https://doi.org/10.31891/2219-9365-2023-76-25
УДК 004.71

ОСТРОВСЬКИЙ Денис
Хмельницький національний університет
https://orcid.org/0009-0002-1747-9444
e-mail: azatot2014@gmail.com
ЛИСИЙ Андрій
Хмельницький національний університет
https://orcid.org/0009-0001-0065-9740
e-mail: andrii.lysyi1@gmail.com
СВИСТУН Сергій
Хмельницький національний університет
https://orcid.org/0009-0009-8210-6450
e-mail: svystuns@khmnu.edu.ua
ОНИШКО Оксана
Хмельницький національний університет
https://orcid.org/0000-0002-2125-4160
e-mail: van4o@ukr.net
СЕРГЕЄВ Євгеній
Хмельницький національний університет
https://orcid.org/0009-0008-9877-9863
e-mail: ysierhieiev@gmail.com

МЕТОД, МОДЕЛЮВАННЯ ТА ОЦІНЮВАННЯ ПРИСКОРЕННЯ ОБ'ЄДНАННЯ МАСИВІВ З ІНТЕГРОВАНИМ ІНДЕКСОМ ЗНАЧЕНЬ

Розглянуто 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.

В цій роботі розроблено метод, здійснено моделювання та оцінювання прискорення об'єднання масивів з інтегрованим індексом значень. Об'єднання індексованих масивів потребувало розробки нового методу оцінювання прискорення об'єднання масивів з інтегрованим індексом значень, бо розмірності та подання даних в їх елементах суттєво різняться в різних прикладних задачах. 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. Для ефективного об'єднання масивів з різними схемами фрагментування було здійснено реалізацію та моделювання різних типів об'єднання.

Напрямами подальших досліджень є удосконалення архітектури системи, яка зберігає масиви з інтегрованою підтримкою індексу та в якій здійснюватиметься автоматичне їх об'єднання.

Проведені експерименти, моделювання та оцінювання прискорення об'єднання масивів з інтегрованим індексом значень підтверджують можливість практичної реалізації розробленого методу.

Ключові слова: сховище масивів, інтегровані індекси, метод, моделювання, оцінювання.

OSTROVSKY Denis, LYSYI Andriy, SVISTUN Sergey,
ONYSHKO Oksana, SERGEYEV Yevgeny
Khmelnitskyi National University

METHOD, MODELING AND EVALUATION OF ACCELERATION OF COMBINING ARRAYS WITH INTEGRATED VALUES INDEX

The model of the array data is considered, such as raster indexes and inverted lists that can be used to encode the items of the elements in the data set, reference information about the format of floating numbers and compression, as well as the architecture of graphic processors. The array stores ordered multidimensional data. Unlike relational databases, where the cortas are stored disordered, the values in the array are ordered and organized by its size, which contributes to a much faster search speed.

This work developed a method, modeling and evaluating the acceleration of the association of arrays with an integrated index of values. The association of indexed arrays required the development of a new method of evaluating the acceleration of the association of arrays with an integrated index of values, as the dimension and submission of data in their elements differ significantly in various applied problems. In general, the structure of unification of indexed arrays corresponds to the general steps of combining simple arrays. The key differences are that the indexed arrays are not only organized by dimensional coordinates, but also organized with different bits that retain attributes. The grainness of the treatment, thus, is no longer segments, but subsets in each set. In order to effectively combine arrays with different fragmentation schemes, different types of association were implemented and modeling.

The areas of further research are to improve the architecture of the system that stores arrays with integrated index

support and which will automatically combine them. Experiments, modeling and evaluation of acceleration of the association of arrays with an integrated index of values confirm the possibility of practical implementation of the developed method.
Keywords: array repositories, integrated indices, method, modeling, evaluation.

Постановка проблеми у загальному вигляді
та її зв'язок із важливими науковими чи практичними завданнями

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 вектору до кортежу атрибутів. Масив можна визначити як відображення один до одного з вектору 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 [1], 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.

Таким чином, розробка методу прискореного об'єднання масивів з компактним інтегрованим індексом, є актуальною науковою задачею і її результати можуть бути використані при проєктуванні систем із сховищами масивів різного призначення.

Аналіз відомих рішень щодо сховищ масивів та їх об'єднання

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 [2-9], а також на наукових наборах даних [10-14]. 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 [5], гістограми [6], вейвлети [7], і ескізи даних [8]. 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 [8], 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 [15]. Непрактично зберігати растровий або перевернутий список для кожного унікального плаваючого числа в наборі даних. Таким чином, потрібна певна 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 [16] передбачає наявність апріорних знань частоти запиту, тому часто не підходять у випадку, коли умова та діапазон запиту можуть змінюватися.

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 [16].

Виклад основного матеріалу

Об'єднання індексованих масивів потребує подальшого дослідження та синтезу нових методів, оскільки розмірності та подання даних в їх елементах суттєво різняться в різних прикладних задачах. 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 рівноз'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.

Схема виводу така. Першим кроком є визначення форми вихідного масиву. Об'єднання рівноцінності та ціннісної подібності зосереджується на клітинках 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 так: 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.

Розглянемо продуктивність запиту об'єднання подібності значень в індексованих наборах даних. Як і у випадку з рівнооб'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, а також продуктивність виконання того ж запиту на растровому зображенні вторинного показника приблизно. 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 показана і порівнюється через показники об'єднання подібності значень на одних і тих же 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. А 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 втричі більше, а продуктивність індексованого об'єднання порівнянна з оператором простого приєднання, коли селективність вища, на рівні 50% - 70% відсотків. Швидкість об'єднання подібності індексованих значень дещо нижча, ніж виконання рівно об'єднання на тому самому наборі даних, оскільки виконання об'єднання в індексних наборах даних передбачає перебітування набору даних, а також порівняння одного біту у вхідному масиві з декількома бітами в іншому вхідному масиві. Приблизний метод растрових індексів є найшвидшим, але він дає помилкові спрацьовування. Виконання граничної перевірки значно знижує продуктивність до рівня, порівнянного з простим об'єднанням двох простих наборів даних. Це пов'язано з тим, що всі сегменти в простих масивах все ще повинні бути завантажені в пам'ять, зводячи

перевагу набагато меншої кількості вводу-виводу, яким користуються растрові індекси. Розбиваємо час на чотири частини: введення-виведення, який включає час читання даних з диска необхідних метаданих; сканування та фільтр, включає час сканування даних та фільтрацію даних на основі умови запиту; час на порівняння елементів, який витрачено для порівняння двох комірок і перевірки на схожість значення один на одного. Для індексованого об'єднання розбивка, також, включає час перегрупування підмножини, тобто час повторного розділення бітів однієї сторони, щоб було зв'язування двох масивів для запиту на об'єднання схожості значень. 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 ні. 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. Вплив порогу відстані на значення та подібність розмірів є суттєвим. 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 заданий. 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.

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. 1 зображено основні 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.

Таким чином, в менеджер зберігання масивів, на якому ґрунтується реалізація, включено 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.

Порівняння результатів експериментів із розробленим методом прискорення об'єднання масивів з інтегрованим індексом значень

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, представляють собою два двовимірних 64-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 подібності для отримання значущих результатів, дискретизуємо згенеровані набори даних до однозначної точності. Використані реальні набори даних є частиною.

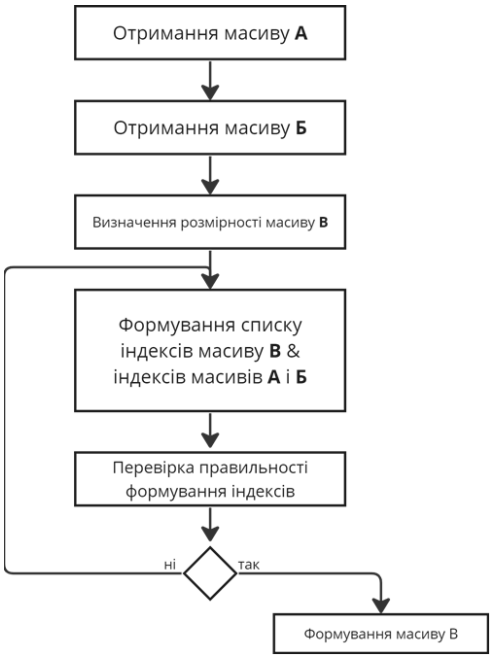

Рис. 1. Кроки методу

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 однопоточний масив, з нестисненим розміром. 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.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Розроблено метод, здійснено моделювання та оцінювання прискорення об'єднання масивів з інтегрованим індексом значень. Об'єднання індексованих масивів потребувало розробки нового методу оцінювання прискорення об'єднання масивів з інтегрованим індексом значень, бо розмірності та подання даних в їх елементах суттєво різняться в різних прикладних задачах. 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. Для ефективного об'єднання масивів з різними схемами фрагментування було здійснено реалізацію та моделювання різних типів об'єднання.

Напрямами подальших досліджень є удосконалення архітектури системи, яка зберігає масиви з інтегрованою підтримкою індексу та в якій здійснюватиметься автоматичне їх об’єднання.

Література

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

2. Truls A Bjørklund et al. Inverted indexes vs. bitmap indexes in decision support systems. In *CIKM ’09*, 2009.

3. 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)*, pages 557–558. IEEE, 2010.

4. 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*, pages 385–396. ACM, 2014.

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

6. Christian Böhm, Bernhard Braunmüller, Florian Krebs, and Hans-Peter Kriegel. Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data. In *ACM SIGMOD Record*, volume 30, pages 379–388.ACM, 2001.

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

8. 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*, pages 95–106, 2008.

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

10. Stefan Büttcher, Charles L A Clarke, and Gordon V Cormack. *Information retrieval: Implementing and evaluating search engines*. Mit Press, 2016.

11. 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.

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

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

14. 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*, 10(2-3):111–122, 2000.

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

16. Островський, Д., Лисий, А., Свистун, С., Онишко, О., & Сергєєв, Є. (2023). Архітектура сховища масивів з компактним інтегрованим індексом. *MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES*, (2), 146–152. https://doi.org/10.31891/2219-9365-2023-74-18

References

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

2. Truls A Bjørklund et al. Inverted indexes vs. bitmap indexes in decision supportsystems. In *CIKM '09*, 2009.

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

4. 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*, pages 385– 396. ACM, 2014.

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

6. Christian Böhm, Bernhard Braunmüller, Florian Krebs, and Hans-Peter Kriegel. Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data. In *ACM SIGMOD Record*, volume 30, pages 379–388.ACM, 2001.

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

8. 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*, pages 95–106, 2008.

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

10. Stefan Büttcher, Charles L A Clarke, and Gordon V Cormack. *Information retrieval: Implementing and evaluating search engines*. Mit Press, 2016.

11. 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.

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

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

14. 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*, 10(2-3):111–122, 2000.

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

16. Островський, Д., Лисий, А., Свистун, С., Онишко, О., & Сергєєв, Є. (2023). Архітектура сховища масивів з компактним інтегрованим індексом. *MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES*, (2), 146–152. https://doi.org/10.31891/2219-9365-2023-74-18