УДК 004.891

к.т.н., доц. **Джулій В.М.** (ХмНУ)
к.т.н., доц. **Чешун В.М.** (ХмНУ)
Лєнков О.С. (НАУ)

АРХІТЕКТУРА ОРГАНІЗАЦІЇ СИСТЕМИ ЗАХИСТУ БАЗ ДАНИХ З КОЛОНКОВИМ ПРЕДСТАВЛЕННЯМ ДАНИХ

У статті розглядаються нові принципи та підходи до проектування систем захисту баз даних з колонковим представленням. Описана загальна архітектура організації системи захисту баз даних з використанням додаткової програмної системи, що забезпечує зв'язок звичайної реляційної СУБД з колонковим представленням даних. Колонкові індекси створюються та підтримуються в розподіленій оперативній пам'яті обчислювального кластера. При створенні інформаційних систем з такою конфігурацією необхідно підключити підсистему відновлення колонкових індексів після збою. Результати експериментів показали, що підходи і методи паралельного виконання запитів класу OLAP на базі доменно-колонкової моделі демонструють хорошу масштабованість для запитів з великою селективністю, які є типовими для сховищ даних. Ефективність використання системи знижується при зменшенні розмірів бази даних і при збільшенні розмірів результуючого відношення. Результати, отримані в ході роботи, можуть застосовуватися при створенні масштабованих колонкових співпроцесорів SQL-СУБД, що дозволить обробляти надвеликі сховища даних на кластерних обчислювальних системах, у тому числі з вузлами, що включають в себе багатоядерні прискорювачі типу GPU або MIC.

Ключові слова: надвеликі об'єми даних, колонкові індекси, колонкові бази даних, OLTP, OLAP, інформаційні системи.

Вступ. Постійно прогресуюче інформаційне навантаження, що пов'язане з розвитком соціальних мереж, електронних бібліотек, геоінформаційних систем, інтелектуальним аналізом даних та ін., призводить до необхідності вдосконалення методів обробки надвеликих об'ємів інформації.

Згідно з прогнозами аналітичної компанії IDC до 2020р. кількість даних у світі досягне 40 зеттабайт. У зв'язку з появою задач, що вимагають обробки надвеликих баз даних, необхідні нові ефективні методи паралельної обробки та аналізу таких обсягів даних на багатопроцесорних обчислювальних системах. Фактично єдиним ефективним вирішенням проблеми зберігання, обробки і захисту надвеликих об'ємів даних є використання паралельних систем баз даних, що забезпечують паралельну обробку запитів на багатопроцесорних обчислювальних системах.

Великі компанії завжди відчували необхідність у вирішенні задач аналізу даних, які перетворять великі обсяги вхідних даних в інформацію, необхідну для своєчасного прийняття рішень. Паралельні системи баз даних («Gamma» [1], «Teradata» [2]) були одними з перших систем для вирішення даної проблеми.

Багато інновацій та покращень в продуктивності потребують в своєчасному і економічно ефективному аналізі великих обсягів даних. За останнє десятиліття дана потреба привела до появи нових значних інновацій в системах аналітики надвеликих об'ємів даних. Наявність паралельних баз даних додали такі методи, як колонкове зберігання і обробка даних [3]. Одночасно з цим були розроблені нові розподілені масиви даних та обчислювальні системи (MapReduce [4], Bigtable [5]). Фактично, єдиним ефективним вирішенням проблеми зберігання і обробки надвеликих баз даних є використання паралельних систем баз даних, що забезпечують розподілену обробку запитів на багатопроцесорних обчислювальних системах з розподіленою пам'яттю [2].

Постановка задачі. Паралельні системи баз даних досягли високої продуктивності і масштабованості секціонування таблиць по вузлах в “shared-nothing cluster", в якій кожен вузол є незалежним і самодостатнім. Така горизонтальна схема розбиття включає реляційні операції, такі як фільтри з'єднання і агрегування для паралельного запуску з різних розділів таблиці, збереженої на різних вузлах.

Традиційним підходом до організації зберігання даних є строкове подання даних. Однак, при виконанні типових аналітичних запитів до баз даних необхідно читати тільки невелику частину полів, тому строкове представлення даних виявляється неефективним в даному випадку. Причиною цього є читання з диску «зайвих» полів на додаток до тих полів, які необхідні в даному запиті [3]. Можливим вирішенням цієї проблеми може бути використання механізмів колонкового представлення даних, що дозволяють отримати значно кращу продуктивність при обробці аналітичних запитів [5]. Колонкове представлення даних полягає в тому, що дані зберігаються не по рядках, а по колонках. Це означає, що з точки зору SQL-клієнта дані представлені у вигляді таблиць, але фізично ці таблиці є сукупністю колонок, кожна з яких являє собою таблицю з одного поля. Додатковою перевагою колонкового представлення є можливість використання ефективних алгоритмів стиснення і захисту даних, оскільки в одній колонці таблиці містяться дані одного типу. Стиснення може призвести до значного підвищення продуктивності алгоритмів стиснення і захисту даних, оскільки менше часу займають операції введення-виведення.

Таким чином, актуальною є задача розробки нових ефективних методів та алгоритмів паралельної обробки даних в оперативній пам'яті на сучасних багатопроцесорних обчислювальних системах з багатоядерними прискорювачами та з використанням колонкового представлення і стиснення даних. Для вирішення даної задачі можна використати індексні структури спеціального виду - розподілені колонкові індекси. Розподілені колонкові індекси дозволяють провести декомпозицію реляційних операцій, що забезпечить їх ефективне паралельне виконання на кластерних обчислювальних системах з багатоядерними прискорювачами. У даній роботі описується архітектура, процес проектування і реалізації колонкових індексів за допомогою програмної системи.

Виклад основного матеріалу досліджень. Система управління розподіленими колонковими індексами, що розміщені в оперативній пам'яті кластерної обчислювальної системи, представлена на рис. 1. Призначення даної системи – попереднє обчислення таблиці для ресурсоємних реляційних операцій за SQL-запитом.

Система включає підсистему «Координатор», що запускається на вузлі обчислювального кластера з номером 0, і програму «Виконавець», що запускається на всіх інших вузлах, виділених для роботи системи. На SQL-сервері встановлюється спеціальний драйвер, що забезпечує взаємодію з координатором по протоколу TCP/IP.

Рис. 1. Взаємодія SQL-сервера з кластерною обчислювальною системою

Система керування розподіленими колонковими індексами працює з даними цілих типів 32 або 64 біта. При створенні колонкових індексів для атрибутів інших типів, їх

значення кодується у вигляді цілого числа або вектора цілих чисел. В останньому випадку довжина вектора є фіксованою і називається розмірністю колонкового індексу.

Система підтримує наступні основні операції, що доступні СУБД через інтерфейс драйвера:

– Create ColumnIndex (TableID, Column ID, Surrogate, Width, Bottom, Top, Dimension) - створення розподіленого колонкового індексу для атрибуту ColumnID таблиці TableID з параметрами: SurrogateID – ідентифікатор сурогатного ключа, Width – розрядність (32 або 64 біта); Bottom, Top – нижня і верхня межі доменного інтервалу; Dimension – розмірність колонкового індексу. Значення, що повертається: CIndexID – ідентифікатор створеного колонкового індексу.

– Insert (CIndexID, SurrogateKey, Value [*]) – додання в колонковий індекс CIndexID нового кортежу (SurrogateKey, Value [*]).

– TransitiveInsert (CIndexID, SurrogateKey, Value [*], TValue [*]) – додання в колонковий індекс CIndexID нового кортежу (SurrogateKey, Value [*]) з фрагментацією і сегментацією, обумовленими значенням TValue [*].

– Delete (CIndexID, SurrogateKey, Value [*]) – видалення з колонкового індексу кортежу (SurrogateKey, Value [*]).

– TransitiveDelete (TCIndexID, SurrogateKey, TransitiveValue [*]) – видалення з колонкового індексу кортежу (SurrogateKey, Value [*]) з фрагментацією і сегментацією, обумовленими значенням TValue [*].

Взаємодія між «Драйвером» і «Координатором» здійснюється шляхом обміну повідомленнями у форматі JSON. Кожен колонковий індекс ділиться на фрагменти, які в свою чергу діляться на сегменти. Всі сегменти одного фрагменту розташовуються в стислому вигляді в оперативній пам'яті одного процесорного вузла. План виконання запиту представлено на рис. 2.

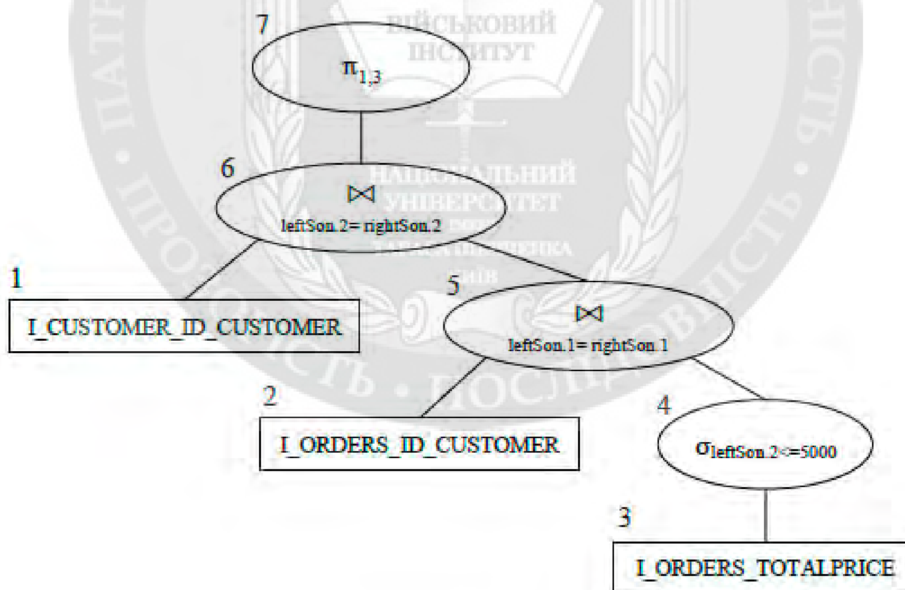

Рис. 2. Дерево плану запиту

Схема включає в себе властивість queryPlan, що описує план виконання запиту. У даному описі розрізняються вузли трьох типів: листя, внутрішні вузли і корінь дерева. Для листків необхідно вказати ідентифікатор вхідного відношення (колонкового індексу). Для кореня і внутрішнього вузла вказуються порядкові номери лівого і правого (за наявності) синів у дереві плану. Корінь відрізняється тим, що у нього немає попередників. Наведемо приклад використання оператора Execute для обчислення таблиці попередніх результатів P(A_ORDERS, A_CUSTOMER), що задається виразом реляційної алгебри. Відповідний план будується драйвером системи за наступними правилами. Атрибути вхідних і проміжних

відношень нумеруються натуральними числами зліва-направо. Для унарних операцій завжди є посилання на лівого сина в дереві плану. Позначення leftSon.1 іменує значення першого зліва атрибуту відношення, обчислюваного лівим сином. Аналогічно, rightSon.2 іменує значення другого зліва атрибуту відношення, обчислюваного правим сином. Всі вузли дерева плану нумеруються в порядку зворотного обходу дерева (лівий син, правий син, корінь). Після закінчення обробки запиту система передає драйверу таблицю попередніх обчислень в стислому вигляді. Драйвер забезпечує доступ до попередніх обчислень для SQL-сервера.

Дані (колонкові індекси та метадані), з якими працює система, зберігаються в розподіленої пам'яті кластерної обчислювальної системи. Для колонкових індексів підтримується дворівнева система їх розбиття на неперетинні частини. Розбиття колонкового індексу, побудованого для атрибуту B представлено на рис. 3.

В основі розбиття лежить домен D_B=[0,99] B.D, на якому визначений атрибут B. Домен розбивається на сегментні інтервали рівної довжини: [0; 11), [11; 22), ..., [77; 88), [88; 99]. Сегментні інтервали розбиваються на послідовні групи, які називаються фрагментними інтервалами. У прикладі на рис. 3 це наступні інтервали: [0; 22), [22; 55), [55; 99]. Кількість фрагментних інтервалів має збігатися з кількістю вузлів-виконавців. Довжини фрагментних інтервалів можуть не співпадати. Це необхідно для балансування завантаження процесорних вузлів в умовах перекосу даних. На першому етапі розподілу даних вхідний колонковий індекс розбивається на фрагменти. Кожному фрагменту відповідає певний фрагментний інтервал. Кортеж (a, b) потрапляє в даний фрагмент тоді і тільки тоді, коли значення b належить відповідному фрагментному інтервалу.

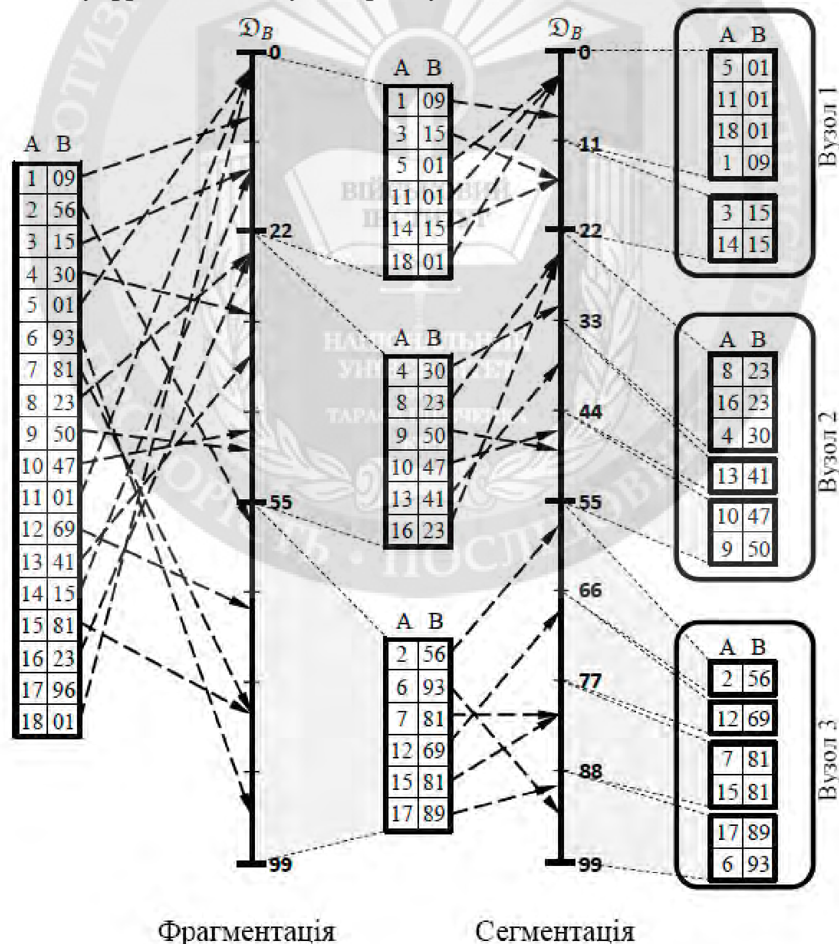

Рис. 3. Приклад дворівневого розбиття індексу на фрагменти та сегменти

Всі кортежі, що належать одному фрагменту, зберігаються на одному і тому ж процесорному вузлі. На другому етапі кожен фрагмент розбивається на сегменти. Кожному сегменту відповідає певний сегментний інтервал. Кортеж (a, b) потрапляє в даний сегмент

тільки тоді, коли значення b належить відповідному сегментному інтервалу. Усередині кожного сегментного інтервалу записи сортуються в порядку зростання значення атрибута. У разі нерівномірного розподілу значень атрибутів, по яких створений колонковий індекс, можна домогтися приблизно однакового розміру фрагментів, зсувом межі фрагментних інтервалів, як це зроблено в прикладі на рис. 2. При цьому можуть вийти сегменти різної довжини. Сегмент є найменшою одиницею розподілу робіт між процесорними ядрами. Якщо кількість сегментів невелика в порівнянні з кількістю процесорних ядер на одному вузлі, то при виконанні запиту отримаємо дисбаланс у завантаженні процесорних ядер. Проблему балансування завантаження можна ефективно вирішити, зменшивши довжину сегментних інтервалів, і тим самим збільшивши кількість сегментів. Система працює тільки з даними цілих типів (32 або 64 біта). Дані інших типів повинні бути попередньо закодовані драйвером у вигляді послідовності цілих чисел. При цьому можуть бути використані відповідні методи. У випадку, коли значення атрибута кодується у вигляді цілочисельного вектора розмірності n (наприклад, це може бути застосоване для довгих символьних рядків), домен перетворюється в n-мірний куб. У даній ситуації n-мірний куб може розбиватися на n-мірні паралелепіпеди по числу процесорних вузлів (аналог фрагментного інтервалу), кожен з яких розбивається на ще менші n-мірні паралелепіпеди по числу процесорних ядер в одному вузлі (аналог сегментного інтервалу). Для стиснення закодованих сегментів можуть використовуватися як «тяжкі» методи (наприклад, Хаффмана або Лемпеля-Зіва), так і «легкі» (наприклад, Run-Length Encoding або Null Suppression), або їх комбінації. «Легкі» методи типу Run-Length Encoding, у разі колонкового подання інформації, можуть виявитися більш ефективними, ніж «тяжкі», оскільки допускають виконання операцій над даними без їх розпакування.

Загальна логіка роботи системи представлена на рис. 4. База даних складається з двох відношень R(A, B, D) і S(A, B, C), що зберігається на SQL-сервері. Нехай нам необхідно виконати запит:

```
SELECT D, C
FROM R, S
WHERE R.B = S.B AND C<13.
```

Припустимо, що система має тільки два вузла-виконавці і на кожному вузлі є три процесорних ядра (процесорні ядра на рис. 4 промарковані позначеннями (P_{11}, ..., P_{23}). Припустимо, що атрибути R.B і S.B визначені на домені цілих чисел з інтервалу [0; 120). Сегментні інтервали для R.B і S.B визначимо наступним чином: [0; 20), [20; 40), [40; 60), [60; 80), [80; 100), [100; 120). В якості фрагментних інтервалів для R.B і S.B зафіксуємо: [0; 59] і [60,119]. Нехай атрибут S.C визначений на домені цілих чисел з інтервалу [0; 25]. Спочатку адміністратор бази даних за допомогою драйвера створює для атрибутів R.B і S.B розподілені колонкові індекси $I_{R.B}$ і $I_{S.B}$. Потім для атрибуту S.C створюється розподілений колонковий індекс $I^{B}_{S.C}$, який фрагментується і сегментується транзитивно щодо індексу $I_{S.B}$. Розподілені колонкові індекси $I_{R.B}$, $I_{S.B}$ та $I^{B}_{S.C}$ зберігаються в оперативній пам'яті вузлів-виконавців. Таким чином, отримуємо розподіл даних всередині системи, що приведено на рис. 4. При надходженні SQL-запиту, він перетворюється драйвером в план, який визначається наступним виразом реляційної алгебри:

$$\pi_{I_{R.B}.A} \to A_R, I_{S.B}.A \to A_S \left(I_{R.B} \underset{\substack{I_{R.B}.B= \\ I_{S.B}.B}}{\bowtie} \left(I_{S.B} \bowtie \sigma_{C<13}\left(I^{B}_{S.C} \right) \right) \right).$$

При виконанні драйвером операції Execute вказаний запит передається координатору у вигляді оператора у форматі JSON. Запит виконується незалежно процесорними ядрами вузлів-прискорювачів над відповідними групами сегментів. При цьому за рахунок доменної фрагментації і сегментації не вимагаються обміни даними як між вузлами-виконавцями, так і між процесорними ядрами одного вузла. Кожне процесорний ядро обчислює свою частину попереднього результату, який пересилається на вузол-координатор. Координатор об'єднує

фрагменти попередніх результатів в єдину таблицю та надсилає її драйверу, який виконує матеріалізацію цієї таблиці у вигляді відношення в базі даних, що зберігається на SQL-сервері. Після цього SQL-сервер замість вхідного SQL-оператора, виконує наступний оператор:

```
SELECT D, C
  FROM
        R INNER JOIN (
        ТПВ INNER JOIN S ON (S.A = ТПВ.AS)
        ) ON (R.A = ТПВ.AR).
```

При цьому використовуються звичайні групові індекси у вигляді B-дерев, заздалегідь побудовані для атрибутів R.A і S.A. Після виконання запиту попередні результати видаляються, а розподілені колонкові індекси $I_{R.B}$, $I_{S.B}$ та $I^{B}_{S.C}$ зберігаються в оперативній пам'яті вузлів-виконавців для подальшого використання.

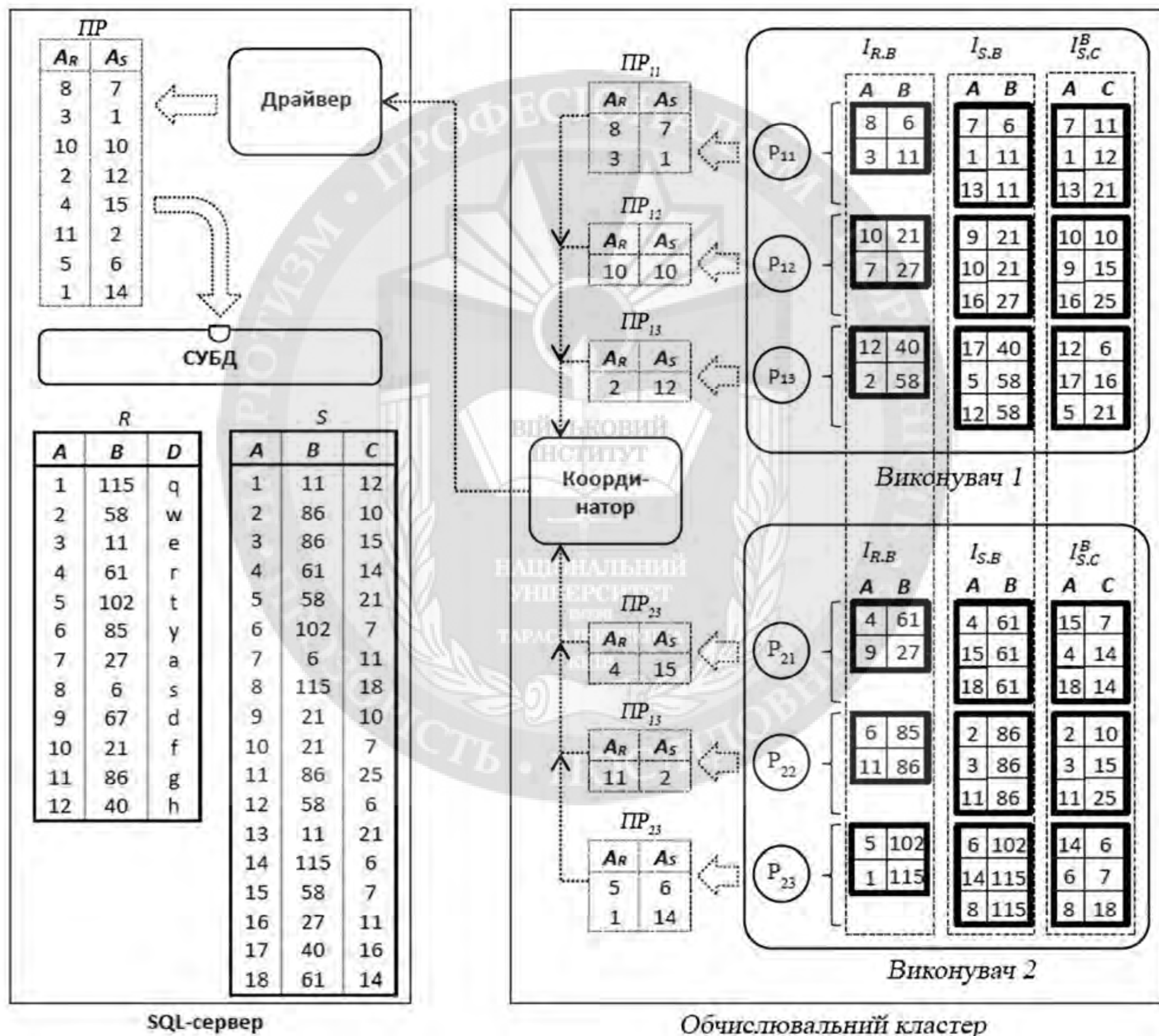

Рис. 4. Приклад обчислення попередніх результатів

На рис. 5 представлена діаграма розгортання системи, заснована на нотації стандарту UML. Дана система є розподіленою системою і включає в себе два типи вузлів: координатори та виконавці. У найпростішому випадку в системі є один вузол-координатор і кілька вузлів-виконавців, як це показано на рис. 5. Однак, якщо при великій кількості виконавців координатор стає вузьким місцем, в системі може бути декілька координаторів. У

цьому випадку вони утворюють ієрархію у вигляді збалансованого дерева, листям якого є вузли-виконавці.

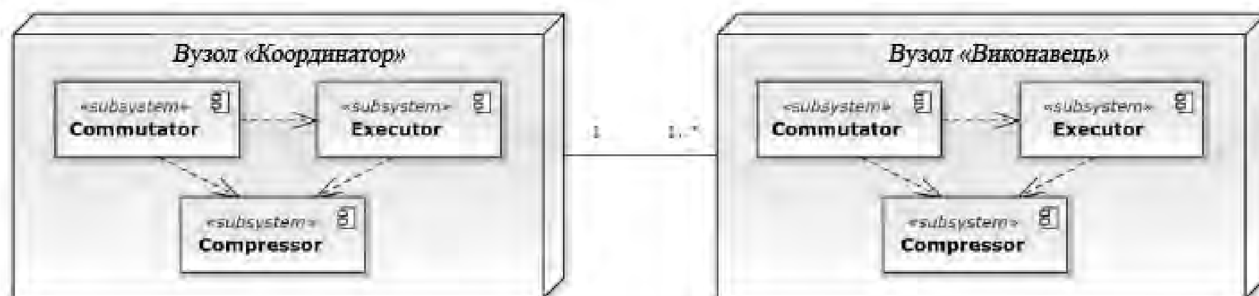

Рис. 5. Діаграма розгортання системи

Координатор і виконавець мають уніфіковану структуру, що передбачає три компоненти: Commutator (Комутатор), Executor (Виконавець), Compressor (Модуль стиснення). Проте, їх реалізації в координаторі й виконавці розрізняються. Компонент Commutator в координаторі виконує дві функції: обмін повідомленнями у форматі JSON з драйвером по протоколу TCP/IP; обмін повідомленнями з виконавцем за технологією MPI. Компонент Commutator у виконавці здійснює обмін повідомленнями з координатором за технологією MPI. Компонент Executor на виконавці організовує паралельну обробку сегментів колонкових індексів, використовуючи технологію OpenMP, і формує фрагменти попередніх результатів. Компонент Executor на координаторі об'єднує фрагменти попередніх результатів, отриманих виконавцями в єдину таблицю. Компоненти Commutator і Executor використовують компонент Compressor для стиснення та розпакування даних. При виконанні оператора CreateColumnIndex координатор додає інформацію про структуру колонкового індексу в свій локальний словник і відправляє виконавцям вказівку створити в своїх локальних словниках дескриптор з інформацією про новий колонковий індекс. При цьому координатор визначає довжину сегментного інтервалу і межі фрагментних інтервалів. Ця інформація зберігається в дескрипторі колонкового індексу. Крім цього, дескриптор включає бітову шкалу сегментів, в якій значення 1 відповідає непустим сегментам, значення 0 - пустим. При початковому створенні колонкового індексу всі сегменти на всіх вузлах-виконавців є пустими. При виконанні оператора Insert координатору передається кортеж (SurrogateKey, Value) для додання в зазначений колонковий індекс. Координатор визначає, в межі якого фрагментного інтервалу потрапляє значення Value, і пересилає цей кортеж на відповідний вузол-виконавець. Виконавець визначає номер сегментного інтервалу, до якого належить значення Value. Якщо відповідний сегмент не пустий, то кортеж додається в сегмент зі збереженням впорядкування по полю Value. Якщо відповідний сегмент порожній, то створюється новий сегмент з одного кортежу. При виконанні оператора TransitiveInsert координатору крім нового кортежу (SurrogateKey, Value) передається додаткове значення TValue, яке використовується для визначення номера фрагментного інтервалу. Кортеж (SurrogateKey, Value) разом зі значенням TValue пересилається на відповідний вузол-виконавець. Виконавець за значення TValue визначає номер сегментного інтервалу і додає новий кортеж у відповідний сегмент. При виконанні оператора Delete координатору передається сурогатний ключ SurrogateKey і значення Value, яке треба видалити. Координатор визначає, в межі якого фрагментного інтервалу потрапляє значення Value, і пересилає цей кортеж на відповідний вузол-виконавець. Виконавець визначає номер сегментного інтервалу, до якого належить значення Value, виробляє у відповідному сегменті пошук кортежа з ключем SurrogateKey і виконує його видалення. Оператор TransitiveDelete виконується аналогічно оператору Delete з тією лише різницею, що номери фрагментного і сегментного інтервалів вираховуються по транзитивності значенню TValue.

Виконання оператора Execute включає в себе дві фази: обчислення фрагментів попередніх результатів на вузлах-виконавцях; злиття фрагментів попередніх результатів в єдину таблицю на вузлі-координаторі і пересилання її на SQL-сервер. На першій фазі

процесорні ядра вибирають необроблені сегментні інтервали і виконують обчислення сегментних попередніх результатів для відповідних сегментів колонкових індексів, задіяних в запиті. Після того, як всі сегментні інтервали оброблені, отриманий фрагмент попередніх результатів пересилається на вузол-координатор в стисненому вигляді. На другій фазі координатор розпаковує отримані сегментні попередні результати і об'єднує їх в єдину таблицю. Для розпаралелювання процесу використовується технологія OpenMP. Отримана таблиця попередніх результатів пересилається на SQL-сервер. При цьому також може використовуватися стиснення даних.

Висновки. Описана загальна архітектура організації системи баз даних з використанням додаткової програмної систем, що забезпечує зв'язок звичайної реляційної СУБД з колонковим представленням даних. До складу системи входить SQL-сервер і обчислювальний кластер. На SQL-сервері встановлюється реляційна СУБД з впровадженим конектором та драйвер. На обчислювальному кластері встановлюється координатор системи. Колонкові індекси створюються та підтримуються в розподіленій оперативній пам'яті обчислювального кластера. При створенні інформаційних систем з такою конфігурацією необхідно підключити підсистему відновлення колонкових індексів після збою. Для цього копії колонкових індексів і метаданих можуть створюватися на дисках, встановлених на кожному обчислювальному вузлі.

Результати експериментів показали, що підходи і методи паралельного виконання запитів класу OLAP на базі доменно-колонкової моделі, демонструють хорошу масштабованість для запитів з великою селективністю, які є типовими для сховищ даних. Використання колонкової системи у взаємодії з СУБД PostgreSQL дозволило підвищити ефективність виконання запитів у середньому на 12%. Однак, ефективність використання системи знижується при зменшенні розмірів бази даних і при збільшенні розмірів результуючого відношення.

Результати, отримані в ході роботи, можуть застосовуватися при створенні масштабованих колонкових сопроцесорів для існуючих комерційних і вільно-поширюваних SQL-СУБД. Це дозволить обробляти надвеликі сховища даних на кластерних обчислювальних системах, у тому числі з вузлами, що включають в себе багатоядерні прискорювачі типу GPU або MIC.

ЛІТЕРАТУРА:

1. David J DeWitt, Shahram Ghandeharizadeh, Donovan A. Schneider, Allan Bricker, H-I Hsiao, and Rick Rasmussen. The Gamma Database Machine Project. IEEE Trans. on Knowledge and Data Engineering, 2(1):44–62, 1990.

2. Teradata, 2012. http://www.teradata.com.

3. Andrew Lamb, Matt Fuller, Ramakrishna Varadarajan, Nga Tran, Ben Vandiver, Lyric Doshi, and Chuck Bear. The Vertica Analytic Database: C-store 7 Years Later. Proc. of the VLDB Endowment, 5(12):1790–1801, 2012.

4. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A Distributed Storage System for Structured Data. ACM Trans. on Computer Systems, 26(2):4, 2008.

5. Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 51(1):107–113, 2008.

Рецензент: д.т.н., проф. Сбітнєв А.І., провідний науковий співробітник науково-дослідного центру Військового інституту Київського національного університету імені Тараса Шевченка

к.т.н. Джулий В.Н., к.т.н. Чешун В.Н., Ленков А.С.

АРХИТЕКТУРА ОРГАНИЗАЦИИ СИСТЕМЫ ЗАЩИТЫ БАЗ ДАННЫХ С КОЛОНКОВЫМ ПРЕДСТАВЛЕНИЕМ ДАННЫХ

В статье рассматриваются новые принципы и подходы к проектированию систем защиты баз данных с колонковым представлением. Описанна общая архитектура организации системы защиты баз данных с использованием дополнительной программной системы, обеспечивает связь обычной реляционной СУБД с колонковым представлением данных. Колонковые индексы создаются и поддерживаются в распределенной оперативной памяти вычислительного кластера. При создании информационных систем с такой конфигурацией необходимо подключить подсистему восстановления колонковых индексов после сбоя. Результаты экспериментов показали, что подходы и методы параллельного выполнения запросов класса OLAP на базе доменно-колоночной модели, демонстрируют хорошую масштабируемость для запросов с большой селективностью, которые являются типичными для хранилищ данных. Эффективность использования системы снижается при уменьшении размеров базы данных и при увеличении размеров результирующего отношения. Результаты, полученные в ходе работы, могут применяться при создании масштабируемых колонковых сопроцессоров SQL-СУБД, что позволит обрабатывать сверхбольшие хранилища данных на кластерных вычислительных системах, в том числе с узлами, включающих в себя многоядерные ускорители типа GPU или MIC.

Ключевые слова: сверхбольшие объемы данных, колонковые индексы, колонковые базы данных, OLTP, OLAP, информационные системы.

Ph.D. Julie V.M., Ph.D. Cheshun V.M., Lenkov A.S.

ARCHITECTURE OF THE PROTECTION SYSTEM DATABASE WITH COLUMNS SUBMISSION OF DATA

The article deals with new principles and approaches to the design of database systems coring data representation. Described the general architecture of the organization database system using the optional software system enables communication with a conventional relational database coring data representation. Coring indexes are created and maintained in a distributed memory computing clusters-ra. When you create information systems with such a configuration is necessary Connectivity chit subsystem index of core recovery after a crash. The experimental results showed that the approaches and methods of parallel query class OLAP-based domain-column model show good scalability for queries with pain-Scheu selectivity, which are typical for data warehousing. System efficiency decreases with decreasing size of the database and increasing the size of the resultant relation. The results obtained in the course of the work, can be used to create scalable coring coprocessor SQL-database will allow to handle extremely large data warehouse on cluster computing systems, including units, including multi-core accelerators such as GPU, or MIC.

Keywords: extremely large amounts of data, core index, core database, OLTP, OLAP, information systems.