослідникам можливість до ефективного отримування та обробки значного великого обсягу даних [1]. Ця тенденція очевидна з обсягу даних, що збираються великомасштабними дослідницькими проектами, починаючи від терабайт великомасштабних зображень телескопів [2] до трильйонів частинок при моделюванні [3, 4, 5], від реконструкції медичних зображень [6] для відстеження змін глобального клімату. Більші та швидші системи будуються для задоволення такого попиту на обробку даних. Такі системи мають обчислювальні потужності від одного до двох ексафлопс [6, 7]. Більшість даних, що збираються, мають багатовимірний характер, що робить модель даних масиву її природним представленням. У такій моделі даних дані представляються в багатовимірних масивах, пов'язуючи кожен кортеж даних з декількома стовпцями (атрибутами) з одним n-вимірним вектором (координатами). Традиційно такі дані зберігаються в паралельних файлових системах [8], і легко покладаються на формати даних масивів, такі як доступ. Збір даних є лише першим кроком. Зібрані дані повинні бути ефективно оброблені або проаналізовані для створення цінності для суспільства. Аналітика даних масиву, як правило, є онлайн аналітичною обробкою [9]. Запити стилю, починаючи від простої фільтрації, агрегації та об'єднання [10, 11, 12, 13, 14, 15] до більш складних запитів, таких як подібність об'єднання та контрастний набір видобутку, в цьому контексті є важливими. Швидкість, або час до розуміння, є важливими у науці, керованій даними, оскільки швидший час до розуміння призводить до швидшого зворотного зв'язку та коригувань, що може значно прискорити загальна ефективність досліджень і розробок. Однак останні архітектурні тенденції представляють значні виклики для такого завдання. По-

перше, сьогодні зростає розрив між обчислювальною та ввідною потужністю систем. Хоча окремі обчислювальні вузли обчислювальних систем сьогодні можуть легко обробляти терафлопс обчислень, навіть з останніми інноваціями, такими як твердотільні диски та розривний буфер, швидкість доступу вводу-виводу залишається на рівні гігабайт для другого рівня. Крім того, основний розмір пам'яті останнім часом не збільшується з такою ж швидкістю, як ядра в одній машині, що робить пам'ять одного ядра такою, що може ефективно зменшуватися. Другою тенденцією є все більш складна і неоднорідна ієрархія пам'яті. Хоча віддалені сховища завжди були популярні у високопродуктивних обчислювальних системах, комерційні, хмарні системи управління даними, такі як Amazon [16] та Microsoft SQL [17], також рухаються до віддаленної системи зберігання. Крім того, системи сьогодні зазвичай включають прискорювачі, такі як графічні процесори [18] для задоволення потреби в масивних паралельних додатках даних. Більшість таких прискорювачів мають незалежну пам'ять пристрою, відокремлену від основної пам'яті. Зазвичай вони підключаються до центрального процесора через повільніші периферійні шини, створюючи додаткові вузькі місця для ефективного аналізу даних. Враховуючи такі проблеми, системи обробки масивів сьогодні повинні максимально економити обмежену пропускну здатність вводу-виводу, а також мати можливість аналізувати розсіювання даних в різних місцях складних стеків вводу-виводу сьогодні. Це спонукає розробника системи ретельно розглянути два питання: по-перше, де дані повинні зберігатися і оброблятися, тобто чи повинні дані зберігатися в файловій системі і оброблятися, або оброблятися по мірі генерації або збору даних. В останньому випадку, чи потрібно обробляти дані в хості, де пам'яті багато, а кеш гнучкий, або за допомогою прискорювача, де паралелізм масивний, а пропускна здатність пам'яті вище. По-друге, які уявлення слід використовувати для такої аналітики, тобто який найбільш стислий та ефективний спосіб зберігання та обробки таких даних. Наприклад, чи корисні індекси, якщо так, то які індекси є оптимальними. Чи потрібно зберігати дані у форматах наближення, щоб підвищити ефективність обробки, і як узгодити втрату точності. Відповіді на ці питання переплітаються: те, як зберігаються дані, частково

визначає, як їх можна передавати і обробляти, і навпаки. Мета роботи спрямована на вивчення рішень цих завдань. Система обробки масивів повинна орієнтуватися у все більш складному та неоднорідному стеку вводу-виводу сьогодні. Зокрема, сучасна система обробки масивів може оптимізувати представлення та розміщення даних у сучасних стеках вводу-виводу даних. Розглянуто два основні сценарії наукової аналітики: як можна розробити кращі уявлення для ефективного аналізу даних на основі диска після документа, і як дані, що генеруються на гетерогенних платформах, можуть бути проаналізовані, як тільки вони генеруються, також відомі як обробка на місці.

Розглянемо пошук ефективного вибору значення для даних масиву на основі диска. Традиційне наукове переповнення значною мірою залежить від файлових робочих процесів. Зібрані та змодельовані дані зазвичай зберігаються в наукових форматах файлів [19, 20]. Файли, у свою чергу, часто зберігаються на обертовому диску або віддалених паралельних файлових системах [21, 22]. Для вводу-виводу та ефективності запиту дані зазвичай зберігаються та доступні в деталізації частин, групуючи комірки з розмірами всередині одного гіперпрямокутника разом. Така організація дозволяє швидко підбирати запити на основі розмірів. У той час як нові системи зберігання розробляються для забезпечення кращої продуктивності та гнучкості [23], базова організація зберігання дотримується того ж принципу. Однак, хоча перевірене традиційне сховище масивів перевершує запити вибору на основі розмірів, його здатність обробляти виділення на основі значень, також відомих як атрибути, набагато більш обмежена. Багатьом системам зберігання даних [24, 25, 26, 27, 28] доводиться повертатися до повного сканування для вибору на основі вартості. Природно, це створює величезні непотрібні введення/виведення, особливо коли селективність низька. Зовнішні індекси, особливо растрові індекси на основі бітів [29], є ще одним методом прискорення вибору на основі значень. Однак отримання вихідних даних на основі індексу також є дорогим через деталізацію вводу-виводу частин. Ця ситуація ускладнена збільшенням розриву між обробкою та входом-виходом. Оскільки як обробка, так і ввід/вивід є більш ефективними, коли розмір пакета даних збільшується. Розмір

частини зазвичай встановлюється на більше значення і часто досягає десятків мегабайт [30], створюючи більше додаткових витрат на отримання окремої змінної на основі зовнішнього індексу. Як наслідок, зовнішні індекси більш широко використовуються як самостійне узагальнення даних для швидких наближених запитів [31, 32, 33, 34, 35].

Розглянемо ідею збільшення сховища масивів для ефективної обробки точних запитів на вибір значення. Ключовий аспект полягає в тому, що замість зберігаючись зовні, індекс може бути інтегрований у сховище масиву, з додатковою залишковою інформацією, що зберігається поряд з індексом, щоб забезпечити точність при додаванні невеликих додаткових витрат на зберігання. Такий інтегральний індекс може бути реалізований і запитаний, а також сховище масивів з такою інтегрованою підтримкою індексів [34]. Крім того, в така організація зберігання оптимізує один з найбільш фундаментальних запитів в обробці масивів - об'єднання масивів. Адаптація до необхідності обробки масивів на гетерогенних платформах має по суті такий підхід. Ємність вводу-виводу системи сьогодні зростає не так швидко, як обчислювальна потужність, викликаючи все більше вузького місця в передачі даних. Одним з рішень є аналітика на місці, тобто виконання аналізу даних у міру генерування даних. Така аналітика дозволяє швидше інтегрувати результати, отже, полегшує більшу взаємодію між користувачем і даними. Така перевага привернула значні дослідження з обробки запитів на місці [35-47].

Ще одним апаратним досягненням останнього десятиліття є зростання багатоядерних прискорювачів, особливо графічних процесорів, або графічних процесорів. З підвищенням продуктивності одного ядра через лімітації потужності та тепловіддачі [48] системи звертаються до все більшого паралельності даних для підвищення продуктивності. Замість того, щоб приховувати затримку за допомогою складної ієрархії кешу та виконання поза порядком, графічні процесори вибирають виконання більшої кількості операцій за тактовий цикл, використовуючи більше одиниць виконання, і приховують затримку пам'яті за допомогою достатньої кількості паралельних обчислень одночасно. Нові графічні

процесори часто пропонують десятки терафлопс продуктивності на одній карті і мають відносно вигідне споживання енергії. Крім того, потреба в масових паралельних додатках даних зростає через популярність інтенсивних обчислювальних методів машинного навчання, а також інших методів, керованих даними. В результаті системи звертаються до інших форм прискорювачів для кращої продуктивності. Деякі сучасні системи упаковують до 97% своєї обчислювальної продуктивності всередині своїх прискорювачів [49].

Моделювання, що виконується на таких різнорідних платформах, створює додаткові проблеми для обробки даних масивів. На додаток до вже обмежених потужностей вводу-виводу, як через історію, так і з причин продуктивності, домінуючі прискорювачі сьогодні зазвичай мають власну пам'ять пристроїв і взаємопов'язані між собою з хост-системою за допомогою шин, таких як PCI-E, що додатково обмежує гучність і пропускну здатність, через яку прискорювач може отримати доступ або вивести дані. Враховуючи, що паралельно з моделюванням відбувається обробка масивів на місці, така аналітика також повинна максимально економити системні ресурси. Друга частина зосереджена на тому, як системи обробки масивів можуть вирішити такі проблеми, ретельно вибираючи також представлення даних як розміщення даних та обчислення. Розглядатимемо також гнучку систему запитів на місці, призначену для даних, що генеруються на гетерогенних платформах. Замість передачі вихідних даних до центрального процесора для обробки генерує приблизне растрове представлення, зручне для графічного процесора, і запитує приблизне значення. Це має перевагу прискорення передачі між акселератором і центральним процесором, а також запити на центральний процесор. Крім того, в різнорідних системах, крім представлення, розміщення даних і обчислень також повинні бути ретельно розглянуті для кращого використання доступної пам'яті та обчислювальних ресурсів. Для обробки конкретного запиту є можливість перенести вихідні дані в центральний процесор і обробити їх там, або за допомогою кнопки растрове представлення для зменшення пам'яті та тиску вводу-виводу. Також, можна стиснути дані в графічному процесорі, щоб зняти тиск пам'яті пристрою, і обробити дані приблизно там.

Враховуючи різноманітність ресурсів обчислень і пам'яті в таких системах, малоймовірно, що універсальний підхід завжди найкращий. Адаптивний гетерогенний аналітичний механізм робить два внески: підтримує розміщення аналітики на графічних процесорах, розширюючи растрове представлення та розробляючи відповідні алгоритми для наближеної обробки з використанням обмеженої пам'яті пристрою; включає офлайнову модель продуктивності, здатну передбачити відносно вартість варіантів обробки запитів і вибрати оптимальний на основі наявного завдання та ресурсів.

1.2 Аналіз моделей даних масивів

Розглянемо модель даних масиву, як растрові індекси та інвертовані списки, які можуть бути використані для кодування позицій елементів у наборі даних, довідкову інформацію про формат чисел з плаваючою комою та стиснення, а також про архітектури графічних процесорів. Масив зберігає впорядковані багатовимірні дані. На відміну від реляційних баз даних, де кортежі зберігаються невпорядкованими, значення в масиві впорядковані і організовані за його розмірами, що сприяє набагато швидшій швидкості пошуку за розмірами запитів. Кожен масив відображає карту один до одного від n-вимірного вектора до кортежу атрибутів. Масив можна визначити як відображення один до одного з вектора розмірності в кортеж з попередньо визначеними значеннями. Назвемо їх рангом масиву, а n-вимірні вектори координатами. Деякі координати можуть не мати відповідних значень. Ці позиції представляються у вигляді порожніх значень. Посилання на масив лише з одним стовпцем буде набором даних. Для ефективності вводу-виводу та підмножини масив часто зберігається та обробляється в термінах множин. Більшість сховищ масивів сьогодні використовують стратегію регулярного фрагментування [50], яка ділить набір даних на кілька гіперпрямокутних множин однакового розміру відповідно до координат елементів. Елементи в множині зберігаються безперервно, з додатковим стисненням для збереження вводу-виводу. Важливими для розгляду є растрові індекси та

інвертовані списки. Як растрові індекси, так і інвертовані списки надають спосіб індексувати записи у стовпці. Для даних N записів, растровий індекс використовує N бітів, щоб вказати, чи задовольняє кожен з цих записів певній умові. Він використовувався для прискорення запитів в реляційних базах даних [51-59], а також на наукових наборах даних [60-64]. Растрове зображення зазвичай зберігається і обробляється в стислому вигляді для ефективності. Більшість методів використовують той факт, що в растровому зображенні часто є суміжні множини 0 і 1. Отже, замість того, щоб зберігати ці 0 та 1 як буквені слова, їх можна зберігати як спеціальне слово. Нестиснений растровий малюнок вважатимемо бітовим вектором. Хоча існують інші методи узагальнення даних, такі як вибірка [65], гістограми [66-71], вейвлети [72-81], і ескізи даних [72-82], поширюючи ці уявлення на високовимірні дані, можуть бути жорстким [83]. На відміну від них, растрові індекси мають переваги, що зберігають просторову інформацію багатовимірного масиву, а також дозволяють швидше перетинатися і об'єднуватися на основі побітових операцій в апаратному забезпеченні. Іншим методом індексування записів є зберігання ідентифікаторів записів, які відповідають умові індексу, у перевернутому списку. Однак зберігати ідентифікатори буквально не бажано через додаткові витрати на простір. Натомість, припускаючи, що ідентифікатори записів збільшують монолітні цілі послідовності, дельта суміжних ідентифікаторів може бути збережена. У випадку, якщо дельта може бути від'ємною, зигзагоподібне кодування може бути використане для перетворення цілих знаків у цілі числа без знака. Це перетворює проблему зберігання перевернутих списків до стиснення малих цілих чисел. Один з найпростіших, але ефективних, методів є змінні байти [84], які використовують кількість байт для кодування значення, і певну кількість бітів для позначення кількості байтів, що використовуються. Найсучаснішими методами є блочні, такі що кодують числа у відносно великих блоках, таких як 64 або 128, і використовують біти для кодування кожного числа в блоці. Якщо число не вдалося закодувати за допомогою бітів, то в місце ставлять спеціальний маркер, а номер кріплять зі зворотного боку блоку. То це може досягти високої швидкості декодування, але зберегти високий ступінь

стиснення за допомогою векторизації [85]. Непрактично зберігати растровий або перевернутий список для кожного унікального плаваючого числа в наборі даних. Таким чином, потрібна якась прив'язка. Найпростішим методом бітінга є рівноширинний бітнінг, який ділить набір даних на біти, що містять рівні інтервали домену значень. Інший метод - це рівноглибинне прив'язування. За ним ділиться набір даних на біти з рівною кількістю елементів і, як правило, більш точні в приблизних агрегаціях, але набагато повільніше будуються. Деякий більш складний метод [86] передбачає наявність апріорних знань частоти запиту, тому часто не підходять у випадку, коли умова та діапазон запиту можуть змінюватися.

Розглянемо число з плаваючою комою та стиснення. IEEE 754 є переважаючим стандартом представлення з плаваючою комою. Він описує скінченне плаваюче число, використовуючи компоненти: знаковий біт, показник. Ці частини об'єднані в двійкову форму, і представляють числове значення. Завжди вибираємо показник як найменший показник, що дозволяє представити значення точно для забезпечення унікального двійкового представлення. Крім того, провідним бітом завжди є 1 і представлений неявно. Формат IEEE з плаваючою комою має властивість зберігати лексичний порядок. Тобто, якщо бінарні представлення двох чисел з плаваючою комою розглядати як цілі числа знакової величини, то це не змінює його часткового порядку. Ця властивість показує, що числа з плаваючою комою в невеликому діапазоні, ймовірно, містять ті самі вищі біти, що є важливим для схеми зберігання. Загальні потоки з плаваючою комою важко стиснути. Більшість успішних методів використовують комбінацію диференціювання і предикатів. Також, використовує два предикатів на основі хешу для прогнозування наступного значення на основі попередніх значень і їх дельт. Потім прогноз порівнюється з фактичними значеннями і обчислюється залишок. Потім використовує модифіковану схему стиснення байтів змінних для стиснення залишкових значень. На практиці загальні компресори, такі як сімейство компресорів, часто використовуються для стиснення вмісту множин набору даних.

Розглянемо гетерогенні платформи. Графічні процесори покладаються на великомасштабний паралелізм даних для покращення обчислень та

енергоефективності. Графічний процесор має багато потокових процесорів, кожен з яких можна розглядати як векторизований процесор, здатний виконувати одні й ті ж інструкції на декількох операндах одночасно. Кожен з них має свій кеш L1. Вони використовують кеш L2 і мають доступ до глобальної пам'яті пристрою з високою пропускною здатністю. Частина кешу L1 може бути явно керована програмістом, і називається спільною пам'яттю. Хоча існують інші експериментальні розробки, в даний час графічний процесор зазвичай не може ефективно виконувати прямий доступ до основної пам'яті хоста і обмінюється даними з хост-системою за допомогою шини PCI-E, яка має обмежену і спільну пропускну здатність. Приховування затримки передачі даних зазвичай є ключем до підвищення продуктивності програми графічного процесора. Користувач програмує графічний процесор за допомогою абстракції, яка називається легкими потоками. Кожен потік графічного процесора являє собою одну частину логіки виконання, яка може бути виконана масово паралельно. Кожна паралельна функція, що виконується на GPU, також звана ядром, складається з тисяч потоків графічного процесора з однаковою логікою виконання. Ці потоки поділяються на групи, які називаються блоками, кожен з яких завжди виконується разом на одному потоковому процесорі. Потоки в одному блоці можуть спілкуватися через спільну пам'ять і бар'єри. Потоковий процесор виконує потоки блоком у зернистості деформацій, кожен з яких зазвичай містить 32 або 64 потоки. Виконання відповідає моделі у будь-який момент однакові інструкції виконуються для всіх потоків у викривленні. Терміни, використані тут, відповідають моделі програмування CUDA від NVIDIA. Інші моделі програмування, такі як OpenCL, можуть використовувати інші терміни. Два важливих аспекти графічних процесорів інформують про проектування гетерогенних аналітичних систем. З точки зору вводу-виводу, хоча існують інші методи в розробці, більшість пристроїв графічного процесора сьогодні все ще підключені до хост-системи через шину PCI-E, і дані переміщуються між пам'яттю хоста і графічного процесора явно. Незважаючи на швидке збільшення, розмір пам'яті на пристрої та пропускна здатність хоста-графічного процесора все ще досить обмежені, особливо враховуючи, що вузли

також стають все більш щільними. З точки зору обчислень, коли виконується ядро, кожному мультипроцесору виділяється один або кілька блоків для виконання. У той час як блоки, що виконуються одним і тим же мультипроцесором, можуть бути перекриті один з одним, щоб приховати затримку пам'яті, блок зазвичай не замінюється з SM, поки він не завершить своє виконання, що ускладнює міжблокову синхронізацію. Більшість графічних процесорів до недавнього часу також не підтримували розбіжне виконання всередині однієї обгортки і не мають гарантії просування вперед, що унеможливлює зайняте очікування або блокування потоків всередині обгортки. Будучи розробленими як співпроцесор, більшість сучасних прискорювачів також покладаються на відносно дорогий API на стороні хоста для розподілу пам'яті та планування ядра. Разом з пам'яттю пристрою та пропускною здатністю ці обмеження необхідно враховувати при розробці та виборі стратегій обробки запитів у гетерогенній системі.

Приблизна обробка запитів викликала інтерес, оскільки вона корисна як в реляційних базах даних, так і в движках потокової обробки. Одним з основних методів є збереження вибірки даних репрезентативними. Одним з напрямків є доповнення відібраних даних додатковим інформуванням для зменшення похибки вибірки. Подібні методи [87] - ще одна популярна методика. Останнім часом пропонується використовувати вибірку для калібрування результату, обчисленого збереженими агрегаціями [88]. Існують також методи узагальнення даних, такі як гістограми, вейвлети або ескізи даних, такі як фільтри Блума [89] та ескізи лічильних хвилин [90]. Хоча ці методи дуже цікаві, однією з особливих проблем багатьох таких структур даних є те, що вони не дуже ефективні для високовимірних даних через великі розмірності. Растрові зображення [91], з його гнучкістю швидкого запиту декількох стовпців і збереження просторової інформації, часто можуть використовуватися як репрезентативні данні для приблизної обробки [92] без вихідних даних.

Розглянемо можливості покращення продуктивності вибору масивів на основі вартості. Через багатовимірний характер масивів, більшість сховищ масивів покладаються на фрагментовану структуру, зберігаючи масиви в блоках вводу-

виводу на основі своїх розмірів, щоб забезпечити ефективне виконання такі операції, як повне сканування та вибір розмірів. Аналогічним чином, можна вибрати всі частинки високої енергії перед подальшим аналізом або візуалізацією. Такі ситуації вимагають ціннісного відбору або фільтрації. З величезною кількістю даних під рукою, а обчислювальна потужність перевершує можливості вводу-виводу систем сьогодні. Важливо, щоб операція фільтра максимально зберігала ввід/вивід. Крім того, оскільки один і той же набір даних часто аналізується різними командами, в різний час і для різних цілей. Також важливо, що діапазон запитів можна вибирати гнучко, не покладаючись на наші апріорні знання про можливі запити. Однак можливості вибору значень на масивах обмежені. Багато існуючих сховищ-масивів зберігають дані лише на основі своїх розмірів і повертаються до сканування під час виконання операції фільтрації. Нерідкі випадки, коли користувачі вручну зберігають відфільтровані результати для різних діапазонів запитів з метою прискорення подальшої аналітики. Це створює непотрібне дублювання даних і вимагає, щоб діапазони запитів були відомі заздалегідь. Іншим варіантом є зовнішні індекси, зокрема растрові індекси на основі бітів. Однак отримання вихідних значень на основі його растрових індексів є надзвичайно дорогим; отже, растровий індекс часто розглядається як стисле представлення даних із втратами і використовується для виконання запиту приблизно без посилання на вихідні дані. Як наслідок, типи запитів, на які можуть відповісти такі індекси, обмежені. Також, ці методи часто спираються на стратегію прив'язки і припущення про розподіл діапазону запитів для досягнення більш високої точності. Крім того, оскільки точні дані не вдалося відновити, неможливо передати відфільтровані дані для подальшого аналізу. В цілому, існують переваги використання зовнішніх індексів як представлення наукових даних, за умови, що можна уникнути втрати точності, відновити вихідні дані, і додаткові витрати на введення-виведення є прийнятними. Одним із способів досягнення цієї мети є інтеграція індексу в сховище. Це сховища масивів з підтримкою компактних та інтегрованих індексів значень. На додаток до традиційних простих наборів даних, також підтримує зберігання та запит даних в індексованих наборах даних, зокрема,

значень індексуються з використанням ряду бітів, вказаних користувачем, а також зберігають необхідну інформацію для відновлення вихідного набору даних. Це дозволяє використовувати методи ефективної фільтрації, повідомляючи точні результати незалежно від використовуваної стратегії зв'язування. Крім того, оскільки вихідні дані можна відновити, відфільтрований результат можна агрегувати, візуалізувати або використовувати як вхідні дані для подальших кроків аналізу. Крім того, це представлення майже завжди займає той самий або менший розмір пам'яті, що й вихідний набір даних. Основна ідея підходу полягає в наступному. Дані в масиві реорганізовуються відповідно до бітів, до яких він належить. Кожен біт відстежує позиції значень в ньому за допомогою стислого позиційного індексу. Підмножина також зберігає залишковий сегмент даних, тому фактичні значення в підмножину можна відновити. Система використовує діапазон підмножина як додаткову інформацію для стиснення залишкових даних, зменшуючи надмірність сховища. Крім того, він реалізує схему зберігання на основі множин, забезпечуючи його швидку здатність підмножини. Для експериментального оцінювання результатів будуть використані як синтезовані, так і реальні дані. Результати показують, що в порівнянні з вихідним набором даних, метод досягає позитивного ступеня стиснення. Він може виконувати точні операції фільтрації на порядок або більше швидше, коли селективність низька, і все ще перевершує операції на простому наборі даних, навіть коли селективність відносно висока. Аналогічну перевагу продуктивності можна отримати над базою даних масивів. Розглядатимемо техніку зберігання масивів, інтегруючи індекс значень з кількістю заданих користувачем бітів у сховище, що полегшує швидке та точне виконання операції фільтрації. У процесі роботи пропонується кілька варіантів ефективного кодування індексу значень та його додаткових даних. Представляємо систему зберігання масивів з підтримкою наборів даних як з індексами значень, так і без них. Покажемо метод доступу, придатний для індексованого набору даних, і як реалізувати загальні операції масиву поверх нього. Детальна оцінка показує, що зберігання даних в індексованій формі часто може зберегти обсяг пам'яті, часто перевершує виконання повного сканування

звичайних наборів даних, навіть коли селективність висока, і може бути ефективно перетворено назад у звичайну форму, якщо це необхідно. Розглянемо організацію зберігання індексованого набору даних. Розглядатимемо новий формат зберігання даних з індексованими значеннями. Індексоване представлення масиву організовує дані масивів за допомогою індексу на основі бітів: значення групуються в ряд бітів, а їх позиції зберігаються в позиційному індексі кожного біту. Оскільки в кожній підмножині існує кілька унікальних значень для даних, позиційний індекс надає лише приблизну картину набору даних. Позиційний індекс відстежує позиції значень у кожній підмножині. Залишкові дані зберігають додаткові дані, необхідні для відновлення вихідного набору даних. Необхідно реконструювати набір даних без втрат. Система зберігає цю інформацію як залишкові дані кожної підмножини. Значення розділені на п'ять бітів залежно від діапазону. Кожен біт зберігає позиції значення в ньому, а також залишкові дані. Використовуючи це представлення запиту потрібно лише отримати біти, які містять елементи, що представляють інтерес, а потім реконструювати елементи, використовуючи позиції та залишки, заощаджуючи як ввід/вивід, так і час обробки. Порожні елементи представлені неявно, не зберігаються в будь-яких бітах, що дозволяє індексованому набору даних ефективно обробляти розріджені набори даних. Крім того, при зберіганні масивів система ділить індексований набір даних на невеликі фрагменти, які є основними одиницями вводу-виводу та обробки набору даних. Як стратегію фрагментування, так і представлення фрагмента потрібно налаштувати для ефективного зберігання індексованого набору даних.

1.3 Постановка задачі дослідження

Поставлена мета досягається необхідністю розв’язання таких основних задач:

– проаналізувати відомі методи для перевірки трансформації та моделювання кешу комп’ютера;

- удосконалити метод перевірки трансформації кешу комп’ютера;
- удосконалити метод моделювання кешу комп’ютера;
- реалізувати експериментальні дослідження розроблених рішень.

1.4 Висновки

Таким чином, проаналізовано відомі методи для перевірки трансформації та моделювання кешу комп’ютера, виділено недоліки та зроблено постановку задачі дослідження для вирішення проблем.

2 СХОВИЩЕ МАСИВІВ З КОМПАКТНИМ ІНТЕГРОВАНИМ ІНДЕКСОМ

2.1 Стратегія зв'язування та поділу

Задана область значень методів зв'язування може бути багато. Розглядатимемо тільки біти на основі діапазону, тобто, якщо набір даних має домен значення, таким чином розділяючи дані на m бітів. Для ефективної операції підмножини набір даних все ще ділиться на n гіперпрямокутних сегментів однакового розміру відповідно до його розмірів за допомогою регулярного поділу. У поєднанні з m бітами це створює загальну кількість фрагментів. Бітів можуть бути сотні, а деякі біти можуть містити лише кілька елементів, система групує ці фрагменти разом на множині для кращої ефективності вводу-виводу. Однак, зберігання всіх m бітів сегмента всередині фрагмента призводить до того, що будь-який вибір на основі значень завантажує всі біти з диска, ігноруючи сенс індексації. Отже, можна згрупувати кілька бітів разом, так що кожен поділ містить дані k бітів у сегменті. Індексований поділ можна однозначно ідентифікувати за його граничними координатами та біт, який він містить. Карта фрагментів на основі дерева зберігає адресу кожного фрагмента. Оскільки кількість множин зазвичай не дуже велика, тому система зберігає цю карту в основній пам'яті.

Індексований поділ містить три сегменти: залишкові дані та позиційний індекс, як описано раніше, а також заголовок фрагмента для цілей обліку. Одним з питань є розташування сховища позиційних індексів та житлових даних всередині множини. Проектне рішення виходить із спостереження, що позиційні індекси не завжди необхідні для обробки запитів. Наприклад, оператор агрегації може вибрати всі елементи всередині множини, і, таким чином, не потребує позиційних індексів для продовження. Таким чином, з точки зору компонування, позиційні індекси та залишкові дані всіх бітів у блоці об'єднуються у два окремих сегменти. Таким чином, коли позиційний індекс не потрібен, завантажується лише залишковий сегмент даних, не шкодуючи зайвого вводу-виводу для завантаження позиційних індексів. Позиційний індекс для кожного елемента в підмножині позиційний індекс

позначає його розмірні позиції всередині сегмента. Щоб досягти цього, багатовимірні координати елементів спочатку серіалізуються до лінійних зміщень. Зміщення можуть бути збережені в стислому вигляді, або у вигляді растрового зображення, або з використанням схеми стиснення з перевернутими списками монолітної послідовності. Обидва способи мають свої переваги. Растрові зображення, як правило, більш ефективні у виконанні наближених запитів на основі встановлених перетинів. Однак декодування зміщень з растрового зображення вимагає відстеження того, скільки бітів вже було відвідано, тому призводить до додаткових додаткових витрат процесора. На обсяг зберігання двох методів може впливати набір даних і стратегія прив'язки. Залишкові дані підмножини зберігають додаткові дані, необхідні для розрізнення різних значень в одній підмножині. Це можна розглядати як проблему стиснення: як без втрат стиснути значення в підмножині з урахуванням його діапазону. Просте застосування загальних або наукових алгоритмів стиснення даних до підмножини значень часто є неоптимальним, оскільки компресор не знає про зв'язок значень. Представимо загальну структуру стиснення даних, обмежених діапазоном.

Стискаємо дані за два кроки: зіставляємо і кодуємо. Фаза карти видаляє надлишкову інформацію з потоку вхідних значень, відображаючи дані в потік цілих чисел з однаковою бітовою довжиною, що полегшує стиснення. Фаза кодування вхідний потік, згенерований фазою карти, стискає його в більш конденсоване представлення. Карта має кілька варіантів виконання операції. Вона не робить ніяких перетворень. Обгрунтуванням є те, що дані всередині однієї підмножини напевно мають деяку схожість, тому повинні бути вже легко стиснені. Якщо розглядати їх як цілі числа знакової величини, то числа з плаваючою крапкою зберігають лексичний порядок своїх двійкових представлень. Тому, якщо двійкові представлення мають загальний префікс, то всі значення всередині діапазону мають однаковий префікс. Довжина загальної приставки визначається діапазонами плаваючих чисел між межами. Видалення цього префікса перетворює значення float на менші цілі числа.

Розглянемо проблеми з непідписаним перевертанням. Проблема видалення префіксів полягає в тому, що числа з плаваючою комою представлені в знаковій величині. Тому, хоча значення мають лише невелику різницю, їх двійкові представлення не мають спільних префіксів бітів. Непідписані перевертання вирішують цю проблему шляхом зіставлення всіх чисел з плаваючою комою в суміжному діапазоні цілих чисел без знака відповідно до їх значення. Це можна зробити, перевернувши біти знаків позитивних чисел і всі біти від'ємних чисел. Після віддзеркалення залишок обчислюється простим відніманням відображеної нижньої межі від перевернутого значення. Кодування після фази зіставлення використовує кодувальник для стиснення зіставлених даних у більш компактну форму. Розглянемо чотири різні методи. Перший метод використовує алгоритм для стиснення зіставлених даних. Якщо зіставлені цілі числа завжди додатні, то він викликається безпосередньо. Якщо дані підписані, перед компресором застосовується зигзагоподібне кодування, так що залишки з меншими абсолютними значеннями все ще зберігаються з меншою кількістю бітів. Другий метод спочатку обчислює дельту між зіставленими цілими числами. Потім він застосовує зигзагоподібне кодування до даних і використовує алгоритм для стиснення результатів. Третій метод використовує алгоритм стиснення плаваючих чисел для безпосереднього стиснення зіставлених даних. Четвертий метод є вдосконаленою версією другого алгоритму. Він спрямований на подолання двох недоліків другого методу, використовуючи його для стиснення обмежених значень. По-перше, при генерації хеш-значення метод витягує фіксовану кількість найбільш значущих бітів зі стисненого значення. Оскільки зіставлене ціле число, ймовірно, містить нуль бітів у більших бітах, це призводить до меншої кількості інформації про використовуване значення. По-друге, алгоритм другого методу використовує модифікований формат варіантів байтів для стиснення різниці між фактичним значенням і прогнозованим значенням. Цей формат може зберігати непотрібні нульові біти, а також вимагає три біти для кодування довжини закодованого числа. Кодувальник використовує ті ж предикати на основі хешу, що і компресор. Однак він налаштовує правий зсув бітів параметра двох предикатів відповідно до

загальних префіксальних бітів верхньої та нижньої межі (рядок 2-4), так що біти вилучення не завжди включають деякі нульові біти. Згенерований залишок також стискається за допомогою алгоритму для кращого ступеня стиснення (рядок 18). Бітовий вектор селектора відстежує, який предикат використовується (рядок 15) і передує стислому залишку. Всі методи відображення можуть бути векторизовані за допомогою інструкцій без умовних стрибків, а алгоритми стиснення оптимізовані для виконання. Поєднання різних методів відображення та кодування дає ряд можливих залишкових компресорів, але не всі комбінації мають сенс.

Метод відображення може працювати тільки з кодувальником, який може ефективно стискати плаваючі дані. Тому, розглядаємо лише такі комбінації: видалення префікса, префікс видалення.

Розглянемо обробку запитів. Набір даних може бути як звичайним набором даних (без індексу), так і індексованим. Обидва типи наборів даних надають API методу доступу на основі фрагментів для клієнтських програм. Система, також, надає набір складених операторів на основі витягування для користувачів, щоб легко отримувати доступ або обробляти ці набори даних, реалізовані за допомогою того ж API на основі фрагментів. Спочатку оператор зчитує набір даних, звичайний або індексований, зі сховища. Система, також, має можливість зчитування даних у зовнішніх форматах безпосередньо у вигляді звичайного набору даних через. Вибір на основі значень або розмірів може бути виконаний на сканованому наборі даних, перш ніж буде додатково оброблений батьківським оператором (наприклад, агрегація) або клієнтською програмою. Індексовані та звичайні набори даних можуть бути перетворені один в одного за допомогою оператора, якщо батьківський оператор не підтримує інший тип набору даних.

Розглянемо метод доступу на основі фрагментів, наданий системою. Клієнт переглядає фрагменти набору даних і зчитує дані всередині фрагмента за допомогою чотирьох основних API для перегляду фрагментів у наборі даних. А також додаткові методи для запиту форми або типу фрагмента або бітів, які він містить. Більшість методів мають свою звичайну семантику. Однак, якщо тільки після отримання фрагмента користувач використовує ітератор фрагмента для

навігації по його вмісту. Повторний виклик наступного методу переходить до наступної підмножини в множині. Для кожної підмножини значення та позиції елементів повертаються у менших блоках фіксованого розміру, які називаються сегментами, у тому ж порядку, що й звичайний набір даних. У порівнянні з розпакуванням всієї підмножини одночасно, це дозволяє уникнути доступу до непотрібних плиток, а також покращує локальність кешу, отже покращує продуктивність сканування. Виклик наступного методу переводить систему до наступного сегмента та розпаковує її. Потім доступ до значень елементів у підмножині можна отримати за допомогою методу. Функція повертає інформацію про позицію, якщо її завантажено. Функція повертає лише непусті елементи, пусті значення природно ігноруються через представлення сховища. Розглянемо, як метод доступу, описаний у попередньому підрозділі, може бути використаний для реалізації загальних операцій обробки масивів. Оператор вибирає значення в певному діапазоні в наборі даних. Спочатку він визначає набір бітів, який перетинається із запитаним діапазоном, і перебирає всі фрагменти, які містять ці біти. Для бітів у діапазоні запитів повністю розпакується та повертається весь біт. В іншому випадку він перевіряє елементи та повертає елементи в діапазоні запитів. Оператор фільтра завантажує позиційний індекс тільки в тому випадку, якщо батьківський оператор вимагає збереження вводу-виводу. Оператор повертає гіперпрямокутну розмірну область масиву. Спочатку він знаходить і повторює всі множини, що містять зацікавлену область. Якщо весь поділ покритий запитуваною областю, він повертає поділ безпосередньо. В іншому випадку він переглядає зміщення та повертає лише елементи, які знаходяться всередині області запиту. Оскільки перетворення зсувів на координати, а потім перевірка наявності умови запиту є обчислювально вартісним, оператор підмножини обчислює діапазони зсувів, запитаних за допомогою умови запиту, і перевіряє, чи збігаються розпаковані діапазони обчисленого діапазону зміщення. Оператор підмножини завжди завантажує інформацію про позицію, зчитуючи множини зі своїх дочірніх елементів. Оператор повертає значення агрегатів всіх елементів вхідного масиву. Він просто перебирає всі непорожні елементи і виконує необхідну агрегацію. При

виконанні позиційно-залежних операцій над індексованим набором даних, таких як множення матриць або згортка, може бути бажано перетворити набір даних на звичайний поділ. Аналогічно, прості множині потрібно перетворити на індексовані множині під час завантаження даних. Система надає оператори для цієї потреби. Оператор приймає вхідний набір даних як вхід і перетворює його на звичайний набір даних з тією ж формою сегмента. Наївна реалізація може виділяти простий поділ, перебирати всі індексовані множини, що перекриваються, один за одним, і де треба стиснути значення до їх правильних позицій. Однак це шкодить пам\яті, і додає значного тиску на систему виділення пам'яті і підкачки. Замість цього реалізація конструює і повертає простий поділ в одиниці підмножини, кожен з яких містить фіксовану кількість суміжних елементів в множині. Внутрішньо, метод працює подібно до об'єднання злиття: він читає наступні сегменти всіх перекритих бітів, і виконує операцію злиття, оскільки наступний сегмент запитується батьківським оператором. Для обробки порожніх елементів він використовує бітовий вектор, щоб відстежити, які елементи не існують у всіх бітах, і повертає бітовий вектор разом з даними. Метод показує, що реалізація на основі ієрархії покращує вбудовану реалізацію. Оператор є оберненою операцією. На додаток до вхідного набору даних, йому також потрібно кількість бітів, діапазони кожного біту, а також кількість бітів у кожному фрагменті як параметри. Для кожної простої множини оператор використовує двійковий пошук для розміщення його значень у правильній підмножині, і комбінує біти для складання індексованих множин.

Порівняємо із запитом даних приблизно на окремому растровому індексі, які компроміси здійснено з точки зору точності та часу запиту. Також, наявність можливості при необхідності ефективно перетворити відфільтрований результат індексованого набору даних на просту репрезентацію. Розглянемо параметри поділу, які підходять для індексованого набору даних. Та оцінюємо реалізацію на платформі. Дані зберігаються на локальному жорсткому диску. Реалізація системи на мові C++. Система не реалізує саме управління кешем і спирається на кеш на рівні ОС і сховища. Очищаємо кеш до того, як всі експерименти були виміряні, час

виконання. Експериментально порівнюється продуктивність тих самих запитів на індексованому масиві з виконанням звичайних наборів даних. Наскільки відомо, не існує механізму зберігання масивів з доступними індексами вартості. Щоб краще проілюструвати продуктивність системи, також порівняємо її з популярною базою даних масивів, чий механізм зберігання даних показав подібну продуктивність вводу-виводу порівняно з популярними науковими форматами файлів. Як синтетичні, так і реальні набори даних використовуються для оцінки продуктивності запитів системи. Синтезований набір даних, який використовували, рівномірний, являє собою двовимірний набір даних подвійних чисел з рівномірним розподілом на його домені значень. Два реальних набори даних взяті. Перший набір містить ретроспективну та перспективну проекцію з кількома моделями. Перший набір даних, містить використання моделі. Набір даних можна розглядати як однопоточний масив з нестисненим розміром. Другий набір даних, що містить прогнозовану кількість є однопоточним масивом з нестисненим розміром. Встановлюємо розмір множини таким чином, щоб кожен сегмент містив приблизно 364 МБ даних. Якщо не буде іншого, то використовуємо 1000 бітів еквівалентної ширини для індексованих наборів даних та автономних растрових індексів, а також зберігаємо 14 бітів в індексованому фрагменті. Вибір параметрів поділу ґрунтується на експериментах.

Вибираємо агрегацію з умовами вибору діапазону, оскільки це один з найпоширеніших запитів в аналітиці, і його ефективність є репрезентативною для різних запитів, що виконують індексоване сканування. Для перевірки продуктивності як вибору на основі значень (фільтр), так і вибору на основі розмірів (підмножини), використовуємо кілька умов вибору. Для запитів на основі значень продуктивність запиту може бути пов'язана як з тим, наскільки великий домен запиту як частка домену значень (діапазон доменів), так і з кількістю елементів, які запитуються. Обидва вищезазначені як відсотки від вихідного набору даних. Невеликий обсяг пам'яті скорочує час вводу-виводу і часто призводить до меншого загального часу відповіді на запит, якщо введення-виведення є вузьким місцем. Досліджуємо слід зберігання залишкових бітів

методів стиснення. Також оцінюємо, чи використання растрового стиснення або стиснення перевернутого списку є більш ефективним для зберігання позиційного індекс. Для методів вибираємо розмір хеш-таблиці та встановлюємо довжину зсуву вліво/вправо для кодерів на 4/16 та 2/12 біта відповідно. Тестуємо ці методи на різних наборах даних, а також зразки, взяті з двох реальних наборів даних, які використовували. Розмір стислих залишкових даних та позиційний індекс задамо у відсотках від вихідного набору даних. Методи з найкращим ступенем стиснення відзначені шрифтом. З точки зору залишкового стиснення, немає явного переможця. Вони є двома методами з найкращими коефіцієнтами стиснення. Хороша перфорація цих методів вказує на важливість для методу залишкового стиснення ефективно кодувати різницю в залишках. В цілому, перший метод здається найкращим вибором;. Навіть коли він не досягає найкращого ступеня стиснення. Різниця зазвичай знаходиться в межах одного або двох відсотків від вихідного розміру даних. Оскільки перший метод захоплює більше інформації та кодує залишок більш ефективно, він також покращує ступінь стиснення в середньому в 1,5x, іноді до 4,4x. Метод відображення є більш ефективним, і це залежить від використовуваного методу кодування: непідписане перевертання краще працює з методом, тоді як видалення префіксів краще працює з кодувальником на певних наборах даних. Що стосується стиснення індексів, то інвертований список працює значно краще, ніж растровий малюнок в цілому, за винятком наборів даних з висококонцентрованими значеннями. Це пов'язано з тим, що растрові індекси, як правило, оптимізовані для швидких операцій перетину та об'єднання, а не для оптимізованого розміру сховища. Оскільки зіставлення збереженого зміщення з його залишковим значенням, а використання бітового зображення як індексу також потребує додаткового відстеження, то перевернутий список, як правило, більше підходить для зберігання позиційного індексу.

Згідно цих спостережень стиснули залишкові дані за допомогою першого методу, а позиційний індекс за допомогою другого методу в наступних експериментах. Це показує комбінований обсяг зберігання проіндексованого набору даних. Навіть з додатковим індексом все одно досягаємо кращого обсягу

пам'яті порівняно з оригінальним набором даних, що ілюструє ефективність схеми індексованого зберігання. Оскільки набори даних зазвичай оновлюються нечасто, час створення індексу є одноразовою вартістю, а не головною турботою. Отже, до продуктивності індексного сканування системи розроблено відповідні методи. А виконання запиту на суму фільтра для індексованих наборів даних з виконанням тієї ж операції на звичайних наборах даних та оцінкою запиту. Продуктивність запиту фільтр-сума на однорідному наборі даних при зміні селективності від 0,1% до 100% є достатньою. Час відповіді на запит фільтрації індексованого набору даних пропорційний вибірковості, за винятком випадків, коли вибірковість нижча за 4%. Це пов'язано з тим, що механізм зберігання завантажує весь поділ, навіть якщо в ньому є доступ лише до однієї підмножини. Це вказує на те, що дисковий ввід/вивід завантаження даних, а не вартість розпакування, є фактичним вузьким місцем. Індексоване сканування також показує значну перевагу перед повним скануванням на звичайному наборі даних, час відгуку якого не змінюється з вибірковістю. При виборі 10% елементів операція фільтра на індексованому наборі даних дорівнює ~ 8, що у 7 разів швидше, ніж на звичайному наборі даних. Кожен біт в уніфікованих наборах даних містить однакову кількість елементів. Щоб проілюструвати ефект розподілу даних, виконуємо той самий запит на реальному наборі даних, який має розподіл, подібний до звичайного розподілу зі значеннями, зосередженими в середньому діапазоні. Час відповіді на запит вибраний як діапазон запиту відносно щільного та розрідженого відповідно. Час відгуку майже пропорційний кількості даних, що читаються. Для порівняння, фільтрація індексованого набору даних перевершує фільтрацію простого набору даних принаймні в 3 рази. Перевага індексованих даних зростає ще більше, коли вибрано менше елементів, а коефіцієнт прискорення досягає максимуму. При фільтрації набору даних, більшість даних якого концентрується всього в декількох околах, індексоване сканування показує аналогічну значну швидкість. Навіть коли всі елементи виділені, фільтрація індексованого масиву все одно відбувається швидше на всіх синтетичних і реальних наборах даних. Це пов'язано з тим, що розроблена схема зберігання ефективно стискає набори даних, додаючи обмежені додаткові

витрати на декомпресію: індексований набір даних майже на 70% менший порівняно зі звичайним набором даних, але для розпакування та агрегації індексованого набору даних потрібно лише близько 25% додаткового часу процесора. Профіль даних показує, що близько половини процесорного часу йде на розпакування залишкових бітів, особливо на оновлення таблиці прогнозування та обчислення фактичних значень, оскільки вони не можуть бути векторизовані ефективно, що вказує на подальше підвищення ефективності декодування, що є ключовим для отримання кращої продуктивності в пам'яті. Індексований набір даних, так і звичайні набори даних перевершують у всіх сценаріях експериментів. Це пов'язано з тим, що оператор реалізовано загальним чином, без спеціальної оптимізації під використовуваний тут запит фільтра діапазону. Отже, це приносить значні додаткові витрати на процесор. Операція підмножини все ще може бути виконана на індексованому наборі даних так само ефективно, як на звичайному наборі даних. Оцінюємо це, виконуючи запит для набору даних. Вибираємо регіон, що займає 1/8 обсягу набору даних, і варіюємо селективність від 0,1% до 100%. Запит індексованих наборів даних все ще зберігає свою перевагу в продуктивності як над звичайними наборами даних, так і над масивами. Це пов'язано з тим, що індексований набір даних все ще використовує регулярне фрагментування для прискорення вибору на основі розмірів, а підфрагментне виділення в граничних фрагментах також може бути виконано відносно швидко.

Різні оператори агрегації можуть мати різну схему продуктивності. Для оцінки протестовано продуктивність запиту з тими самими параметрами. Виконання запиту до індексованого набору даних більш швидше, ніж виконання його на звичайному наборі даних. Оператору потрібно лише прочитати підмножини та фрагменти, які можуть містити менші значення, ніж поточне мінімальне значення. Продуктивність індексованого методу покращується, коли діапазон запитів збільшується. Це пов'язано з тим, що дані розподіляються подібно до звичайного розподілу, тому, коли діапазон запитів збільшується, мінімальна підмножина і фрагмент містять менше даних.

Здійснимо порівняння з приблизним методом агрегації. Наближені методи підвищують продуктивність, жертвуючи певним рівнем точності. Перспективний наближений метод агрегаційних зображень передбачає використання растрових зображень як представлення. Використовується растровий алгоритм стиснення, як широко використовуваний варіант відомого алгоритму з акцентом на швидкість декомпресії. Досліджуємо точність і час відгуку запиту при різних налаштуваннях Використовуємо для цього набір даних і все одно встановлюємо область підмножини на 12,5%. Відносна похибка виконання фільтра-підмножини суми та фільтра-підмножини часу приблизно з використанням растрових зображень однакова. Ширина діапазону запитів встановлюється на 1,5% і 15% від доменного діапазону, а його нижня межа переміщується через діапазон доменів. Хоча мінімальна агрегація може бути виконана досить точно На бітовому зображенні з прив'язкою однакової ширини виконання запиту суми приблизно в доменному діапазоні 1.5% майже завжди призводить до відносної похибки більше 50%. Наближений метод все ще призводить до середньої відносної похибки, навіть якщо селективність збільшується до 15%. Хоча деякі методи можуть підвищити точність, ці методи зазвичай призводять до значного збільшення часу побудови індексу і можуть вимагати апріорних знань щодо розподілу діапазону запитів. Навпаки, фільтрація індексованого набору даних повертає точні результати незалежно від стратегії, показуючи надійність методу індексування. Ще одним способом підвищення точності растрових методів є збільшення кількості бітів. Точність на обох рівнях відбору значно покращується. Однак розмір растрового зображення також збільшується. Стислий растровий малюнок займає ~ 60% простору індексованого набору даних. Це зменшує прискорення приблизного методу. Навпаки, додавання більшої кількості бітів до індексованого набору даних не сильно змінює обсяг пам'яті, оскільки більш дрібнозернисте зв'язування також покращує коефіцієнт залишкового стиснення. Час відповіді запиту індексованого набору даних при збільшенні кількості бітів визначається двома факторами: додавання бітів зменшує обсяг непотрібних оброблюваних даних, але також збільшує додаткові витрати на обробку додаткових бітів. При цьому при низькому

рівні селективності дрібнозернисте зв'язування сильно скорочує непотрібні оброблювані дані і підвищує продуктивність. Однак на рівні вибірковості 15% збережений ввід/вивід був затінений накладними витратами на додані біти при використанні понад 100 бітів, збільшуючи час відповіді на запит.

Для порівняння, продуктивність використання індексованого набору даних зі 100 бітів порівняна з використанням 400-бітового растрового індексу в цьому випадку і завжди повертає точні результати, що робить його привабливою альтернативою для приблизного методу запиту.

Розглянемо перетворення індексованого фрагмента на звичайний фрагмент. Іноді може знадобитися перетворити індексований набір даних у його просту форму через оператор, щоб деякі операції можна було виконувати більш ефективно. Оцінюнимо продуктивність оператора. При цьому порівнюється час відгуку виконання операції фільтр-підмножина-сканування на звичайному наборі даних, та на індексованому наборі даних, але з використанням оператора для перетворення його в позиційне представлення. Перетворення індексованого набору даних у звичайний набір даних вимагає додаткового обчислювального часу через додаткові витрати на реорганізацію комірок масиву, тому бажано звертатися до індексованого набору даних за допомогою методу доступу на основі. Однак додаткові витрати обмежені, оскільки алгоритм на основі поділів уникає дорогої пам'яті. Час відгуку операції фільтра на 25% більше області підмножини уніфікованого набору даних, з різним розміром множини та кількістю бітів на поділ. Підмножини на поділ операції, яка шкодить локальності кешу. Ці додаткові витрати компенсуються зменшенням кількості вводу-виводу диска. Загалом, індексоване сканування прискорює запит із вибірковістю 10% у 3,9 рази. Його продуктивність добре масштабується з кількістю вибраних елементів і відповідає безпосередньому виконанню запиту на звичайному наборі даних. Розглянемо особливості параметрів поділу. Досліджуємо вибір параметрів поділу із заданою стратегією зв'язування. Вибираємо фільтр-підмножину-суму як типовий запит як зі значенням, так і з вибором параметра, вибираючи 1% або 50% доменного діапазону та 12,5% розмірного простору. Використовуємо 100 наборів еквівалентної ширини

з розміром сегмента ~ 16 Мб або ~ 64 Мб, і кількість нвборів на поділ варіюється від 1 до 32. Час відповіді на запит показує, що найкраща стратегія поділу залежить від розміру сегмента та вибірковості. Загалом, зберігання занадто великої кількості сегментів у множині збільшує непотрібний ввід/вивід і шкодить продуктивності. Однак час відгуку також не приносить великої користі від занадто дрібнозернистих розмірів множин, оскільки додаткові витрати на обробку метаданих і додаткові витрати на пошук затьмарюють зменшені введення-виведення даних. У налаштуваннях здається, що цей запит працює досить добре, коли розмір множини встановлений на порядку мегабайт. На основі цих спостережень встановили розмір сегмента як 64 МБ і використовували 4 підмножини на поділ в інших наших експериментах. Обидва методи є двома домінуючими форматами файлів масивів. Обидва формати зосереджені на переносимості та зручності використання, і не забезпечують жодної підтримки підмножини на основі вартості. Було докладено ряд зусиль для розробки більш багатофункціональних масивів. Вони реалізують зберігання масивів як розширення реляційних баз даних. Також, пропонується двошарова стратегія фрагментування, щоб збалансувати вимоги до процесора та вводу-виводу сховища масиву. Цей метод реалізує механізм зберігання спільного доступу з такими функціями, як реплікація та підтримка версій. Другий метод використовує стратегію розбиття фіксованого розміру і розкладає набір даних на кілька перекритих фрагментів, що дозволяє ефективно записувати і оновлювати розріджені масиви. Пропонований метод використовує стратегію регулярного поділу, широко використовувану цими системами, і показує, як індекс на основі вартості може бути реалізований у таких системах з обмеженими додатковими витратами. Багато сховищ масивів стискають фрагменти перед записом на диск, щоб зменшити розмір даних. У багатьох з цих систем використовуються компресори загального призначення без втрат. Однією з проблем є паралельне написання стислих наукових файлів. Загальні компресори з плаваючою комою важко спроектувати. Він використовує комбінацію двох предикатів для генерації залишків і використовує варіантні байти для стиснення залишку. Попередній

кондиціонер визначає байти, що стискаються потоку чисел з плаваючою комою, і стискає їх лише для покращення загального ступеня стиснення. Два методи зберігають дані моделювання на місці з точним індексом та злитим кодуванням. У цих системах дані реорганізовуються в біти на основі його k байтів вищого порядку, а інвертований індекс відстежує позиції значень у кожному біті. Нижні байти значень у кожній підмножині потім стискаються і зберігаються окремо. Однак кількість бітів і їх діапазони в цих системах визначаються кількістю унікальних байт високого порядку на вході, обмежуючи прискорення за рахунок індексу, і зменшуючи гнучкість його індексу. Ці системи також не розглядають, як включити таке плавлене кодування в багатовимірне сховище масивів, зберігаючи при цьому можливість виконання ефективних операцій підмножини. В системі показано, як ідея злиття даних та кодування індексів може бути використана в системі обробки масивів з підтримкою довільного прив'язки. Вона представляє ефективну бітову залишкову схему стиснення та дизайн гнучкого механізму зберігання, здатного обробляти різні запити як на простому, так і на індексованому наборі даних. Розглянемо особливості растрового індексу і приблизної обробки запитів. Ефективне стиснення растрового індексу зменшує обсяг пам'яті, а також час запиту. Популярні методи використовують кодування довжини виконання для ефективного кодування безперервного заповнення 0 або 1 заповнення. Растрове зображення розбиває довге растрове зображення на менші множини і зберігає кожен фрагмент або як бітовий вектор, або як відсортований цілочисельний масив індекси елементів, заснований на щільності заданих бітів в множині. Ці методи використовують мультирастрові вектори, щоб дозволити оновлення растрових індексів. Растрові індекси та їх варіанти ефективні в реляційних навантаженнях у стилі OLAP, особливо коли кардинальність стовпця низька. Вони підтримують прискорення наукових запитів щодо існуючих форматів файлів за допомогою зовнішніх растрових індексів. Растрові зображення також можуть використовуватися для відповіді на приблизні запити в наборі даних без посилання на вихідні дані. Точність таких запитів часто залежить від наших апріорних знань про розподіл даних і запитів, а також від використовуваної стратегії. Об'єднуючи

дані та індекси, система зменшує надмірність зберігання зовнішніх растрових індексів, підвищує ефективність вводу-виводу та забезпечує точні результати незалежно від запитів та стратегії прив'язки. Розглянемо інвертовані списки та стиснення даних. Інвертовані списки є широко використовуваною структурою даних в інформаційно-пошукових системах, і зазвичай зберігаються у вигляді стиснення. Методи стиснення можна умовно розділити на три категорії: бітове стиснення; гамма-кодування; байт-орієнтоване стиснення. А також словесно-орієнтовані кодери, такі як різні компресори. Хоча інвертовані списки і растровий індекс служать аналогічній меті, останні широко використовуються в інформаційно-пошуковому контексті, в той час як перші широко використовуються базою даних. До недавнього часу не було багато дискусій про схожість цих двох структур даних. Вони пропонують використовувати растрові індекси для прискорення роботи перетину двох списків проводок. Останнім часом наведено дуже детальне порівняння між двома структурами. Існують підходи для безпосереднього аналізу стислих даних в інших контекстах, які можна розглядати як таку ж широку категорію, як проєктована система.

Таким чином, проєктована системв – система зберігання масивів з інтегрованою підтримкою індексу вартості. Система реорганізовує елементи в ряд користувацьких бітів і ефективно кодує індекси на основі бітів і відповідні їм значення, генеруючи індексоване представлення масиву, яке додає мало додаткових витрат на зберігання.

2.2 Загальні концепції об’єднання індексованих масивів

Система є компактним сховищем масивів з індексом значень. Хоча вона добре обробляє такі запити, як фільтрація та підмножина, є цінним для розгляду більш складних запитів обробки масивів і способів обробки таких запитів. Це може отримати вигоду від покращеної продуктивності вибору на основі вартості. Розглядатимемо один з найбільш використовуваних типів запитів масивів – об'єднання масивів. Об'єднання масивів включає в себе порівняння двох масивів і

пошук відповідних або схожих пар комірок. Приклади об'єднання масивів включають пошук помилкових датчиків, які продовжують повідомляти однакові показники, знаходження спільних прогнозів між різними прогнозними моделями або знаходження сусідніх фрагментів. Такий запит може бути виражений об'єднанням масиву за допомогою запиту. Крім того, у багатьох з цих випадків користувачів цікавить лише невелика підмножина всього масиву – наприклад, користувач може захотіти знайти значення (наприклад, менше −20), які були точно передбачені, оцінити потужність прогнозної моделі. Такі запити вимагають, щоб оператор об'єднання масиву ефективно обробляв предикати фільтрації значень. Крім того, оскільки дані часто мають дисперсію, порівняння між клітинами часто не є точним і потребує терпимості до невеликих неточностей, що вказує на необхідність приєднання подібності. Однак, хоча попередні роботи існують як для рівно об'єднання масивів, так і для об'єднання подібності масивів, вони в основному будуються на сховищах масивів, організованих навколо просторових розмірів. Наслідком цього є те, що їм доводиться сканувати весь масив при роботі з предикатами вибору на основі значень, витрачаючи введення-виведення та обчислювальну потужність. Крім того, розглядали об'єднання подібності на основі розмірності, яке знаходить сусідні комірки в масивах, які мають однакові значення, при цьому не розглядали об'єднання подібності на основі значень, яка знаходить клітинки зі схожими значеннями, незалежно від їхнього розташування. Обмеження, викладені вище, пов'язані з тим, що представлення масивів більш ефективно при запиті через розмірні координати, а не значення комірок. Одним із способів вирішення проблеми є використання нового представлення масиву, яке включає інформацію про значення комірки в організації. Розглянемо оператор з'єднання масивів, розроблений для ефективної роботи на малій (на основі значень) підмножині масиву. Окрім об'єднання, він також підтримує як ціннісне, так і розмірне об'єднання, що робить його придатним для аналізу зашумлених числових даних. Він оперує індексованим представленням масивів системи. Під час об'єднання використовує індекс для зменшення непотрібного завантаження клітинок та порівняння. Оскільки індексний масив можна ефективно фільтрувати

на основі значення, він повторно зменшує загальний ввід/вивід, необхідний під час об'єднання. Крім того, оскільки потрібно порівнювати лише значення в одному кошику, а індекс можна використовувати для прискорення порівняння, індексоване об'єднання також скорочує час порівняння клітинок, забезпечує кращу загальну продуктивність, особливо коли вибірковість низька. Експериментально оцінено оператор приєднання як на синтезованих, так і на реальних наборах даних. Результат показує, що він в кілька разів швидше працює при об'єднанні невеликих підмножин доменів вхідних масивів, і зберігає конкурентоспроможні показники навіть при рівні селективності 50% і вище. Об'єднання подібності на основі розмірів також значно прискорює обробку запитів, зменшуючи додаткові витрати як на процесор, так і на ввід/вивід. Оператор об'єднання масивів для представлень індексованих масивів, який також використовує індексоване представлення масивів для прискорення обробки об'єднання на підмножинах масивів малих значень-доменів як зменшення порівняння клітин є ефективним. Представляємо перший наскрізний фреймворк реалізації об'єднань масивів з підтримкою для об'єднань подібності як на основі значень, так і на основі розмірів. Наскрізна оцінка показує, що він значно прискорює об'єднання на невеликих підмножинах масивів і прискорює об'єднання розмірності на всіх рівнях селективності на порядок і більше. Розглядатимемо визначення об'єднань масивів. Подібно до свого аналога в реляційних базах даних, об'єднання масиву ідентифікує комірку комбінації між розмірами і атрибутами. Це задовольняє критерію приєднання, логічній функції на розмірності та атрибутам двох комірок. Загальність цього визначення вказує, що можлива кількість комірок в згенерованому масиві може бути великою. Коли функція завжди істинна, результатом буде масив, який буде складатися з декартового добутку всіх комірок, що робить зберігання результатів недоцільним. Також кількість його розмірів буде сумою розмірів двох вхідних масивів. Оскільки сховище масиву організовано навколо його розмірів, це створює додаткові труднощі в ефективному представленні та запиті результатів. В реальності цікавлять клітинні пари обмежені умовами з'єднання і кількість пар зазвичай набагато менше в порівнянні з декартовим добутком на всі розміри в

об'єднувальних масивах. Тому, схему виходу можна коригувати, видаляючи повторювані розміри і атрибути, включаючи, можливо, зменшення певних розмірів на менші. Основну увагу приділимо трьом важливим категоріям об'єднань масивів. Всі три типи встановлюють різні обмеження на з'єднувальні предикати.

Перший з них знаходить пари комірок з рівними вимірними координатами та атрибутом valus. Приклади запитів включають пошук помилкових показань датчиків, які не змінюються між різними записами або іншим чином однакових спостережуваних елементів у різних спостереженнях. Через властиву проблему точності, пов'язану з числовими додатками, також потрібно здійснювати пошук клітин з подібними атрибутами. Замість того, щоб шукати атрибути, які точно збігаються, об'єднання подібності значень або об'єднання взаємного діапазону шукає атрибути, які знаходяться на певній відстані, визначена функція відстані. Одним із прикладів таких запитів є порівняння результатів прогнозування різних моделей. Потрібно знайти клітини, які різні моделі мають схожі (або різні) прогнози. Однак малоймовірно, що прогнози двох моделей прогнозування абсолютно однакові. Замість цього потрібно досліджувати схожі значення. Третім типом об'єднання є об'єднання розмірної подібності, яка порівнює клітинки координати яких знаходяться в межах певної відстані, задавши визначену користувачем відстань. Ці типи запитів корисні під час пошуку схожих об'єктів у різних списках або миттєві, наприклад, знаходять одну і ту ж (повільно) рухому клітинку.

Функція об'єднання символів у також повертає пари комірок у відповідному положенні з різницею значень. Подібне об'єднання між масивом повертає пари комірок, що збігаються, відстань до яких не перевищує 1. Тому що результатом об'єднання є чотиривимірний масив, де є лише відповідні комірки центральної комірки в масиві для спрощення представлення.

Розглянемо загальні рамки об'єднання індексованих масивів. Розглядатимемо загальну структуру для об'єднання двох індексованих масивів. Спочатку розглянемо процедури об'єднання двох простих масивів, а потім

коригування, необхідне для об'єднання двох індексованих масивів. На рис. 2.1 зображено схему об’єднання індексованих масивів.

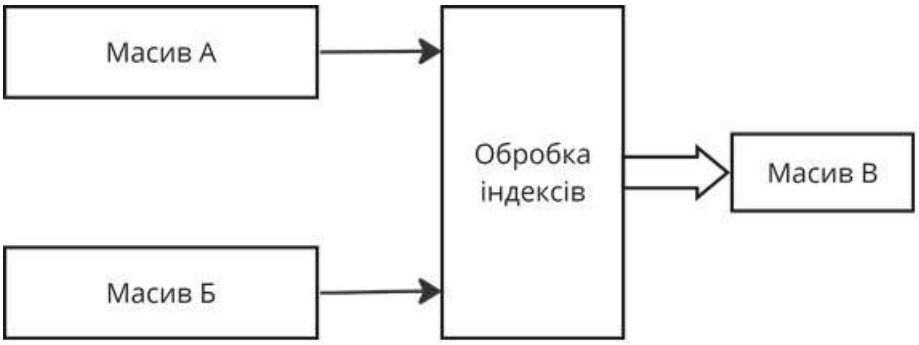


Рисунок 2.1 - Схема об’єднання індексованих масивів

Першим кроком об'єднання двох простих масивів є визначення схеми вихідного масиву. Розміри вихідного масиву можуть бути стільки ж, скільки сума вимірів у двох вхідних масивах. За типом і параметрами об'єднання, ці розміри часто можна згорнути (якщо потрібно дорівнювати двом вимірам) або стиснути (якщо одне значення виміру завжди знаходиться в межах невеликої відмінності від іншого значення параметра), що спрощує обробку стику. Фреймворки зберігання масивів організовують дані відповідно до їх розмірних координат. Таким чином, комірки можна порівняти за лінійний час, якщо комірки у вхідних фрагментах сортуються в тому ж порядку, що і вихідний фрагмент. Отже, після визначення схеми може бути корисно перегрупувати вхідні масиви, щоб її фрагментація була такою ж, як і фрагментація вихідного масиву і, таким чином, простіше в обробці.

Останнім кроком є з'єднання сегментів. Для кожного можливого вихідного фрагмента алгоритм об'єднання отримує можливі сегменти з обох вхідних масивів, що містять потенційно відповідні пари комірок, знаходить можливі відповідні пари комірок у цих сегментах та порівнює комірки, щоб побачити, чи задовольняють вони предикатам об'єднання. Можливі відповідні комірки одного вхідного фрагмента охоплюють кілька сегментів, у цьому випадку кожну пару сегментів потрібно об'єднати окремо. Фрагменти можуть бути об'єднані за допомогою

сортування в залежності від типу і параметрів конкретного запиту на об'єднання. Вихідний сегмент може бути оброблений оперативно до сховища.

2.3 Висновки

В результаті представлено метод доступу на основі фрагментів, придатний для обробки індексованих даних масиву, і відповідні стратегії обробки запитів. Сховище індексованих масивів виконує точні операції фільтрації незалежно від обраної користувачем стратегії прив'язки і зберігає ефективність операцій підмножини в традиційні сховища масивів. Операція фільтрації на індексованому наборі даних масштабується з кількістю вибраних елементів і перевершує просте представлення послідовно, навіть якщо рівень селективності відносно високий. Відфільтрований результат індексованого набору даних також може бути ефективно перетворений назад у просте представлення, якщо це необхідно. В подальшому потрібно розглянути включення в систему того, як реалізувати більш складні операційні пристрої поверх методу індексованого доступу і як індексований набір даних може бути динамічно оновлений, і як стратегії та параметри прив'язки можуть бути автоматично обрані.

Таким чином, удосконалено метод перевірки трансформації кешу комп’ютера.

3 МЕТОД ПРИСКОРЕННЯ ОБ’ЄДНАННЯ МАСИВІВ З ІНТЕГРОВАНИМ ІНДЕКСОМ ЗНАЧЕНЬ

3.1 Об’єднання індексованих масивів

Загалом, структура об'єднання індексованих масивів відповідає загальним крокам об'єднання простих масивів. Ключові відмінності полягають у тому, що індексовані масиви не тільки організовані за розмірними координатами, але й організовані з різними бітами, що зберігають значення атрибутів. Зернистість обробки, таким чином, становить вже не сегменти, а підмножини в кожній множині. Аналогічно, оператор перебудови також повинен мати можливість вирівнювання прив'язки. Алгоритм дає огляд об'єднання двох індексованих масивів. Алгоритм починається з виведення вихідної схеми та вирівнювання вхідних масивів відповідно до схеми. Після цього процесор перераховує сегменти в лівому бічному масиві та підмножини всередині кожного масиву. Для кожної підмножини він отримує підмножини (можливо, більше однієї) у правому масиві, які могли б відповідати коміркам відповідно до конкретного типу та параметрів з'єднання. Передбачається, що форми двох вхідних масивів завжди однакові. Це пояснюється тим, що потрібне ефективне виконанню запиту на об'єднання з низькою вибірковістю з використанням індексованих масивів. Для вирішення проблеми ефективного об'єднання масивів з різними схемами фрагментування потрібна реалізація різних типів об'єднання. Рівнооб'єднання відповідає коміркам з однаковими розмірами та атрибутами. Для масивів, що не відображаються в критеріях приєднання, отримують декартів добуток комірок збігу. Це можна розглядати як окремий випадок об'єднання розмірної подібності, тому проєктуватимемо умову з'єднання, які містять усі виміри. У цьому випадку вихідний масив має ту ж форму, що і вхідні масиви. Розглянемо випадок, коли сегментування і зв'язування двох з'єднувальних масивів однакові. У той час як прості сегменти потрібно об'єднати, перевіряючи комірки у відповідних позиціях один за одним, об'єднання між двома індексованими масивами може бути виконано шляхом об'єднання відповідних сегментів. Це двоетапний процес. Спочатку

об'єднуємо позиційний індекс двох інтервалів, а потім порівнюємо значення в відповідних позиціях. Реалізація перетину позиційних індексів залежить від його представлення. Певні схеми кодування та побудова растрових зображень мають більш швидкий час перетину. Зіставляючи значення, ділимо залишкові значення на менші сегменти і розпаковуємо підмножини лише тоді, коли це необхідно. Як просте, так і індексоване рівно'єднання вимагає одноразового сканування відфільтрованих вхідних масивів. Однак індексовані масиви можна ефективно фільтрувати, не читаючи зайвих полів. Отже, індексований оператор об'єднання набагато ефективніший при об'єднанні підмножини вихідних масивів. Якщо схеми прив'язки двох вхідних масивів відрізняються, то одну сторону масивів потрібно перепрофілювати, щоб вона відповідала схемі іншої. Це можна зробити за рахунок одного додаткового сканування масиву, що перебудовується. Спочатку розраховується нова схема зв'язування відповідно до зв'язування і порогу відстані. Потім масив перебудовується. Кожна структура в масиві потім зіставляється з усіма сегментами, що містять можливі пари комірок.

Об'єднання подібності значень знаходить пари комірок з однаковими розмірами та подібними значеннями атрибутів (тобто в межах визначеної користувачем функції відстані). Вихідний масив має ту ж форму, що і вхідні масиви. У разі з'єднання досить порівняти біт із відповідним бітом на іншому вхідному масиві. Однак, якщо значення подібності об'єднуються, зіставлені значення також можуть існувати в суміжних бітах. Завдання полягає в тому, щоб зменшити непотрібне повне сканування на сусідніх стовпцях. З цієї причини повторно розбиваємо біти однієї сторони з'єднаних масивів наступним чином. Перерозподілені біти визначаються зв'язуванням іншого вхідного масиву та визначеною функцією відстані. Без втрати спільності, припустимо, що повторно розбиваємо вхідний масив за допомогою бітів. Просто можна отримати, виконавши злиття між стовпцями. Алгоритм проходить через процес об'єднання за подібністю значень. Алгоритм починається з ініціалізації вихідного масиву відповідно до вхідної схеми. Потім він вибирає меншу сторону масиву для перерозподілу. Без втрати спільності, припускаємо, що менший масив дорівнює частині більшого.

Потім він використовує алгоритм для визначення нового зв'язування масиву та сканує масив для виконання операції переформування. Після цього оператор сканує масив з сегменту за сегментом, структуру за структурою. Для кожної підмножини він отримує структуру, яка містить можливі відповідні значення, знаходить перетин значень в обох сегментах, а також перевіряє, чи задовольняє він умовам об'єднання, і виводить відповідні пари клітинок. У порівнянні зі звичайними масивами, об'єднання значень подібності до індексованих масивів вимагає одного додаткового сканування для розбиття однієї сторони вхідних даних і сканування граничних сегментів кілька разів. Однак, оскільки індексовані масиви ефективно зменшують ввід/виведення, коли запит шукає лише невелику частину домену значень, він все ще має значну перевагу, коли вибірковість низька. Розглянемо об'єднання розмірної подібності. Враховуючи два масиви, визначену користувачем функцію відстані, і поріг відстані, об'єднання розмірної подібності знаходить пари комірок з розмірною відстанню меншою за порогові та відповідні значення. Тоді розглядається, як такий запит може бути оброблений в індексованому сховищі масивів. Можна визначити форму вихідного масиву, а потім як можна реалізувати об'єднання подібності.

Першим кроком є визначення форми вихідного масиву. Об'єднання рівноцінності та ціннісної подібності зосереджується на клітинах із відповідними елементами. Таким чином, відповідні розміри можна не вказувати, і в результаті вийде вихідний масив з однаковими розмірами. Однак, подібно до об'єднання, комірку з одного боку можна зіставити з кількома клітинками з іншого боку. Отже, кількість вимірів у результуючому масиві є сумою вимірів у вхідних масивах. Просте поєднання розмірів вхідних масивів як вихідної схеми створює великий розмірний простір. Оскільки дана клітинка в масиві, у результуючому масиві існують лише клітинки масиву з відстанню меншою за задану. Таким чином, замість того, щоб зберігати розмірність безпосередньо, можемо замість цього зберігати різницю розмірів. Таким чином, в різницеві розміри потрібно лише розмістити розмір максимальної граничної рамки.

Реалізація об'ємної подібності об'єднання. Загальна процедура обробки виглядає наступним чином: перебираємо комірки на одній стороні входів, посилання як сторона збірки. І для кожної комірки перевіряємо можливі комірки на відстані на іншій стороні входів (масив, іменований стороною зонда), перевіряємо, чи збігається значення. Враховуючи сегмент на стороні збірки, оператор індексованого приєднання починає з пошуку сегментів, які можуть містити відповідні комірки. Ця процедура подібна до оператора простого з'єднання: для сегмента на стороні карти можуть бути сегменти в декількох сегментах, що містять потенційні комірки, що збігаються. Які підмножини на стороні зонда мають відповідні комірки, залежить від функції відстані та порогу відстані і можуть не бути легко визначеними. Для простоти механізм запитів обчислює (можливо, приблизно) максимальну обмежувальну рамку відповідних клітинок у зондувальній стороні та знаходить усі сегменти (можливо, з декількох сегментів), які перетинаються. Це робиться шляхом запиту індексованого дерева з фрагментованою інформацією. Після отримання зацікавлених сегментів механізм запитів перевіряє наявність відповідних пар клітинок. Подібно до розмірного об'єднання на звичайному операторі, це можна зробити за допомогою простого вкладеного циклічного об'єднання або індексного об'єднання. Індексоване об'єднання будує багатовимірний індекс його елементів, а потім досліджує індекс для зіставлення комірок у межах розмірної відстані. Цей індекс може бути або багатовимірною структурою індексу, такою як дерево або хеш-таблиця. У реалізації використовуємо комбінацію двох стратегій, тобто ділимо осередки на невеликі фрагменти, і будуємо індекс на цих фрагментах. Для кожної підмножини на стороні зонда робимо вкладений цикл з'єднання з потенційними підмножинами на стороні збірки. Оскільки на стороні збірки може бути сполучений з декількома стовпцями на стороні зонда, індекс для неї зберігається в пам'яті як кеш. Індекс видаляється з кешу після обробки всіх відповідних сегментів. У порівнянні з об'єднанням простих масивів, об'єднання розмірів не тільки економить час вводу-виводу, але і покращує час процесора. Це тому, що йому потрібно лише порівняти

кожну структуру з відповідними підмножинами. Отже, це зменшує обсяг пам'яті та час побудови та зондування структури індексу.

Оцінимо експериментально продуктивність виконання запитів на об'єднання масивів з використанням як синтетичних, так і реальних наборів даних. Для оцінки продуктивності об'єднання реалізуємо алгоритми об'єднання на основі індексованих алгоритмів та базових об'єднань поверх розробленої системи, механізму зберігання масивів з підтримкою як масивів з інтегрованим індексом значень, так і без нього. Очищаємо кеш до того, як всі експерименти були виміряні, час виконання. Експериментально порівнюється продуктивність тих самих запитів на індексованому масиві з виконанням запитів на простих масивах. Використовуємо регулярне розбиття і ті ж параметри фрагментування для простих масивів. Для об'єднання рівноцінного об'єднання та подібності значень також порівнюємо продуктивність запитів індексованого набору даних із методом приблизного об'єднання на основі растрового зображення. Продуктивність алгоритмів прискореного об'єднання оцінюється як на синтезованих даних, так і на реальних наборах даних. Синтезовані набори даних, які використовували, однорідні, являють собою два двовимірних 32-Гібайтових набори даних подвійних чисел з рівномірним розподілом в області його значень. Оскільки випадково згенеровані плаваючі дані майже завжди унікальні, Для об'єднання рівних та розмірних подібностей для отримання значущих результатів, дискретизуємо згенеровані набори даних до однозначної точності. Використані реальні набори даних є частиною. Проект публікує зменшені данні з кількома моделями і містить як ретроспективні, так і прогностичні прогнози . Набори даних, які використовували, відзначені як такі, що містять прогнози для результатів. Вибираємо проекції з двох різних моделей, як вхідні аргументи запиту приєднання, який відповідає реальному сценарію порівняння проекцій різних моделей. Обидва набори даних можна розглядати як 3105 × 7025 × 408 одноточний масив, з нестисненим розміром ~ 34ГБ. Встановимо розмір порції даних таким чином, щоб кожен сегмент містив приблизно 64 МБ даних. Якщо не вказано інше, то використовуємо 100 бітів еквівалентної ширини для індексованих наборів даних та

автономних растрових індексів, а також зберігаємо 4 біти в індексованому фрагменті. Вибираємо об'єднання запитів з операціями фільтрування за атрибутами як репрезентативні запити для оцінки. Такі запити показують як ефективність порівняння клітинок, так і здатність знижувати умови фільтра, і є загальними в сценарії аналізу даних. Фіксуємо час відповіді запитів від початку до кінця. Отриманий масив сканується, а не матеріалізується на диску. Для об'єднання значень подібності вибираємо абсолютну різницю з двох атрибутів як функцію відстані. Для об'єднання розмірної подібності в якості функції відстані вибираємо нормальну відстань між точками. Розглянемо продуктивність запиту рівно об'єднання на індексованих масивах. Оскільки оператор порівнював відповідні клітинки між двома вхідними масивами, то результуючий масив має ту саму форму, що й масив вводу, але з ще одним атрибутом. Спочатку розглянемо продуктивність об'єднання на синтезованих уніфікованих наборах даних. Масиви з умовою фільтра значення такі, що відображається час відгуку як на індексованих наборах даних, так і на простих наборах даних, а також виконання операції над вторинним растровим індексом, який може надати лише приблизні результати.

Як і очікувалося, рівні об'єднання на індексованих масивах досягають загального підвищення продуктивності порівняно з тим самим запитом на простих наборах даних. Рівно поєднання на індексованих масштабах наборів даних із селективністю фільтра умова, тоді як час запиту на звичайному наборі даних не змінюється, коли вибірковість змінюється. При вибірковості 10% відсотків індексоване об'єднання прискорює продуктивність більш, ніж в 5 разів. Це показує можливість для індексованого оператора рівно приєднання ефективно використовувати умову фільтра оператору сканування. При приєднанні до індексованого масиву комірки непотрібних сегментів не потрібно читати або порівнювати; тоді як оператору простого об'єднання масиву потрібно читати обидва масиви в пам'ять повністю. Оскільки оператор прив'язаний до вводу-виводу,а система зчитує дані в зернистості сегментів, прискорення не збільшується далі, коли селективність нижча, оскільки сегмент все одно потрібно прочитати з диска. Продуктивність об'єднання на індексованих масивах також виграє від

зменшеного вводу-виводу індексованого представлення. Це можна побачити в сценаріях більш високої вибірковості. Об'єднання масиву все ще більш ніж на 30% швидше, ніж об'єднання простих масивів, коли селективність становить 50%. Що стосується продуктивності реальних даних, то показано час відповіді на запит при об'єднанні двох наборів даних за допомогою різних методів. Схема продуктивності аналогічна: час відповіді на запит індексованого оператора з'єднання масштабується з селективністю, в той час як простий оператор підтримує постійну продуктивність. Оператор приєднання до індексу має дещо краще відносне прискорення на наборах даних, оскільки індексоване представлення краще обробляє розріджену область набору даних. Приєднання вторинних растрових індексів відбувається швидше, ніж звичайний та індексований оператор приєднання на обох наборах даних. Це більш ефективно для реальних даних, оскільки растрові індекси обробляють дані краще, ніж уніфіковані набори даних. Однак це дає неточні результати без гарантії ймовірності помилки. Цю неточність можна пом'якшити, виконавши перевірку меж, завантаживши вихідний набір даних і перевіривши, чи рівні значення, але цей крок є дорогим через додаткові витрати на введення-виведення повторного отримання вихідних наборів даних у зернистості сегментів. Виконання перевірки меж робить оператор ще повільнішим, ніж безпосереднє приєднання до простих масивів. Перевірка меж на реальних наборах даних буде працювати аналогічно.

3.2 Об'єднання подібності значень

Розглянемо продуктивність запиту об'єднання подібності значень в індексованих наборах даних. Як і у випадку з рівним об'єднанням, об'єднання подібності значень порівнює відповідні комірки за однаковими координатами розмірності, створюючи вихідний масив з однаковою формою.

Розпочнемо з розгляду подібності значень на двох синтетичних наборах даних. Порівняємо час відповіді на запит індексованих наборів даних з виконанням

того ж запиту на простих наборах даних. Також порівняємо продуктивність виконання того ж запиту на растровому зображенні вторинного показника.

Виконання об'єднання значень подібності за допомогою растрового індексу призводить до великої кількості помилкових спрацьовувань через прив'язку значень. Щоб усунути помилкові спрацьовування, можна виконати перевірку меж після знаходження відповідної пари з використанням растрових індексів. Продуктивність такого методу також порівнюється з показниками об'єднання подібності значень на однорідних наборах даних за допомогою методів з порогом.

Подібно до об'єднання, час відповіді на запит об'єднання подібності значень на простих наборах даних мало змінюється на рівні вибірковості, оскільки обчислення та введення-виведення не сильно змінюються. Але індексовані набори даних масштабуються з вибірковістю.

Загалом, індексоване об'єднання все ще значно швидше, коли селективність нижча. При селективності 10%, індексоване об'єднання має прискорення приблизно в 3 рази, а продуктивність індексованого об'єднання порівнянна з оператором простого об'єднання, коли селективність вища, на рівні 50% або 70% відсотків.

Швидкість об'єднання подібності індексованих значень дещо нижча, ніж виконання рівного об'єднання на тому самому наборі даних. Це стається через перебіг всього набору даних в індексованих наборах, а також порівняння одного біту у вхідному масиві з декількома бітами в іншому вхідному масиві.

Використання приблизного методу растрових індексів дозволяє досягти найшвидшої продуктивності, але він призводить до помилкових спрацьовувань. Виконання граничної перевірки значно знижує продуктивність до рівня, порівнянного з простим об'єднанням двох простих наборів даних. Це стається через те, що всі сегменти в простих масивах все ще повинні бути завантажені в пам'ять, що зменшує перевагу набагато меншої кількості введення-виведення, якою користуються растрові індекси.

Розбиваючи час на чотири частини - введення-виведення, сканування та фільтр, порівняння клітин, і перегрупування підмножини - ми можемо більш

детально аналізувати продуктивність кожного кроку в процесі об'єднання подібності значень в індексованих наборах даних.

Загалом, розглядана продуктивність запиту об'єднання подібності значень в індексованих наборах даних залежить від рівня селективності, методів виконання та властивостей даних. Дослідження та оптимізація цих аспектів можуть допомогти досягти кращої продуктивності при виконанні подібних запитів.

Об'єднання індексованого масиву економить значний час вводу-виводу та час сканування/фільтрування порівняно зі звичайним об'єднанням масиву, коли селективність низька, оскільки йому не потрібно сканувати сегменти зі значеннями поза умовою фільтра. Це також економить час порівняння клітинок, оскільки індексоване представлення масиву є більш ефективним для представлення розріджених масивів після операції фільтрації. Це пов'язано з тим, що оператор простого об'єднання повинен перевірити кожну комірку, щоб побачити, чи є вона порожньою коміркою або не. Якщо селективність зростає, перевага вводу-виводу індексованого масиву зменшується, а вартість перебудови збільшується, але індексований масив все ще зберігає порівнянну продуктивність при селективності 70%. Час відповіді запиту на уніфікованих наборах даних при зміні селективності на відміну від значення подібності об'єднання, індексований оператор приєднання перевершують оператора простого об'єднання в діапазоні селективності. Це пов'язано з тим, що селективності залежать від селективності об'єднання подібності процесора або вводу-виводу. Коли селективність нижча, об'єднання розмірної подібності пов'язане з введенням-виведенням. Зі збільшенням селективності вартість побудови та зондування структури індексу погіршує вартість сканування вхідного масиву, і оператор стає прив'язаним до процесора. На додаток до економії вартості вводу-виводу за рахунок більш ефективної фільтрації даних, оператор індексованого приєднання також може скоротити час процесора, оскільки потрібно порівнювати лише точки у відповідному сегменті, зменшуючи обсяг пам'яті та вартість побудови індексу та зондування. Отже, індексований оператор приєднання є більш ефективним, коли селективність є як нижчою, так і вищою.

Час відповіді на запит операторів об'єднання подібності у двох типових сценаріях селективності. Час розділений на чотири частини, крім часу вводу-виводу та часу сканування/фільтрації. Також записуємо час побудови та зондування структури розмірних індексів. Виконання об'єднання індексів економить час як процесора, так і вводу-виводу. Коли вибірковість збільшується до 70%, то індексований масив витрачає більше часу на побудову структури індексу, але витрачає менше часу на його зондування та загалом економить час відповіді на запит. Перевага продуктивності об'єднання індексованих наборів даних є більш значною при виконанні операції на реальних наборах даних. Об'єднання індексованих наборів даних відбувається на порядки швидше порівняно з об'єднанням простих наборів даних.

Вплив порогу відстані на значення та подібність розмірів є суттєвою. Розглянемо запит на об'єднання за схожістю значень. Для цього візьмемо запит для уніфікованих наборів даних, фіксуємо рівень селективності до 25%, змінюємо вибірковість від 1% ширини до 5x ширини та фіксуємо відповідний час відповіді на запит. Час відповіді на запит при об'єднанні простих масивів не змінюється при зміні порогу відстані, що очікується, оскільки не передбачає додаткових порівнянь. Об'єднання індексованих масивів передбачає зіставлення одного сегмента з більш ніж одним сегментом, який містить можливі збіги, тому для більшого порогу відстані йому потрібно порівняти більше елементів. Однак відповідь на запит лише незначно збільшується, коли поріг відстані перевищує 1x ширини бункера. Це пов'язано з тим, що коли рівень вибірковості не дуже високий, то ефективно фільтрує дані, використовуючи інтегральний індекс, тому час вводу-виводу домінує над часом виконання. Аналогічно для об'єднання розмірної подібності беремо запит на уніфікованих наборах даних, фіксуємо селективність до 25%, змінюємо відстань порогу від 1 до 9, і фіксуємо час відповіді відповідного запиту. Час відповіді запиту як для звичайного, так і для індексного з'єднання масивів збільшується зі збільшенням відстані. Однак час виконання об'єднання подібності на простих масивах збільшується набагато швидше. Це пов'язано з тим, що об'єднання подібності в індексованих наборах даних потрібне лише для порівняння

клітинок всередині одних і тих самих сегментів, отже, зі збільшенням порогу відстані йому потрібно враховувати меншу кількість кандидатів і він може ефективніше виконувати пошук індексу. Розбиваючи продуктивність розмірного з'єднання, коли відстань дорівнює 3 і 11, час порівняння клітинок як для звичайних, так і для індексованих масивів збільшується, але індексований масив збільшується набагато повільніше, забезпечуючи кращу продуктивність.

Масиви зберігання даних організовують дані в заздалегідь заданому багатовимірному порядку, дозволяючи ефективно запитувати визначені розміри. Такі системи будують сховища масивів поверх реляційних баз даних. Формати даних організовують дані масиву в щільні набори даних, які зберігаються горизонтально. Ці набори даних також зберігаються безперервно або в зернистості сегментів і упаковані у файли в підмножини. Однією з ключових проблем при зберіганні даних масиву є фрагментування. Різні схеми фрагментування, включаючи нерегулярне фрагментування та двошарове фрагментування для обробки розріджених даних масиву та балансування вимог процесора та вводу-виводу в обробці масиву є однією з найпопулярніших повноцінних масивних систем, яка пропонує сховище на основі стовпців, керування версіями і контролю дозволів. Це менеджер зберігання масивів з ефективним керуванням версіями та оновленнями розріджених масивів. Він використовує стратегію блоків фіксованого розміру всередині більших сегментів і розкладає кілька записів на кілька фрагментів, які об'єднуються за потреби. Ці системи зберігання масивів ефективні у виконанні запитів на основі діапазонів розмірів, але виконання предикатів на значеннях атрибутів не дуже ефективне. Вторинні індекси, такі як растрові зображення на основі бітів, використовуються для прискорення наукових запитів на існуючих системах зберігання масивів. Оскільки пошук вихідних значень на основі даних індексу може бути дорогим, також були запропоновані алгоритми, які запитують автономні растрові індекси. Однак, вони дають лише приблизні результати. Менеджер зберігання масивів, на якому ґрунтується реалізація, включає нову схему зберігання, яка реорганізовує сховище масивів відповідно до

розмірних координат та індексів значень на основі бітів, а також зберігає залишкові дані, що описують елементи всередині кожного біта.

Розглянемо об'єднання масивів. Більшість систем баз даних масивів підтримують однакові об'єднання масивів на масивах з однаковим фрагментуванням. Однак така підтримка часто обмежується об'єднанням комірок в одному положенні масиву і отриманням декартового добутку клітин і фільтра на основі умов з'єднання згодом. Оператор об'єднання є вкладеним циклічним об'єднанням, яке завантажує дані у відповідні сегменти та перебирає всі пари фрагментів/комірок. Оскільки запит розглядає ефективну обробку підмножини масиву, відфільтрованої значенням, то робота зосереджена на оптимізації порівняння комірок вводу-виводу на рівні вузлів за допомогою сховища масиву з інтегрованим індексом. На рис. 3.1 зображено основні кроки методу.

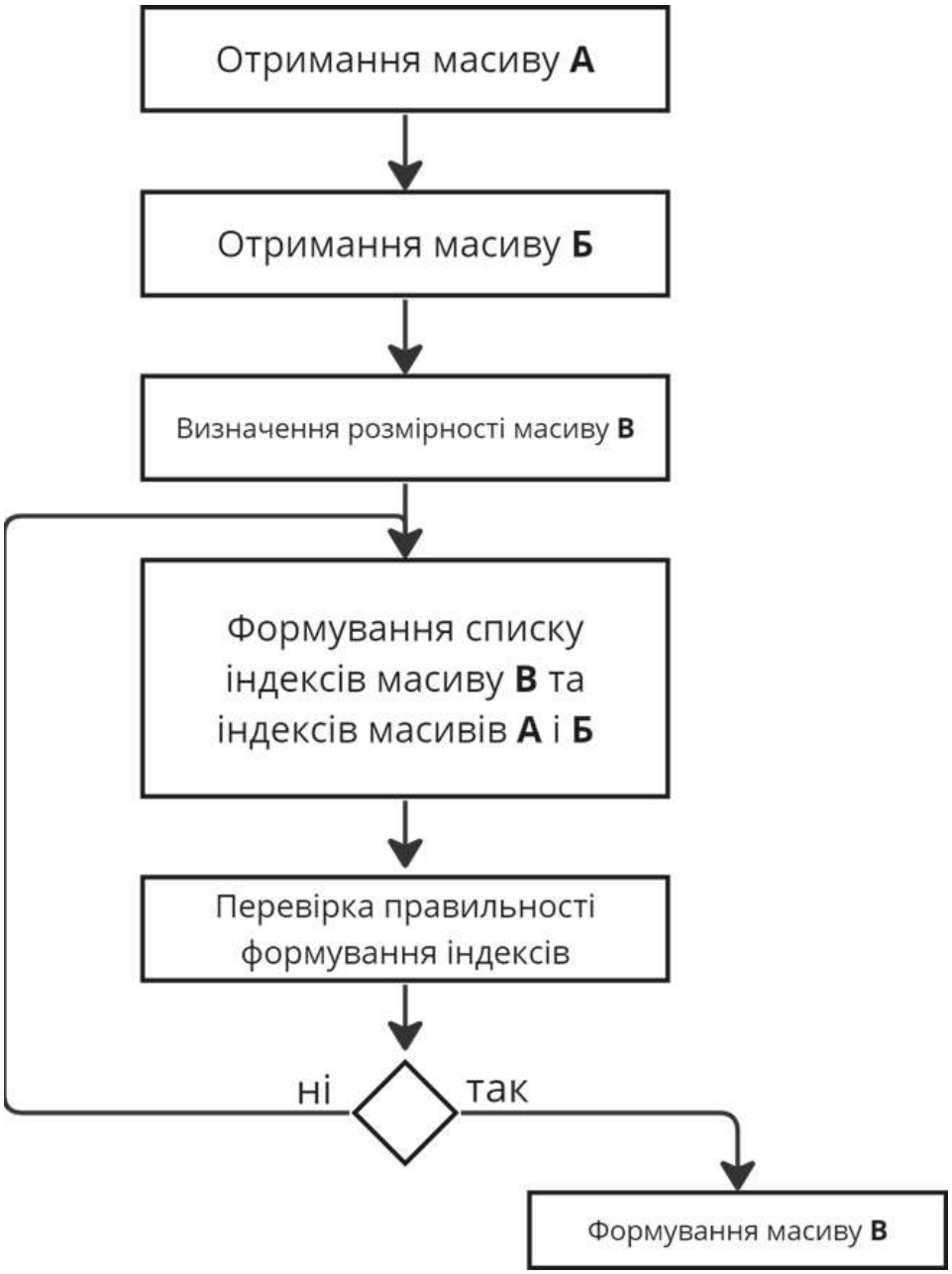


Рисунок 3.1 – Кроки методу

Об'єднання подібності реляційних баз даних в основному передбачають використання високовимірних індексних структур, таких як дерево. Хешування з урахуванням розташування також використовується для об'єднання подібності. Неіндексовані методи передбачають повторний розділ точок даних, поки кожен розділ не стане достатньо малим, ці розділи потім можна об'єднати. Інші подібні запити включають множину подібності об'єднання та просторове об'єднання. Оператор об'єднання масивів з акцентом на ефективне припинення предикатів на основі значень. Оператор використовує інтегроване представлення індексу з

масиву для ефективного обрізання клітин-кандидатів перед їх порівнянням, полегшуючи швидке та точне порівняння між комірками як на розмірних, так і на ціннісних з'єднувальних предикатах. Оператор також ввів і підтримав новий тип операцій об'єднання, об'єднання подібності значень, яке повертає числові пари комірок, значення яких знаходяться в межах а певний діапазон похибок.

3.3 Висновки

Менеджер зберігання масивів, на якому ґрунтується реалізація, включає нову схему зберігання, яка реорганізовує сховище масивів відповідно до розмірних координат та індексів значень на основі бітів, а також зберігає залишкові дані, що описують елементи всередині кожного біта.

Таким чином, удосконалено метод перевірки трансформації кешу комп’ютера.

4 СИСТЕМА ДЛЯ АНАЛІТИКИ ДЛЯ ГЕТЕРОГЕННИХ СИСТЕМ

4.1 Модель обробки масивів

Сучасна модель обробки масивів зіткнулася з викликами, пов'язаними зі зростаючою обсягом даних і необхідністю надавати швидкий зворотний зв'язок користувачам. Традиційно дані зберігаються на диску, і аналітика відбувається лише після збору даних. Однак, цей підхід стає все менш ефективним з врахуванням зростаючої кількості даних та обмежень щодо швидкості обчислень та вводу-виводу. Відтак, для зменшення цих недоліків, аналітика почала переходити до обробки даних на місці, або частково заміняти аналіз перед збереженням даних на диск. Досліджуються нові архітектурні тенденції, щоб адаптувати обробку масиву до вимог обробки в реальному часі.

Хоча аналітика часто використовується для обчислювального управління та моделювання циклів, архітектурні тенденції роблять її більш універсальною. Проте, незважаючи на значний прогрес у цій галузі, аналітика на місці залишається спрощеною і не відповідає вимогам, пов'язаним зі стрімкими архітектурними та прикладними тенденціями.

Друга важлива архітектурна тенденція полягає в тому, що акселератори стають нормою у системах екстремального масштабу, а не винятком. Прискорювачі, зокрема графічні процесори, забезпечують до 97% загальної обчислювальної потужності машини. Проте, обсяг роботи з аналітики, який може бути виконаний за допомогою акселераторів, обмежений. Оскільки більшість обчислень відбувається на прискорювачах з обмеженою пам'яттю, аналітика на місці повинна ефективно використовувати пам'ять, мінімізуючи її використання.

Аналогічно, змінюються потреби додатків щодо аналітики на місці. Загалом, потреби переходять від візуалізації до детальної аналітики. Розглянемо наступну вимогу, зазвичай пов'язану з тим, «коли моделювання запущено, порівняйте вихідні дані з кожного кроку часу зі збереженими даними (або спостережуваними даними, або вихідними даними з іншої моделі) та підсумуйте просторові області

та/або змінні, де є відмінності значний». Інші приклади додатків, які вимагають прогресу в аналітиці на місці, включають ансамблеве моделювання, де всі набори одночасно запущених симуляцій можуть бути припинені, якщо є одне успішне виявлення подій.

Розглянемо спроєктовану гетерогенну систему запитів. Система дозволяє здійснювати складні запити на місці, що генеруються прискорювачами. Основна ідея полягає в тому, щоб побудувати зведене або стисле представлення даних, а потім виконати передачу даних та обробку запитів на цьому поданні замість вихідних даних. У цьому підході представлення можна розглядати як свого роду гомоморфне стиснення. Специфічною репрезентацією є растровий індекс, структура індексації, спочатку розроблена в контексті сховищ даних і згодом широко використовується. Растровий малюнок використовується не як індекс для пошуку даних, а як стислий підсумок даних на основі масиву, який зберігає як просторовий, так і ціннісний розподіл. Таке використання растрових зображень (на відміну від вихідних даних) дозволяє знизити витрати на переміщення даних, зменшити обсяг вводу-виводу, а також ефективність пам'яті та часу при аналізі. Крім того, система надає кілька інших важливих функцій. Вона підтримує ряд основних операторів для даних. Всі реалізовані з використанням растрових зображень і розпаралелені для полегшення розробки складних рішень для обробки запитів і підтримує паралельну обробку запитів як всередині, так і між вузлами. Оцінюванн системи здійснимо за допомогою двох реальних симуляцій. Результати показують, що в порівнянні з базовою лінією, яка обробляє запити з використанням вихідних даних, система значно економить вартість передачі між хостами та прискорювачами навіть після розгляду вартості генерації растрових зображень, забезпечує прискорення як вводу-виводу, так і обчислювальних інтенсивних запитів, і має паралельну масштабованість. Таким чином, вводиться покращений формат растрового зображення, який підходить для генерації та обробки графічних процесорів. Такі растрові формати дозволяють швидко та обмежено обробляти складні запити на місці, такі як із набором контрастів та регіон інтересів, що перекриваються. Він представляє систему, як паралельну та композитну систему

запитів на місці для гетерогенних платформ. Експериментальна оцінка показує, що система може обробляти запити на місці на гетерогенних платформах зі значно меншою вартістю вводу-виводу, а також обчислювальної вартості.

Розглянемо загальні відомості про проєктовану систему. Гетерогенна аналітична система розроблена з кількома цілями. З точки зору продуктивності, цілі полягають у зменшенні часу, що витрачається на передачу даних та введення-виведення, зменшення вимог до пам'яті для аналітики та скорочення часу, витраченого на аналітику. З боку програмованості потрібна підтримка управління вихідними даними моделювання, спрощений логічний вигляд даних, на яких вказується аналітика, і можливість складати запити від ряду операторів.

Розглянемо моделювання та представлення імітаційних даних. Моделювання зазвичай генерує багато часових кроків, кожен з яких містить кілька багатовимірних наборів даних. Отже, система розглядає всі дані як (віртуальні) єдиний багатовимірний масив. Ідентифікатор кроку часу є одним із його вимірів, а кожен набір даних є одним із атрибутів масиву. Коли моделювання виконується, кожен прискорювач зазвичай генерує частину кроку часу, відповідну гіперпрямокутному масиву. Система приймає потік таких сегментів і виконує аналітику на ньому. Кожен сегмент представлений растровим індексом, що розділяє домен значень на ряд неперетинаючих сегментів і використовує растрове зображення для представлення значень, які потрапляють в кожен сегмент. Таке представлення може бути ефективно згенеровано на графічному процесорі і може бути використано для ефективного виконання складних запитів. Це представлення служить кільком цілям: його зменшений розмір полегшує тиск пам'яті та вузьке місце передачі даних на процесор, і це більш ефективно для багатьох запитів, насамперед тому, що це прискорює операції фільтрації на основі значень.

Розглянемо побудову натурного механізму аналітики для гетерогенних платформ. На високому рівні процесор запитів структурований як будь-який з поширених реляційних і масивних баз даних. Кожен план запитів в системі будується за допомогою складених операторів, кожен з яких описує, як вхідні дані повинні бути передані, змінені, агреговані або оброблені крок за кроком. Процесор

запитів виконує оператори, використовуючи відповідні обчислювальні ресурси та ресурси вводу-виводу для генерації результатів. Хоча система дотримується стандартної моделі, потреби запитів та різнорідних платформ вимагають певних змін. По-перше, обрана модель оператора повинна враховувати потоковий характер результатів моделювання. Зокрема, механізм запитів адаптує модель оператора, при цьому кожен оператор надсилає частину масиву вихідних результатів. Це створює більш природний потік управління і ближче відповідає асинхронному характеру обробки запитів на місці. По-друге, оскільки різні оператори можуть бути запущені на різних потоках і пристроях, щоб використовувати паралелізм, що забезпечується різнорідними платформами, передбачені спеціальні оператори для передачі сегментів між потоками і від прискорювача до центрального процесора. Нарешті, система представляє вхідні дані у вигляді растрових індексів для використання обмеженої пам'яті та комунікаційних можливостей прискорювачів. Для цього надається растровий оператор перетворення.

Розглянемо розпаралелювання всередині системи і між вузлами. Однією з проблем обробки запитів на гетерогенних платформах є ефективне використання різних доступних форм паралелізму в системі. Модель виконання допускає паралелізм як всередині, так і між обчислювальними вузлами, через конвеєризацію, а також паралелізм даних. План запитів Система розділений на різні фрагменти, які можуть працювати на різних пристроях або машинах одночасно. Різні фрагменти з'єднуються через пари операторів відправки/прийому, які передають дані від фрагментів, що виробляють дані, до фрагментів, які їх споживають. Передача даних відбувається між прискорювачами та пам'яттю хоста, між різними робочими потоками на хост-процесорі, а також може відбуватися через вузли. Всередині вузла, система породжує один або кілька робочих потоків для запуску фрагментів замість окремих процесів. Це дозволяє більш ефективно обмінюватися між фрагментами через спільний простір пам'яті, а не через дорогі засоби. Крім того, глобальні джерела, такі як буфери пам'яті, спільно використовуються між різними потоками. При необхідності багатоспоживчі, мультивиробникові черги

використовуються для буферизації та передачі проміжних результатів, а також розподілу завдань між різними робочими потоками. На додаток до внутрішньовузлового паралелізму, Система підтримує розпаралелювання між вузлами, коли аналіз на місці повинен об'єднати результати моделювання з декількох вузлів. Асинхронна пара операторів передачі / прийому служить для перемішування результатів між різними вузлами. Між процесами всі дані передаються за допомогою директив. Усі запити спрямовуються через один потік. Щоб надіслати повідомлення, потоки введення серіалізують повідомлення та поміщають його в чергу надсилання. Потім циклічний ланцюжок видає запити на надсилання та контролює стан. Процес на приймальному кінці аналогічний: приймальний потік поміщає отримані буфери в чергу. Потім робочі потоки десеріалізують повідомлення та керують запитом, викликаючи API на батьківському операторі. Щоб максимально уникати непотрібного розподілу пам'яті та дублювання треба це досягати шляхом підтримки загальних буферних пулів і дозволу пропускати буфери між потоками в міру необхідності. Генерація растрових зображень на місці на графічному процесорі є одним з ключових компонентів системи Система має компактне приблизне представлення даних моделювання. Таке подання 1) знижує тиск пам'яті при обробці запитів; 2) прискорює операції передачі даних; 3) прискорює аналітику. Представлення растрового індексу зручне для графічного процесора. Растрові зображення спочатку розроблялися як індекси для дискретних значень в реляційних наборах даних, представляючи діапазон подібних суміжних значень з використанням одного і того ж біту. Пізніше вони були визнані придатними в якості індексу для наборів даних. Зовсім недавно вони стали використовуватися в якості ефективного або приблизного представлення даних, на основі яких запити можуть оброблятися безпосередньо. Коли моделювання виконується на графічних процесорах, і враховуючи, що передача даних між графічним процесором і процесором, ймовірно, буде вузьким місцем в роботі системи, бажано конвертувати моделювання виводу даних в растрові карти на графічних процесорах. Однак використання растрових зображень на графічному процесорі не є простим. Це

пояснюється тим, що високопаралельний характер багатоядерних процесорів суперечить послідовному характеру механізмів кодування довжини виконання, що використовуються популярними растровими схемами стиснення. Хоча існують роботи зі створення растрових зображень на графічних процесорах, вони в основному зосереджені на генерації растрових представлень кодування довжини виконання на графічному процесорі. Таким чином, вони генерують послідовне представлення на прискорювачі, призначеному для виконання масивних паралельних обчислень. Як наслідок, вони вимагають лінійних тимчасових просторів для виконання і не є ефективними. Замість цього система використовує сегментоване (фрагментоване) растрове представлення, подібне до ревучих растрових зображень. У такому поданні растрове зображення ділиться на кількість зрізів однакової довжини, і кожен фрагмент стискається відповідно до його кардинальності, яка є кількістю 1 біт у фрагменті. Ця структура має ту перевагу, що забезпечує ефективний і паралельний випадковий доступ, а також зберігає кожен окремий фрагмент таким чином, щоб забезпечити ефективний розмір. Тому що в растровому представленні згенерованого є кілька сегментів набору даних. Те, як зберігається зріз фізично, залежить від його кардинальності та кількості. Поточна реалізація системи зберігає один фрагмент в одному з трьох форматів. Фрагмент завжди зберігається у форматі, який займає найменше місця. Як випливає з назви, бітовий фрагмент зберігає значення як суміжний бітсет, використовуючи 216 біт (4096 байт) слів, причому 1 біт представляє значення в позиції, а 0 біт в іншому випадку. Фрагмент масиву зберігає зміщення значень у ньому за допомогою 16-бітових цілих чисел, використовуючи 2 байти для фрагмента з кардинальністю. Нарешті, повний фрагмент просто представляє фрагмент, який містить усі, він не займає жодних додаткових пробілів, крім метаданих. Звичайно, певні фрагменти можна зберігати більш ефективно, якщо ввести більше представлень. Наприклад, кодування довжини виконання може бути використано для стиснення фрагмента з довгими, суміжними діапазонами і для фрагментів, що містять усі, крім декількох значень у наборі даних, представлення фрагмента як його заперечення може зберегти деякі пробіли. Однак більш складні схеми можуть

ускладнити генерацію фрагментів, а також обробку запитів. Однією з ключових відмінностей представництва є його компонування сховища. Такий макет відіграє важливу роль у забезпеченні зручності растрового зображення для графічного процесора. Це пояснюється тим, що динамічний розподіл на графічних процесорах, хоча і можливий, дорогий і викликає фрагментацію, що перешкоджає ефективному переміщенню даних на хост. Система завжди зберігає всі фрагменти в одному безперервному сховищі. Бітове представлення зберігає зміщення сховища та тип кожного фрагмента як метадані. Оскільки розмір фрагмента можна зробити зі зміщень сусідніх фрагментів, це дозволяє ефективно отримати доступ до будь-якого фрагмента. Кожен вид метаданих зберігається в окремих масивах для кращого розпаралелювання та локальності даних. Просторово-ефективна растрова генерація потрібна, коли нові дані з'являються в результаті моделювання, то оператор перетворює окремі вхідні фрагменти в растрові формати, які використовує система. Розглянемо метод генерації растрових зображень, що використовується системою, який вимагає лише додаткового простору із загальною кількістю фрагментів у растровому зображенні. Генерація растрових карт ведеться в два проходи для подолання неефективного розподілу пам'яті на графічних процесорах. Перший прохід обчислює необхідні метадані, а також загальні вимоги до простору. Після цього другий прохід виробляє фактичні фрагменти, які потрібно зберігати. Перший прохід обчислює розміри та типи кожного фрагмента. Оскільки те, як зберігається частинка даних, залежить від його кардинальності, перший прохід по суті повинен обчислити кардинальність кожного фрагмента. Оскільки кожне значення належить точно до одного сегмента бітових індексів, це, по суті, побудова гістограми на ідентифікаторах сегментів значень. Коли кількість сегментів невелика, гістограма може бути створена шляхом обчислення часткових гістограм у спільній пам'яті на потік графічного процесора та агрегації результатів. В іншому випадку для агрегування часткових рахунків потрібні атомні інструкції. Після генерації гістограми обчислюється тип і розмір кожного фрагмента на основі обчисленої кардинальності. Потім визначаються загальні потреби в сховищі кожного і виділяється пам'ять. Потім другий прохід на

вхід виробляє фактичне фізичне представлення зрізу. При цьому растрове представлення кожного фрагмента генерується відповідним блоком потоку, тобто, якщо растровий індекс має сегменти і набір даних сканується багато разів, один раз для кожного сегмента. Це може здатися неефективним, оскільки теоретично одне сканування може генерувати всі блоки одночасно. Однак виявляємо, що збережена робота не перетворюється на зекономлений час генерації. Пояснити це можуть два фактори. По-перше, ефективні механізми кешування в графічних процесорах зменшують додаткові витрати на читання даних багаторазово, а по-друге, виділення декількох фрагментів в одному блоці збільшує складність коду, додає більше тиску в регістрах і знижує ефективність механізму розкладу блоків потокових процесорів. Кожне викривлення графічного процесора генерує одне 32-бітне слово бітсетів одночасно. Це досягається кожним потоком графічного процесора генерації одного біта, а потім використання інструкцій синхронізації для деформації (голосування в CUDA) для створення бітового слова. Хоча також можна уникнути зв'язку з деформацією, дозволяючи кожному потоку графічного процесора обробляти 32 вхідні значення та генерувати одне бітсетне слово самостійно.

Фрагмент масиву зберігає значення у фрагменті як список 16-бітових зсувів. Фрагменти масиву можуть бути сформовані за допомогою модифікованого алгоритму паралельного сканування. У цьому підході спочатку генерується логічний масив, який вказує, чи знаходиться значення в фрагменті всередині сегмента. Потім на цьому згенерованому масиві виконується ексклюзивна сума префіксів; в результаті утворюється список індексу кожного значення в сегменті. Нарешті, якщо початкове значення знаходиться всередині множини, його зміщення записується в індекс масиву, створюючи сегмент масиву. Оскільки логічний масив та індексний список не потребують фактичної матеріалізації, цей процес також не потребує додаткового тимчасового простору. Нарешті, повні сегменти і порожні масиви не займають місця зберігання, і їх можна пропустити в другому проході. Глобальний простір пам'яті для генерації растрових зображень призводить до дуже ефективного алгоритму генерації растрових зображень. Виконання складної

аналітики за допомогою растрових зображень є доцільним. Система дозволяє розробляти комплексну аналітику на місці. Однією з проблем для системи є переміщення згенерованих растрових зображень на хости для аналітики. Це робиться оператором. Коли оператор отримує згенерований растровий сегмент, він розподіляє буфер хоста відповідно до його розміру та ініціалізує асинхронну передачу для копіювання фрагмента в пам'ять хоста. Оскільки багато передач вимагають закріплених буферів, розподіл пам'яті може значно знизити продуктивність. Система використовує безпечні для потоків буферні масиви пам'яті для повторного використання виділених буферів, якщо це можливо.

4.2 Аналітичні механізми на растрових зображеннях

Розглянемо реалізацію основних операцій на растрових зображеннях в системі, які є ключовими компонентами складних аналітичних запитів. Одна з таких операцій - оператор фільтра, який виконує виділення предикатів на основі атрибутів. У випадку растрового індексу така операція може бути розкладена на логічні побітові операції. Застосування фільтра за умовами до масиву атрибутів включає виконання побітових операцій на вибраних сегментах в кожному атрибуті, а потім побітово об'єднує оператори на всіх растрових зображеннях, що були згенеровані в результаті операцій. Для покращення реалізації, система використовує ефективні методи роботи з пам'яттю для обробки растрових зображень. Наприклад, не потрібно матеріалізувати всі проміжні згенеровані растрові зображення, замість цього растрові індекси обробляються сегмент за сегментом. Оскільки не можна гарантувати однакову кардинальність двох операцій або операцій між стовпцями, бітовий зріз використовується для зберігання проміжного фрагмента незалежно від типу вхідного фрагмента. Це також дозволяє прискорити обробку по бітах. Для певних операцій, які виконуються над підмножиною даних, застосовується виділення на предикатах на основі розмірів. Наприклад, користувач може запитувати лише область підмножини інтересів на певний проміжок часу. Для таких виділень генерується маска бітового зрізу

залежно від вибраних предикатів, а потім виконується побітова операція між маскою та растровими індексами для отримання кінцевого результату. Основна ідея полягає в збереженні загальної кількості агрегацій за атрибутом, а потім використанні підрахунку генеральної сукупності відфільтрованого результату для оцінки кінцевого результату на основі збережених агрегацій.

Додатковою перевагою використання побітових операцій та ефективних методів роботи з пам'яттю є зниження обсягу обробки даних. Замість того, щоб обробляти всі записи або проміжні результати, система може працювати лише з вибраними бітами, які відповідають умовам запиту. Це сприяє підвищенню продуктивності і зменшенню витрат обчислювальних ресурсів.

Окрім того, враховуючи поширення акселераторів, особливо графічних процесорів, в системах екстремального масштабу, стає необхідним розробити архітектурні рішення, що використовують ці прискорювачі. Підхід, який використовує аналітику на місці, повинен бути сумісним із специфічними обмеженнями прискорювачів, такими як обмежена пам'ять. Використання мінімальної кількості пам'яті для аналітики на місці є важливим фактором для досягнення оптимальної продуктивності та забезпечення швидкого зворотного зв'язку для користувачів. Розробка ефективних реалізацій операцій на растрових зображеннях є важливим завданням для забезпечення швидкої та потужної аналітики. Використання побітових операцій, оптимізованих методів роботи з пам'яттю та інтеграція з прискорювачами дозволяють досягти ефективної обробки даних та задоволення потреб користувачів у реальному часі.

Таким чином, завдання системи полягає в розв'язанні проблеми підрахунку бітів установки в растровому представленні, що є досить простим завданням. Основна мета системи - знайти пари комірок, які задовольняють певним умовам об'єднання. Для реалізації предикатів рівного об'єднання на вимірах та атрибутах використовуються прості операції між виділеними атрибутами та їх розмірами. З введенням таких примітивів можна реалізувати багато складних запитів.

Перший запит, який розглядається, спрямований на визначення подібності між поточним запуском моделювання та попереднім базовим планом. Користувач

може зацікавитися, наскільки кожна симуляція відрізняється від попередніх, з метою визначення необхідності коригування або зміни методології. Така подібність може бути обчислена шляхом порівняння кількості схожих атрибутів між відповідними часовими кроками, тобто для кожного атрибута ми рахуємо кількість комірок з однаковими розмірними положеннями часових кроків, що мають значення атрибутів, відмінні менше певного порогу. Потім оцінки для кожного атрибута комбінуються для обчислення загальної подібності.

Продовжуючи дослідження, було виявлено, що реалізація таких запитів з використанням растрового представлення дозволяє ефективно виконувати аналіз подібності на великій кількості даних. Растрове представлення дозволяє зберігати та обробляти інформацію у вигляді бітових значень, що сприяє швидкості операцій та зменшенню обсягу пам'яті, необхідної для зберігання даних. Крім того, використання растрового представлення у поєднанні з оптимізованими алгоритмами обробки даних дозволяє значно прискорити час виконання запитів і підвищити продуктивність системи в цілому.

Ці результати свідчать про переваги використання растрового представлення для проведення аналітичних запитів та підкреслюють його значення у сфері обробки великих обсягів даних. Розроблена система та система запитів, яка спеціально призначена для складного та ефективного аналізу на сучасних хмарних платформах, забезпечують зручне растрове представлення даних та підтримують виконання різноманітних складних запитів на місці. Оцінка реальних симуляцій підтверджує, що система в середньому перевершує базовий план на 20 відсотків, одночасно заощаджуючи пам'ять та пропускну здатність вводу-виводу.

Це можна розглядати як просту агрегацію поверх операції об'єднання між різними масивами. За допомогою растрових зображень операція об'єднання може підтримуватися побітовим виконанням і між відповідними бітами. Растрове малюнок прискорює запит двома способами: економить витрати на введення-виведення при читанні даних з диска і передачі з графічного процесора на процесор, а також дозволяє швидше проводити порівняння між елементами.

Крім того, растрове представлення може бути використане для оцінки різних інших показників подібності між часовими кроками. Наступний запит спрямований на вивчення відмінностей між запущеним моделюванням і попереднім базовим планом. Пара контрастних множин є двома підмножинами моделювання та базової лінії, обраними за допомогою одних і тих самих предикатів фільтрації. Важливим етапом є вибір тих предикатів фільтрації, які створюють найбільш значну різницю у значеннях цільових атрибутів. Це може вказувати на виявлення різниці між двома експериментальними результатами, що розглядаються. Тому цей запит спрямований на пошук предикатів, які створюють найрізноманітніші контрастні множини, зазвичай визначені за допомогою функції якості.

Важливо зазначити, що використання растрових зображень та растрового представлення даних в загальному контексті дозволяє ефективно аналізувати та порівнювати великі обсяги інформації. Це особливо корисно в задачах, де потрібно швидко обробляти та визначати подібність між складними даними. Застосування растрових зображень у поєднанні з відповідними алгоритмами аналізу дозволяє ефективно вирішувати завдання порівняння, агрегації та виявлення відмінностей у даних. Такий підхід відкриває нові можливості для швидкого та точного аналізу даних у різних сферах, включаючи науку, медицину, фінанси та багато інших.

Функція якості, у свою чергу, використовує такі величини, як розмір (підтримка) та середнє значення двох підмножин. Для пошуку таких контрастних множин діапазон пошуку дискретизується на ряд бітів. Для вирішення задачі може бути використана стратегія пошуку, заснована на дереві перерахування. Пошук починається з підмножини, що містить всі елементи в наборах даних, і додавання однієї умови фільтра за раз. Для кожної підмножини операція фільтрування та агрегації обчислює середнє значення та розмір підмножин, а потім обчислюється якість контрастних наборів. Така операція фільтра і агрегації може бути виконана ефективно на растрових представленнях. Знову ж таки, растрове представлення прискорює введення-виведення за рахунок зменшення переміщуваних даних, а також призводить до швидшої фільтрації та агрегації в менших сценаріях

селективності. Будучи природно дискретизованим повторенням даних, використання растрових індексів для обробки цього запиту також має додаткові переваги надання результатів, настільки точних, як вихідне представлення, за умови, що кількість бітів становить мінімальну можливу. Безперервні області в результатах моделювання, що задовольняють певному порогу, часто шукаються для подальшого вивчення. Враховуючи запущене моделювання та умову фільтра, що визначає, які клітинки є «цікавими», запит шукає суміжні кроки часу з великими безперервними перекритими областями, що свідчить про те, що ці кроки часу мають стабільні функції. Запит може бути реалізований шляхом фільтрації двох суміжних часових кроків для клітинок, потім виконується операція приєднання для пошуку перекритих комірок інтересів. Потім ці клітини об'єднуються, щоб знайти найбільшу суміжну область. Тут растрові індекси прискорюють запит, скорочуючи час передачі з графічного процесора на процесор, а також полегшуючи швидшу роботу фільтра. Оцінимо продуктивність запитів системи на кластері, використовуючи реальні запити, застосовані до фактичного результату моделювання. Зокрема, завдяки новим варіантам обладнання, доступним у хмарних налаштуваннях, можна виконувати інтенсивні робочі навантаження в хмарі з продуктивністю та масштабованістю аналогічно тому, що отримують на локальних об'єктах. Всі експерименти виконуються на великих екземплярах, з 8 процесорами Intel і графічним процесором NVIDIA. Екземпляри мають пропускну здатність мережі до 10 гігабайт. Тестування показує, що пропускна здатність порівнянна з локальними кластерами з підключеннями PCI-e. Коли дані зберігаються в паралельній файловій системі з базовою пропускною здатністю. Система працює під управлінням операційної системи. У експериментах використовуються два реальних орієнтира. Моделювання обчислює взаємодію за допомогою методів, заснованих на сітці, фіксуючи координати ї, а також їх силу, швидкість та прискорення. Системний кеш очищається перед усіма експериментами, пов'язаними з введенням-виведенням. Порядок експериментів рандомізований для амортизації базових коливань продуктивності системи. Час обробки виключає час ініціалізації моделювання та час отримання результату, щоб

підкреслити продуктивність. Основна ідея системи полягає в тому, щоб замінити вихідні дані моделювання на графічних процесорах їх растровим представленням. Хоча це зменшує розмір даних, що передаються, створення растрових індексів також передбачає більше обчислень. Перший набір експериментів досліджує, як додаткові витрати на генерацію растрових зображень порівнюються з економією за рахунок передачі меншої кількості даних. Час моделювання також відображається як орієнтир. Всі симуляції виконуються протягом 10000 кроків, і потім змінюємо обсяг даних, що генеруються, контролюючи розмір проблеми. Вибираємо п'ять розмірів задачі для кожного моделювання, так що різниця обсягу даних, що генеруються за крок часу між сусідніми налаштуваннями, становить приблизно два рази. Крім того, перенесення растрових зображень лише додає відносно невеликі додаткові витрати порівняно з моделюванням. Дані профілювання показують, що коли розмір проблеми великий, то вартість створення та копіювання растрового представлення становить близько 29% копіювання вихідних даних, при цьому більшість часу витрачається на генерацію растрових зображень. Інше спостереження полягає в тому, що, незважаючи на те, що кількість даних, що генеруються в секунду, залишається відносно незмінною зі збільшенням розміру кроку часу, відносна продуктивність растрового представлення покращується. По-перше, зі збільшенням розміру проблеми потрібно видавати менше передач для тієї ж кількості даних, зменшуючи додаткові витрати на ініціалізацію передач; по-друге, оскільки кожен блок потоку обробляє фрагмент індексів растрового зображення, збільшення розміру кроку часу збільшує кількість потоків, які можуть бути заплановані, що дає планувальнику графічного процесора кращі можливості для покращення використання пристроїв. Хоча передача вихідних даних відбувається дещо швидше, коли розмір задачі менший, растрове представлення поступово перевершує вихідне представлення зі збільшенням розміру задачі. Коли розмір проблеми великий, то система має краще прискорення порівняно з оригінальним варіантом. Це відповідає попереднім спостереженням, які полягали в тому, що генерація растрових індексів є більш ефективною, коли графічний процесор має більше даних для роботи. Дані профілювання підтверджують це, тоді

як передача растрового представлення характерна передачі вихідних даних за крок часу. В цілому, переваги збереженого перенесення значно переважають додаткові витрати на генерацію растрових зображень при обох моделюваннях.

Розглянемо, наскільки ефективно система може обробляти різні робочі навантаження запитів, зокрема запити, які використовують растрове представлення даних. В рамках набору експериментів буде порівняна продуктивність таких запитів з базовою лінією, яка використовує попередній прогін з іншим параметром балансу.

Для порівняння система виконує запити, які генеруються безпосередньо за допомогою прикладів запитів, моделює подібність та обчислює кількість клітинок, що достатньо схожі з попереднім тестом. Результати цих запитів використовуються для оцінки продуктивності.

Загалом, виконання запитів на растровому представленні даних значно прискорює процес в порівнянні з іншими методами моделювання. Додаткові витрати на обробку запитів є незначними порівняно з запуском самої симуляції. Іншими словами, обробка запиту щодо найбільшого розміру проблеми призводить до додаткових витрат, але ці витрати є малозначними порівняно з часом моделювання.

Однією з переваг растрового представлення є те, що порівняння між результатами моделювання може бути виконане швидко за допомогою простих побітових операцій, що призводить до невеликих додаткових витрат. Крім того, завдяки компактності растрового представлення, витрати на ввід/вивід також дуже обмежені.

З іншого боку, запит вихідних даних може вимагати читання значно більшого обсягу даних з віддаленої файлової системи, а також переміщення такої ж кількості даних з пристрою в пам'ять хоста. Це може призводити до уповільнення процесу.

Хоча запит подібності моделювання не надмірно навантажує процесори, існують інші, більш інтенсивні запити, які можуть вимагати значних ресурсів процесора. Один з таких запитів, який зараз розглядається, - це інтелектуальний

аналіз з набором контрастів. Цей запит визначає, яка умова фільтрації створює пару підмножин кроку часу, який має найбільший "контраст" між ними.

Для оцінки продуктивності системи проводять порівняльний аналіз, враховуючи її продуктивність при виконанні запиту на вихідних даних для обох моделей. У цьому випадку було встановлено 200 ітерацій. При пошуку предикатів підмножин, всі розміри і значення дискретизуються з однаковою шириною в 64 біти як для растрового зображення, так і для вихідного представлення даних.

Загальний час обробки запиту на обох наборах даних, в цілому, виявився помітно повільнішим у випадку запитів з контрастним набором на простих наборах даних. Щоб проілюструвати характеристики виконання запиту з контрастним набором, було виділено чотири складові часу між обробкою растрових індексів і вихідними даними: вартість генерації растрових даних, час передачі даних до основної системи, сумарний час обробки всіх робочих потоків запиту та вартість вводу-виводу при зчитуванні даних з файлової системи.

Розбивка цих часових складових показує, що в більшості сценаріїв растрове представлення перевершує вихідне представлення у всіх трьох аспектах запиту: генерації та передачі даних до основної системи, зчитуванні даних з диска та пошуку. Це свідчить про те, що растрове представлення даних має переваги щодо продуктивності як у випадку операцій вводу-виводу, так і випадку ресурсів процесора, що демонструє, що згенеровані дані мають меншу спотвореність порівняно з модельними даними. Такі результати підкреслюють значення растрового представлення для виконання ефективних та швидких аналітичних запитів на великі обсяги даних.

Загалом растрове представлення може не тільки прискорити продуктивність вводу-виводу системи запитів на місці, але також може прискорити запити з інтенсивним процесором. Розглянемо запит області інтересів, що перекривається, який шукає протягом різних кроків часу. Оскільки набір даних є неструктурованим моделюванням і не має розмірної структури, придатної для такого запиту, цей експеримент зосереджений на моделюванні. Хоча растрове представлення не має такого великого прискорення порівняно з попередніми

запитами, які включали введення-виведення і це пов'язано з тим, що хост-процесори здатні працювати з вхідними даними, тому основним вузьким місцем все ще залишається рух даних від графічного процесора до центрального процесора, який можна зберегтись, використовуючи растрове представлення. Розглянемо продуктивність системи за запитами з міжвузловим зв'язком. Оскільки кожен вузол імітує частину сітки, завданням видобутку може бути обчислення контрастних наборів між вузлами. Цей запит застосовується до обох симуляцій, масштабуючи їх так, щоб кожен вузол генерував фіксовану кількість даних. Представлений середній час обробки контрастного набору запиту, в якому зі збільшенням кількості вузлів час обробки запитів як растрового зображення, так і вихідного представлення залишається стабільним лише з незначним збільшенням часу. Однак у 16 вузлах час обробки запиту простого представлення значно збільшується, тоді як час запиту растрового представлення залишається стабільним. Це можна пояснити збільшенням вартості вводу-виводу. Оскільки паралельна файлова система, яку використовуємо, має фіксовану загальну пропускну здатність, то зі збільшенням загального обсягу даних, що читаються з файлової системи, вузьке місце кожного вузла перетворюється на ввід/вивід від обчислень. Оскільки растрове представлення вимагає менше вводу-виводу, воно може масштабуватися далі.

Хоча ідея спільної обробки має десятиліття історії, але методи були зосереджені на візуалізації і функціональності аналізу на місці. Протягом цього часу також проводилася робота по обробці вихідних даних, які знаходяться на дисках, без їх збереження в базі даних. Початково растрові індекси були запропоновані для реляційних баз даних, але в системі вони були реалізовані для покращеної швидкодії обробки запитів.

Щодо стиснення даних, в історичному контексті використовувалися варіантні схеми кодування довжини виконання. Однак, останнім часом растрові зображення стали дуже популярними завдяки їх високій ефективності обробки запитів. Виникають спроби підвищити точність таких зображень або ще більше

зменшити їх розмір. Запропоновані растрові зображення з деревним кодуванням для економії простору.

Хоча растрові зображення можуть бути використані як вторинний індекс, вони також можуть використовуватися як первинний індекс або наближене представлення для швидшої обробки запитів. Однак, для цього потрібно або сортування даних, або попереднє побудова нестисненого растрового зображення, що призводить до додаткових витрат часу та простору. Така побудова не є бажаною, особливо коли задіяні прискорювачі, такі як графічні процесори.

Таким чином, було розроблено систему та систему запитів, призначену для складної та ефективної аналітики на сучасних хмарних платформах. Її представлення зручне для растрового представлення, оскільки таке представлення може бути ефективно згенеровано на графічних процесорах. Це відкриває широкі можливості для обробки різноманітних складних запитів на місці без значного втрати продуктивності.

Оцінка реальних симуляцій підтверджує, що розроблена система перевершує базову лінію в середньому на 20 відсотків. Це означає, що застосування даної системи дозволяє досягти значних покращень у продуктивності обробки запитів. Не лише система є швидкішою, але вона також ефективно використовує ресурси, що призводить до значних заощаджень пам'яті і пропускної здатності вводу-виводу.

Ці результати демонструють значний потенціал системи для використання у різних областях, де потрібна швидка та ефективна аналітика. Будь-то це наукові дослідження, фінансовий аналіз, медична діагностика чи інше, система запитів на базі растрового представлення може стати незамінною у вирішенні складних завдань та забезпечити високу якість обробки даних.

Потенціал цієї системи може бути ще більшим, якщо продовжувати вдосконалювати алгоритми та методи оптимізації обробки запитів. Подальші дослідження і розвиток системи можуть призвести до ще більш значних покращень продуктивності та розширення її можливостей. Застосування системи запитів на базі растрового представлення в майбутньому може привести до

перетворення підходу до аналітики та обробки даних, дозволяючи швидше та ефективніше робити висновки з великих обсягів інформації.

4.3 Висновки

Таким чином, було розроблено систему для перевірки трансформації та моделювання кешу комп’ютера. З нею було реалізовано експериментальні дослідження.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень удосконалено методи для перевірки трансформації та моделювання кешу комп’ютера.

У першому розділі проаналізовано відомі методи для перевірки трансформації та моделювання кешу комп’ютера, виділено недоліки та зроблено постановку задачі дослідження для вирішення проблем.

У другому розділі удосконалено метод моделювання кешу комп’ютера за рахунок обробки растрових індексів графічним процесором.

У третьому розділі удосконалено метод перевірки трансформації кешу комп’ютера за рахунок об’єднання індексованих масивів.

У четвертому розділі реалізовано експериментальні дослідження розроблених рішень.

Практична значимість отриманих результатів полягає у розробеній перевірки трансформації та моделювання кешу комп’ютера.

За темою дипломної роботи опублікована одна публікація у Збірнику наукових праць за матеріалами XIV Всеукраїнської науково-практичної

конференції «Актуальні проблеми комп’ютерних наук АПКН-2022». Хмельницький – 2022 [93].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Dan A Alcantara, Vasily Volkov, Shubhabrata Sengupta, Michael Mitzenmacher, John D Owens, and Nina Amenta. Building an efficient hash table on the gpu. In GPU Computing Gems Jade Edition. *Elsevier*. 2012. Pp. 39–53.

2. Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 2017. Pp. 469–482.

3. Witold Andrzejewski and Robert Wrembel. GPU-WAH: Applying GPUs to compressing bitmap indexes with word aligned hybrid. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, Berlin, Heidelberg, 2010. Vol. 6262 LNCS. Pp. 315–329.

4. Vo Ngoc Anh and Alistair Moffat. Index compression using 64-bit words. 2010. Software Practice and Experience.

5. Panagiotis Antonopoulos, Alex Budovski, Cristian Diaconu, Alejandro Hernandez Saenz, Jack Hu, Hanuma Kodavalla, Donald Kossmann, Sandeep Lingam, Umar Farooq Minhas, Naveen Prakash, Vijendra Purohit, Hugh Qu, Chai- tanya Sreenivas Ravella, Krystyna Reisteter, Sheetal Shrotri, Dixin Tang, and Vikram Wakade. Socrates: The new SQL server in the cloud. *ACM SIGMOD International Conference on Management of Data*. 2019. Pp. 1743–1756.

Gennady Antoshenkov. Byte-aligned bitmap compression. In DCC’95, 1995.

6. Lars Arge, Octavian Procopiuc, Sridhar Ramaswamy, Torsten Suel, and Jef- frey Scott Vitter. Scalable sweeping-based spatial join. *VLDB*. Citeseer, 1998. Vol. 98. Pp. 570–581.

7. Manos Athanassoulis et al. Upbit: Scalable in-memory updatable bitmap indexing. *SIGMOD ’16*. 2016.

8. Utkarsh Ayachit, Andrew Bauer, Earl PN Duque, Greg Eisenhauer, Nicola Ferrier, Junmin Gu, Kenneth E Jansen, Burlen Loring, Zarija Lukic, Suresh Menon, et al. Performance analysis, design considerations, and applications of extreme-scale in situ

infrastructures. *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. *IEEE*. 2016. Pp.77. Pp. 921–932.

9. Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. ParaView catalyst: Enabling in situ data analysis and visualization. *In Proceedings of ISAV 2015: 1st International Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, Held in conjunction with SC 2015: The International Conference for High Performance Computing, Networking, Storage and Analysis*. 2015. Pp. 25– 29.

10. Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic Sample Selection for Approximate Query Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2003. Pp. 539–550.

11. Alex R Van Ballegooij. RAM: A Multidimensional Array DBMS. *EDBT'04 Proceedings of the 2004 international conference on Current Trends in Database Technology*. 2004. Pp. 154–165.

12. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. More- land, P. O'Leary, V. Vishwanath, B. Whitlock, and E. W. Bethel. In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms. *In Computer Graphics Forum*. 2016. Wiley Online Library. Vol. 35. Pp. 577–597.

13. Stephen D Bay and Michael J Pazzani. Detecting group differences: Mining contrast sets. *Data mining and knowledge discovery*. 2001. 5(3):213–246.

14. Tekin Bicer, Jian Yin, and Gagan Agrawal. Improving I/O Throughput of Scientific Applications using Transparent Parallel Compression. In *CCGrid'14*. 2014.

15. Tekin Bicer, Jian Yin, David Chiu, et al. Integrating online compression to accelerate large-scale data analytics applications. In *IPDPS '13*. 2013.

16. Robert Bird, Patrick Killian, and Brian Albright. VPIC on GPU. *Bulletin of the American Physical Society*. Pp. 64. 2019.

17. Truls A Bjørklund et al. Inverted indexes vs. bitmap indexes in decision support systems. In *CIKM '09*. 2009.

18. Sergey Blagodurov, Alexandra Fedorova, Sergey Zhuravlev, and Ali Kamali. A case for numa-aware contention management on multicore systems. In *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE. 2010. Pp. 557–558.

19. Spyros Blanas, Kesheng Wu, Surendra Byna, Bin Dong, and Arie Shoshani. Parallel data analysis directly on scientific file formats. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM. 2014. Pp. 385– 396.

20. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*. 1970. 13(7):422–426.

21. Christian B¨ohm, Bernhard Braunmu¨ller, Florian Krebs, and Hans-Peter Kriegel. Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data. In *ACM SIGMOD Record*. ACM. 2001. Vol. 30. Pp. 379–388.

22. Norm Buchanan, Steven Calvez, Pengfei Ding, Derek Doyle, Christopher Green, Alexander Himmel, Burt Holzman, James Kowalkowski, Andrew Norman, Marc Paterno, et al. Enabling neutrino and antineutrino appearance observation measurements with hpc facilities.

23. Gregory Buehrer and Kumar Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the 2008 inter- national conference on web search and data mining*. 2008. Pp. 95–106.

24. Martin Burtscher and Paruj Ratanaworabhan. FPC: A high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers*. 2009. 58(1).

25. Stefan Bu¨ttcher, Charles L A Clarke, and Gordon V Cormack. *Information retrieval: Implementing and evaluating search engines*. Mit Press. 2016.

26. Surendra Byna, Andrew Uselton, David Knaak, and Yun Helen He. Lessons Learned from a Hero I / O Run on Hopper. *Cray User Group conference*. 2013.

27. John C. Shafer and Rakesh Agrawal. Parallel Algorithms for High-dimensional Proximity Joins. 1999.

28. Guadalupe Canahuate et al. Update conscious bitmap indices. In *SSDBM*. 2007.

29. Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate query processing using wavelets. *Proceedings of the 26th Interna- tional Conference on Very Large Data Bases*. *VLDB'00*. 2000. 10(2-3):111–122.

30. Samy Chambi, Daniel Lemire, et al. Better bitmap performance with Roaring bitmaps. *Software: practice and experience*. 2016. 46(5).

31. Samy Chambi, Daniel Lemire, Owen Kaser, and Robert Godin. Better bitmap performance with Roaring bitmaps. *Software - Practice and Experience*. 2016. 46(5):709–719.

32. Chee Yong Chan and Yannis E. Ioannidis. Bitmap index design and evaluation. In *SIGMOD Record*. ACM. 1998. Vol. 27. Pp. 355–366.

33. Chee Yong Chan and Yannis E. Ioannidis. An Efficient Bitmap Encoding Scheme for Selection Queries. In *SIGMOD Record (ACM Special Interest Group on Management of Data)*. 1999. Vol. 28. Pp. 215–226.

34. Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Lecture Notes in Computer Science (including sub- series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformat- ics)*. Springer. 2002. Vol. 2380 LNCS. Pp. 693–703.

35. Surajit Chaudhuri, Gautam Das, Mayur Datar, Rajeev Motwani, and V. Narasayya. Overcoming limitations of sampling for aggregation queries. In *Proceedings - International Conference on Data Engineering*. IEEE. 2001. Pp. 534–542.

36. Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*. 1997. 26(1):65–74.

37. Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. Approximate Query Processing: No silver bullet. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2017. Vol. Part F127746. Pp. 511–519.

Jerry Chen, Ilmi Yoon, and E. Wes Bethel. Interactive, Internet delivery of visualization via structured prerendered multiresolution imagery. *IEEE Trans- actions on Visualization and Computer Graphics*. 2008. 14(2):302–312.

38. Shimin Chen and Qin Jin. Persistent b+-trees in non-volatile main memory. *Proceedings of the VLDB Endowment*. 2015. 8(7):786–797.

39. Yu Chen and Ke Yi. Two-level sampling for Join size estimation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2017. Vol. Part F127746. Pp. 759–774.

40. Hank Childs. Architectural challenges and solutions for petascale postprocessing. In *Journal of Physics: Conference Series*. IOP Publishing. 2007. Vol. 78. Pp. 12012.

41. Hank Childs, Kwan-Liu Ma, Hongfeng Yu, Brad Whitlock, Jeremy Mered- ith, Jean Favre, Scott Klasky, Norbert Podhorszki, Karsten Schwan, Matthew Wolf, et al. In situ processing. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States). 2012.

42. Jerry Chou, Mark Howison, Brian Austin, Kesheng Wu, Ji Qiang, E Wes Bethel, Arie Shoshani, Oliver Ru¨bel, and Rob D Ryne. Parallel index and query for large scale data analysis. In *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*. 2011. Pp. 1–11.

43. Jerry Chou, Mark Howison, et al. Parallel index and query for large scale data analysis. In *SC '11*. 2011.

44. Alessandro Colantonio and Roberto Di Pietro. Concise: Compressed 'n' Composable Integer Set. *Information Processing Letters*. 2010. 110(16).

Yann Collet. Zstandard - real-time data compression algorithm, 2016.

45. Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*. 2011. 4(1-3):1–294.

46. Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*. 2005. 55(1):58–75.

47. Jeffrey Dean. Challenges in building large-scale information retrieval systems. In *WSDM '09*. 2009.

48. Fran¸cois Deli`ege and Torben Bach Pedersen. Position list word aligned hybrid. In *EDBT '10*. 2010.

49. Sheng Di and Franck Cappello. Fast error-bounded lossy hpc data compression with sz. In *2016 IEEE international parallel and distributed processing symposium (ipdps)*. 2016. Pp. 730–739.

50. Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. Sample + Seek: Approximating aggregates with distribution precision guarantee. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2016. Pp. 679–694.

51. Jennie Duggan, Olga Papaemmanouil, Leilani Battle, and Michael Stonebraker. Skew-aware join optimization for array databases. In *SIGMOD, ACM*. 2015.

52. Earl P Duque, Daniel E Hiepler, Robert Haimes, Christopher P Stone, Steven E Gorrell, Matthew Jones, and Ronald A Spencer. Epic-an extract plug-in com- ponents toolkit for in-situ data extracts architecture. In *22nd AIAA Computa- tional Fluid Dynamics Conference*. 2015. Pp. 3410.

53. Soumya Dutta, Han-Wei Shen, and Jen-Ping Chen. In situ prediction driven feature analysis in jet engine simulations. In *2018 IEEE Pacific Visualization Symposium (PacificVis)*. 2018. Pp. 66–75.

54. Soumya Dutta, Jonathan Woodring, Han-Wei Shen, Jen-Ping Chen, and James Ahrens. Homogeneity guided probabilistic data summaries for analysis and visu- alization of large-scale data sets. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*. 2017. Pp. 111–120.

55. Roee Ebenstein et al. Bitjoin: Executing range based joins in distributed envi- ronments. 2018.

56. Oliver Fernandes, David S Blom, Steffen Frey, Alexander H Van Zuijlen, Hester Bijl, and Thomas Ertl. On in-situ visualization for strongly coupled partitioned fluid-structure interaction. *Coupled Problems*. 2015.

57. Thomas Fogal, Fabian Proch, Alexander Schiewe, Olaf Hasemann, Andreas Kempf, and Jens Kru¨ger. Freeprocessing: Transparent in situ visualization via data interception. In *Eurographics Symposium on Parallel Graphics and Visualization: EG PGV:[proceedings]/sponsored by Eurographics Association in cooperation with ACM*

SIGGRAPH. Eurographics Symposium on Parallel Graphics and Visualization. NIH Public Acces. 2014. Pp. 49.

58. Denis Foley and John Danskin. Ultra-performance pascal gpu and nvlink interconnect. *IEEE Micro*. 2017. 37(2):7–17.

59. Mike Folk et al. {HDF5}: A file format and {I/O} library for high performance computing applications. In *SC*. 1999. Vol. 99.

60. Mike Folk, Gerd Heber, and Quincey Koziol. An overview of the HDF5 technology suite and its applications. 2011.

61. Henning Funke and Jens Teubner. Data-parallel query processing on non-uniform data. *Proceedings of the VLDB Endowment*. 2020. 13(6):884–897.

62. Francesco Fusco, Michail Vlachos, Xenofontas Dimitropoulos, and Luca Deri. Indexing million of packets per second using GPUs. *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*. 2013. Pp. 327–332.

63. Isaac Gelado and Michael Garland. Throughput-oriented GPU memory allocation. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP*. 2019. Pp. 27–37.

64. William F Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, et al. Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*. 2020. 12:100561.

65. Luke Gosink, John Shalf, Kurt Stockinger, Kesheng Wu, and Wes Bethel. HDF5-FastQuery: Accelerating complex queries on HDF datasets using fast bitmap indices. In *Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM*. 2006. Pp. 149–158.

66. Anshuman Goswami, Yuan Tian, Karsten Schwan, Fang Zheng, Jeffrey Young, Matthew Wolf, Greg Eisenhauer, and Scott Klasky. Landrush: Rethinking in-situ analysis for gpgpu workflows. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 2016. Pp. 32–41.

67. Naga Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. Gputerasort: high performance graphics co-processor sorting for large database management. In

Proceedings of the 2006 ACM SIGMOD international conference on Management of data. 2006. Pp. 325–336.

68. Goetz Graefe. Volcano—An Extensible and Parallel Query Evaluation System. *IEEE Transactions on Knowledge and Data Engineering*. 1994. 6(1):120–135.

69. Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*. 1997. 1(1):29–53.

70. Oded Green, Robert McColl, and David A Bader. Gpu merge path: a gpu merging algorithm. In *Proceedings of the 26th ACM international conference on Supercomputing*. 2012. Pp. 331–340.

71. Oded Green, Pavan Yalamanchili, and Llu´ıs-Miquel Mungu´ıa. Fast triangle counting on the gpu. In *Proceedings of the 4th Workshop on Irregular Applica- tions: Architectures and Algorithms*. 2014. Pp. 1–8.

72. Sudipto Guha and Kyuseok Shim. A note on linear time algorithms for maximum error histograms. *IEEE Transactions on Knowledge and Data Engineering*. 2007. 19(7):993–997.

73. Deke Guo, Jie Wu, Honghui Chen, Ye Yuan, and Xueshan Luo. The dy- namic bloom filters. *IEEE Transactions on Knowledge and Data Engineering*. 2009. 22(1):120–133.

74. Gheorghi Guzun, Guadalupe Canahuate, David Chiu, and Jason Sawin. A tunable compression framework for bitmap indices. In *ICDE '14*. 2014.

75. Robert Haimes. pv3-a distributed system for large-scale unsteady cfd visualization. In *32nd Aerospace Sciences Meeting and Exhibit*. 1994. Pp. 321.

76. Bingsheng He, Mian Lu, Ke Yang, Rui Fang, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. Relational query coprocessing on graphics processors. *ACM Transactions on Database Systems*. 2009. 34(4):1–39.

77. Bingsheng He, Ke Yang, Rui Fang, Mian Lu, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. Relational joins on graphics processors. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2008. Pp. 511–524.

78. Jiong He, Mian Lu, and Bingsheng He. Revisiting Co-Processing for hash joins on the coupled CPU-GPU architecture. *Proceedings of the VLDB Endowment*. 2013. 6(10):889–900.

79. Jiong He, Shuhao Zhang, and Bingsheng He. In-cache query co-processing on coupled CPU-GPU architectures. 2014.

80. Randy Heiland and M Pauline Baker. A survey of co-processing systems. *CEWES MSRC PET Technical Report*. 1998. Pp. 52–98.

81. Max Heimel, Michael Saecker, Holger Pirk, Stefan Manegold, and Volker Markl. Hardware-oblivious parallelism for in-memory column-stores. *Proceedings of the VLDB Endowment*. 2013. 6(9):709–720.

82. John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier. 2011.

83. Mert Hidayetolu, Tekin Bi¸cer, Simon Garcia De Gonzalo, Bin Ren, Doa Gu¨rsoy, Rajkumar Kettimuthu, Ian T. Foster, and Wen Mei W. Hwu. MemXCT: Memory-centric X-ray CT reconstruction with massive parallelization. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*. 2019. Pp. 1–56.

84. Tomas Karnagel, Dirk Habich, and Wolfgang Lehner. Adaptive work placement for query processing on heterogeneous computing resources. *Proceedings of the VLDB Endowment*. 2017. 10(7):733–744.

85. Martin Hirzel, Robert Soul´e, Scott Schneider, Bugra Gedik, and Robert Grimm. A catalog of stream processing optimizations. *ACM Computing Surveys*. 2014. 46(4):1–34.

86. Nicolas Steven Holliman, Manu Antony, James Charlton, Stephen Dowsland, Philip James, and Mark Turner. Petascale cloud supercomputing for terapixel visualization of a digital twin. *IEEE Transactions on Cloud Computing*. 2019.

87. Xiaohuang Huang, Christopher I. Rodrigues, Stephen Jones, Ian Buck, and Wen Mei Hwu. XMalloc: A scalable lock-free dynamic memory allocator for many-core machines. In *Proceedings - 10th IEEE International Conference on Computer and*

Information Technology, CIT-2010, 7th IEEE International Conference on Embedded Software and Systems, ICESS-2010, ScalCom-2010. 2010. Pp. 1134–1139.

88. Xiaohuang Huang, Christopher I Rodrigues, Stephen Jones, Ian Buck, and Wen- mei Hwu. Scalable SIMD-parallel memory allocation for many-core machines. *The Journal of Supercomputing*. 2013. 64(3):1008–1020.

89. Yasuhiko Igarashi, Kenji Nagata, Tatsu Kuwatani, Toshiaki Omori, Yoshinori Nakanishi-Ohno, and Masato Okada. Three levels of data-driven science. In *Journal of Physics: Conference Series*. IOP Publishing, 2016. Vol. 699. P. 12001.

90. Zeljko Ivezic et al. Lsst: from science drivers to reference design and anticipated data products. *arXiv preprint arXiv:0805.2366*. 2008.

91. Edwin H Jacox and Hanan Samet. Metric space similarity joins. *ACM Transactions on Database Systems (TODS)*. 2008. 33(2):7.

92. John Jenkins et al. ALACRITY: Analytics-driven lossless data compression for rapid in-situ indexing, storing, and querying. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems X*. 2013.

93. Christopher Jermaine. Robust estimation with sampling and approximate pre-aggregation. In *Proceedings - 29th International Conference on Very Large Data Bases, VLDB 2003.* Elsevier. 2003. Pp.886-897.

ДОДАТОК А
(обов'язковий)

Міністерство освіти і науки України
Хмельницький національний університет

ЗБІРНИК НАУКОВИХ ПРАЦЬ
за матеріалами XIV Всеукраїнської науково-практичної конференції
«Актуальні проблеми комп'ютерних наук АПКН-2022»

18-19 листопада 2022

Хмельницький 2022

Островський Д.О.

Хмельницький національний університет

МЕТОДИ ПЕРЕВІРКИ ТРАНСФОРМАЦІЇ ТА МОДЕЛЮВАННЯ КЕШУ КОМП'ЮТЕРА

Більш потужні процесори та прискорювачі, які додаються до систем призводять до того, що більшість сучасних обчислювальних систем містять дедалі складніший стек вводу-виводу, починаючи від традиційних дискових файлових систем і закінчуючи гетерогенними прискорювачами з окремими просторами пам'яті. Результатом роботи є нові методи перевірки трансформації та моделювання кешу комп'ютера, які надають можливість покращити обробку великих даних.

More powerful processors and accelerators added to systems mean that most modern computing systems contain an increasingly complex I/O stack, ranging from traditional disk file systems to heterogeneous accelerators with separate memory spaces. The result of the work is new methods for verifying the transformation and modeling of the computer cache, which provide an opportunity to improve the processing of big data.

У зв'язку з великою кількістю даних, які генеруються щоденно, ефективне зберігання та запити таких даних отримують все більшу актуальність [1-3]. Як правило, ці данні багатовимірні і можуть бути представлені за допомогою моделі даних масиву. Але такі методи втрачають ефективність, коли відбувається суттєве нарощування даних.

Наявність все більш потужних процесорів та прискорювачів, які додаються до систем призводить до того, що більшість сучасних обчислювальних систем містять дедалі складніший стек вводу-виводу, починаючи від традиційних дискових файлових систем і закінчуючи гетерогенними прискорювачами з окремими просторами пам'яті. Ефективний доступ до такого складного стеку вводу-виводу в обробці масиву є важливим для використання великої обчислювальної потужності сучасних обчислювальних платформ. Одним із ключів до досягнення такої ефективності є визначення місця, де генеруються або зберігаються дані, і відповідно до вибору відповідних стратегій представлення та обробки.

Тому актуальним науковим завданням є оптимізації обробки масивів у таких складних стеках вводу-виводу. Напрямами для покращення є такі: вибір подання даних для використання та місце зберігання і обробки таких даних.

UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1014897037

Дата перевірки:
03.05.2023 10:42:25 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
03.05.2023 10:45:08 EEST

ID користувача:
100005591

Назва документа: **Островський_Методи перевірки трансформації та моделювання кешу комп'ютера**

Кількість сторінок: **84** Кількість слів: **21113** Кількість символів: **161695** Розмір файлу: **221.31 KB** ID файлу: **1014594305**

7.06% Схожість

Найбільша схожість: **0.8%** з Інтернет-джерелом (https://dl.acm.org/doi/10.1145/3335783.3335790)

6.46% Джерела з Інтернету	475	Сторінка 86
1.48% Джерела з Бібліотеки	99	Сторінка 91

0.25% Цитат

Цитати	4	Сторінка 92
Посилання	1	Сторінка 92

0% Вилучень

Немає вилучених джерел

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 0.0%

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

ID: 112934 Назва: МКР Методи перевірки трансформації та моделювання кешу комп'ютера Додано в БД: 2023-05-03 Автора: Островський Д. Керівники: Березька К. Консультанти: Опоненти:	Документ		Сумарний збіг по Бази Даних	
	Символи	Лексеми	Символи	Лексеми
	151789	1253	741 (0%)	12 (1%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Островський Денис Олексійович

Тема: Методи перевірки трансформації та моделювання кешу комп'ютера

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість сторінок записки 86

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є розробка методів перевірки трансформації та моделювання кешу комп'ютера.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі проаналізовано відомі методи для перевірки трансформації та моделювання кешу комп'ютера, виділено недоліки та зроблено постановку задачі дослідження для вирішення проблем. В другому розділі удосконалено метод перевірки трансформації кешу комп'ютера за допомогою сховища масивів з компактним інтегрованим індексом. В третьому розділі удосконалено метод перевірки трансформації кешу комп'ютера. В четвертому розділі реалізовано експериментальні дослідження розроблених рішень.

4. Позитивні сторони роботи: отримання двох пунктів наукової новизни

5. Негативні сторони роботи:

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на високому науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: добре.

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи)
Партенюк Валерій Володимирович, зав. кафедрою „Біотехнології, харчових виробництв і інженерних технологій та робототехніки" ХНУ

"18" 05 2023 р. (підпис)

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

Островському Денису Олексійовичу
ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (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.

22 квітня 2023 року

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

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

Назва: «Метод та підсистема ідентифікації користувача кіберфізичної системи «Розумний будинок»»

Автор: Островський Денис Олексійович

Спеціальність: 123 – Компютерна інженерія

Освітня програма: освітньо-наукова

Науковий керівник: Березька Катерина Михайлівна, д.т.н, професор

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) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10-40 джерелами на один фрагмент речення.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості Unicheck, складає 7.06% і адресується до 93 першоджерел; та системою Anti-Plagiarism складає 1%, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ К.М. Березька

Гарант ОП _____ О. С. Савенко

Завідувач кафедри КІІСч _____ Т. О. Говорущенко