Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи обробки запитів в системах паралельних баз даних

(тема)

Виконав:

студент II курсу, групи СПм-20-1

Бессараб Є.В.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: доц. Філімончук Т.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Бессарабу Єгору Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи Методи обробки запитів в системах паралельних баз даних

затверджена наказом по університету від “ 5 ” листопада 2021 р. № 1657 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 13 грудня 2020 р.

3. Вхідні дані до роботи
база даних
запит
паралельна обробка запитів
MPP архітектура

4. Перелік питань, що потрібно опрацювати у роботі
Аналіз сучасних методів обробки sql-запросів у СКБД
Принципи побудування систем для паралельної обробки запитів
Методи та засоби перетворення запитів для паралельного виконання
Експериментальне дослідження ефективності механізму розпаралювання запитів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 12 слайдів ф.А4

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання. Пошук літератури	09.11.2021–15.11.2021	
2	Порівняльний аналіз сучасних методів	16.11.2021–25.11.2021	
3	Розробка модифікованого методу	26.11.2021–28.11.2021	
4	Реалізація методу	29.11.2021–05.12.2021	
5	Аналіз результатів	06.12.2021–08.12.2021	
6	Оформлення ПЗ	09.12.2021–12.12.2021	

Дата видачі завдання 8 листопада 2021 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Філімончук Т.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 68 с., 10 рис., 5 табл., 1 дод., 26 джерел.

БАЗА ДАНИХ, ВІДНОШЕННЯ, ЗАЛЕЖНІСТЬ, НАДЛИШКОВІСТЬ ДАНИХ, АТРИБУТ, ПРОЕКТУВАННЯ, АЛГОРИТМ, РЕЛЯЦІЙНА СТРУКТУРА, ОПЕРАЦІЯ.

Метою кваліфікаційної роботи є дослідження методів паралельного виконання SQL – запитів в гетерогенних архітектурах для підвищення продуктивності розподілених СКБД.

На базі проведеного дослідження був зроблений висновок, що найкращі загальні показники продуктивності системи можна досягнути, розробляючи систему на базі MPP архітектури, допускаючи використання SMP машин у вигляді вузлів системи, що фактично перетворює її у гібридну. Таке рішення, в свою чергу, робить можливим використання існуючої обчислювальної мережі та встановленого програмного забезпечення, тобто, забезпечує низьку вартість інтеграції.

ABSTRACT

Master’s thesis: 68 pages, 10 figures, 5 tables, 1 appendices, 26 sources.

DATABASE, RELATIONSHIP, DEPENDENCE, SURPLUS DATA, ATTRIBUTE, DESIGN, ALGORITHM, RELATIVE STRUCTURE, OPERATION.

The purpose of the qualification work is to study the methods of parallel execution of SQL - queries in heterogeneous architectures to improve the performance of distributed databases.

Based on the study, it was concluded that the best overall performance of the system can be achieved by developing a system based on MPP architecture, allowing the use of SMP machines in the form of system nodes, which actually turns it into a hybrid. This solution, in turn, makes it possible to use the existing computer network and installed software, that is, provides a low cost of integration.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 9
1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ОБРОБКИ SQL-ЗАПРОСІВ У СКБД 12
1.1 Архітектури паралельних систем 12
1.2 Класифікація архітектур паралельних систем баз даних 14
1.3 Масштабованість паралельних систем обробки даних 17
1.4 Методи пошуку субоптимальних алгоритмів використання запиту 19
1.5 Методи обробки даних реляційної структури 22
2 ПРИНЦИПИ ПОБУДУВАННЯ СИСТЕМ ДЛЯ ПАРАЛЕЛЬНОЇ ОБРОБКИ ЗАПИТІВ 25
2.1 Дослідження архітектур систем паралельного виконання запитів з використанням надмірного представлення інформації 25
2.2 Дослідження та розробка методу декомпозиції запиту 31
2.3 Дослідження та модифікація методу паралельного виконання запитів 34
2.4 Апріорні оцінки часу виконання запитів 36
3 МЕТОДИ ТА ЗАСОБИ ПЕРЕТВОРЕННЯ ЗАПИТІВ ДЛЯ ПАРАЛЕЛЬНОГО ВИКОНАННЯ 44
3.1 Перетворення запитів за умов фільтрації в інструкції WHERE 44
3.2 Перетворення запиту з агрегуючими функціями та оператором групування атрибутів 46
4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ МЕХАНІЗМУ РОЗПАРАЛЮВАННЯ ЗАПИТІВ 49
4.1 Структура, особливості реалізації, методика вибору алгоритму перетворення 49

4.2 Результати експериментального дослідження розпаралелювання запитів..... 54
ВИСНОВКИ..... 57
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 58
ДОДАТОК А..... 61
Графічний матеріал кваліфікаційної роботи..... 61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

СКБД – система керування базами даних

CD (Clustered-Disk) – архітектура із SD кластерами, об’єднаними за принципом SN

CE (Clustered-Everything) – архітектура з SE кластерами, об’єднаними за принципом SN

NUMA (Non-Uniform Memory Architecture) – мультипроцесорні системи з неоднорідним доступом до пам’яті

MPP (Massively Parallel Processing) – масово-паралельні системи

OLAP (Online Analytical Processing) – технологія обробки інформації, що включає складання та динамічну публікацію звітів та документів

OLTP (Online Transaction Processing) – обробка транзакцій у реальному часі

SMP (Symmetric Multiprocessor) – симетричні мультипроцесорні системи

SD (Shared-Disks) – архітектура з дисками, що розділяються

SE (Shared-Everything) – архітектура з пам’яттю і дисками, що розділяються

SMP (Symmetric Multiprocessor) – симетричні мультипроцесорні системи

SN (Shared-Nothing) – архітектура без спільного використання ресурсів.

ВСТУП

Підвищення продуктивності систем керування базами даних (СКБД) є невирішеною проблемою та дуже актуальною зараз. Зі збільшенням кількості користувачів, що працюють з базою даних та нарощування обсягів інформації відповідно потрібно підвищувати швидкодію баз даних для отримання прийнятного часу виконання/реакції на запит користувача. Ефективним рішенням та економічно обґрунтованою альтернативою однопроцесорним СКБД є паралельні СКБД, які одночасно функціонують на кількох процесорах. Дана технологія дозволяє об’єднати кілька ЕОМ малої потужності для отримання того ж рівня продуктивності, як і за однієї (значно потужніше). Таким чином отримуємо виграш у масштабованості та надійності системи у порівнянні з однопроцесорною СКБД. Звичним явищем є використання СКБД у всіх сферах людської діяльності, що виконують як зберігання, так і переробку даних. Аналізуючи існуючі методи роботи з даними, можна зробити висновок, що вони не повною мірою відповідають усім пред’явленим до них зараз вимогам. Інтенсивні дослідження у сфері баз даних свідчать про це. Природно формування надвеликих баз даних, що пов’язане з розширенням видів та сфер застосування СКБД. Нині вони використовують у електронної комерції, формуванні електронних бібліотеках, системах збору статистики, мультимедійних архівів тощо. Стрімкий розвиток паралельних та розподілених СКБД з боку компаній різних напрямів викликало підвищення попиту до таких технологій. Пов’язано це з тим, що експеримент із автоматизації філій практично скрізь пройшов успішно. На цьому компанії зупинятися не збираються, наступна мета, якої вони прагнуть – створення єдиної бази даних, яка зможе давати можливість оперативного доступу та аналізу даних. Крім цього, важливо створити рішення специфічних завдань для різних компаній, тобто. для компаній у різних сферах діяльності потрібні продукти, які відрізнятимуться

у деяких аспектах. На сьогоднішній день єдиним ефективним рішенням для обробки та зберігання надвеликих баз даних залишається використання паралельних систем баз даних з реплікацією інформації, що зберігається, яка забезпечує паралельну обробку різних запитів на багатопроцесорних обчислювальних системах. У 80-х роках вчені почали серйозно займатися дослідженнями у галузі паралельних СКБД. За останніх 30 років такі системи зробили величезний стрибок, пройшовши шлях від науково-дослідних прототипів до повнофункціональних комерційних систем. Найуспішнішими проектами в даній галузі можна вважати DB2 Parallel Edition, NonStop та NCR Teradata. Але вартість спеціалізованого ПЗ для таких систем порівнянна, у деяких випадках навіть вища за вартість апаратної складової.

Подібні системи можуть обробляти бази даних розміром у десятки терабайт, поєднуючи сотні процесорів та жорстких дисків. Але до сьогодні у сфері паралельних баз даних існують напрями, у яких ведуться дослідження оптимізації запитів до виконання у складі паралельних і розподілених систем баз даних. Багато дослідників вказують на обмеження в масштабованості більшості існуючих паралельних баз даних через обмеження, притаманну архітектурі систем, що підтримують їхню роботу. Через велику кількість процесорів у системах такого роду виникають конфлікти доступу до пам’яті, що розділяється, що призводить до занепаду загальної продуктивності системи. Тому реальна масштабованість SMP систем обмежується 32 процесорами. В одному зі звітів, написаному провідними фахівцями в галузі систем управління базами даних, про напрямки досліджень в області баз даних, йдеться, що в майбутньому великі організації матиме в своєму розпорядженні бази даних обсягом, що перевищує один петабайт. За можливості використання стандартного програмного забезпечення для широко масштабованих моссово-паралельних архітектур суттєво скорочується вартість рішення. Як подібні системи можна розглядати набір існуючих серверів СКБД, які входять до складу високошвидкісної локально-обчислювальної мережі, які спочатку не використовують паралельної

обробки даних. Грамотне використання їх ресурсів для паралельної обробки дозволить уникнути придбання дорогих готових продуктів, що використовують складну програмно-апаратну конфігурацію. На сьогоднішній день всі підходи до паралельної обробки запиту базуються на розподілі навантаження між локальними вузлами системи в середній та кінцевій стадії формування плану виконання запиту. Врахування особливостей паралельної обробки запиту вже в початковій стадії компіляції дозволяє досягти низки переваг. Зокрема, спрощується процедура розробки програмного забезпечення для паралельного оброблення запиту, забезпечується робота у складі гетерогенних СКБД, подальша обробка запиту допускає використання вже відомих методів паралельного обчислення. Слід, що актуальність теми роботи обумовлюється відсутністю методів забезпечення паралельної обробки запитів у стадії компіляції.

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ОБРОБКИ SQL-ЗАПРОСІВ У СКБД

1.1 Архітектури паралельних систем

В основі різних класів паралельних систем лежать структурні відмінності у компонуванні різних функціональних елементів, основними з яких є процесори та оперативна пам’ять. Відповідно до даного підходу виділяють наступні чотири класи мультипроцесорних систем [4, 17]:

- SMP (Symmetric Multiprocessor) – симетричні мультипроцесорні системи (рисунок 1.1 а);
- NUMA (Non-Uniform Memory Architecture) – мультипроцесорні системи з неоднорідним доступом до пам’яті (рисунок 1.1 б);
- MPP (Massively Parallel Processing) – масово-паралельні системи (рисунок 1.1 в);
- кластери – набори повноцінних комп’ютерів, об’єднаних в єдину обчислювальну систему.

SMP архітектури складаються з невеликої кількості процесорів, з’єднаних за допомогою шини або комутатора із загальною оперативною пам’яттю. У цих системах час звернення до фізично єдиної загальної пам’яті однаково всім процесорів. Тому SMP системи належать до класу UMA (Uniform Memory Architecture) [16]. Вузьким місцем SMP систем є загальна шина доступу до оперативної пам’яті. З огляду на це кількість процесорів в SMP системі рідко перевищує 32 [16]. NUMA архітектури характеризуються наявністю фізично розподіленої оперативної пам’яті, що розділяється усіма процесорними модулями як єдиний адресний простір [16, 49]. При цьому окремий процесорний модуль може являти собою SMP систему.

Процесорні модулі NUMA систем з’єднуються мережею. При цьому для забезпечення уніфікованого доступу до пам’яті, що розділяється, і синхронізації змін в кеш-пам’яті процесорів у кожному процесорному модулі

використовується спеціальний пристрій. Системи з NUMA архітектурою можуть включати до 64 процесорних модулів із загальним числом процесорів до 128. Проте неоднорідність доступу до оперативної пам’яті, що розділяється, веде до того, що при вибірці даних з пам’яті чужого модуля ми спостерігатимемо уповільнення від 200% до 750% у різних NUMA системах [49], і ця цифра зростатиме зі збільшенням числа процесорних модулів. Через це ми можемо говорити лише про обмежену масштабованість багатопроцесорних систем з NUMA архітектурою. Слід зазначити, що різниця в часі доступу до локальної та віддаленої кеш-пам’яті в різних системах NUMA становить 400-1500% [49]. Тому продуктивність NUMA систем може істотно знижуватися на додатках, що вимагають частої синхронізації даних. У той же час слід зазначити, що відмінності у швидкості доступу до різних ділянок пам’яті в сучасних архітектурах стають все менш помітними [8].

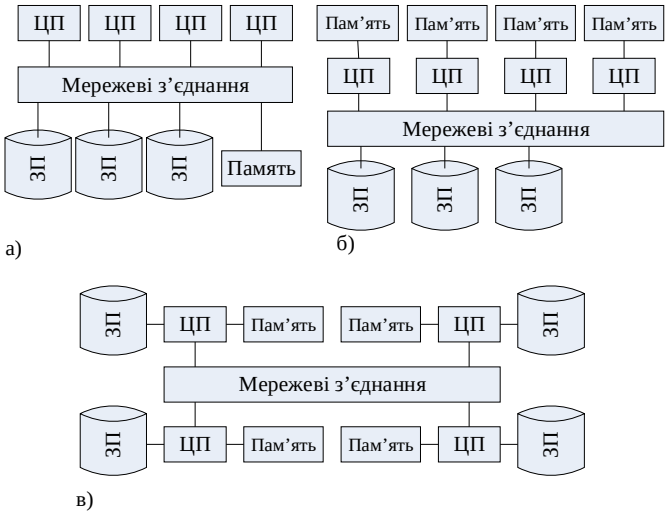

Рисунок 1.1 – Архітектури паралельних систем: а) симетричні мультипроцесорні системи; б) мультипроцесорні системи з неоднорідним доступом до пам’яті; в) масово-паралельні системи

MPP архітектури будуються як сукупність великої кількості однорідних процесорних модулів, з’єднаних високошвидкісною мережею [8].

При цьому кожен модуль має власну оперативну пам’ять та, можливо, власні диски. Подібні системи можуть включати сотні і тисячі процесорних пристроїв. Ключовою проблемою при використанні MPP систем є такий розподіл даних між процесорними модулями, який дозволив би розділити завдання на підзавдання (по одному на кожен модуль), звести до мінімуму обміни між різними підзадачами та забезпечити збалансоване завантаження всіх процесорних модулів. У багатьох випадках це є надзвичайно нетривіальним завданням. Слід зазначити, що сучасні тенденції у розвитку апаратного забезпечення призвели до фактичного злиття MPP та кластерних архітектур [1, 4].

Кластер – це пов’язаний набір повноцінних комп’ютерів, що використовується як єдиний обчислювальний ресурс [1]. У загальному випадку кластерна система може включати в себе дискові масиви та інші спеціалізовані пристрої. Як вузли кластера можуть використовуватися як однопроцесорні системи, так і системи з архітектурою SMP і навіть NUMA. Проте, слід зазначити, що для кластерів характерні самі загальні проблеми з розподілом даних і балансуванням завантаження, що й для MPP систем. Розподіл багатопроцесорних систем на описані класи за структурно-функціональними ознаками є досить умовним у тому сенсі, що реальні багатопроцесорні системи можуть поєднувати у собі риси, притаманні різним класам. Дані класи паралельних систем не враховують особливості, що виявляються при зберіганні та обробці великих обсягів даних, тому практично застосовуються спеціальні класифікації архітектур паралельних баз даних.

1.2 Класифікація архітектур паралельних систем баз даних

Найбільш поширеною системою класифікації паралельних систем баз даних є система, запропонована Майклом Стоунбрейкером. Схематично ця класифікація зображена на рисунок 1.2 (а, б, в). У міру розвитку систем

паралельної обробки баз даних дана класифікація була визнана недостатньою, у зв’язку з появою багатопроцесорних систем, що одночасно поєднують у собі риси кількох архітектур.

Для опису подібних систем запропоновано розширення класифікації Стоунбрейкера шляхом запровадження двох додаткових класів архітектур паралельних машин баз даних (рисунок 1.2 (г, д)). Відповідно до класифікації Стоунбрейкера паралельні системи баз даних можуть бути розділені на наступні п’ять базових класів залежно від способу поділу апаратних ресурсів (спочатку клас-сифікація Стоунбрейкера включала перші три класи систем):

- SE (Shared-Everything) – архітектура з пам’яттю і дисками, що розділяються (рисунок 1.2 а);

- SD (Shared-Disks) - архітектура з дисками, що розділяються (рисунок 1.2 б);

- SN (Shared-Nothing) – архітектура без спільного використання ресурсів (рисунок 1.2 в);

- РЄ (Clustered-Everything) – архітектура з SE кластерами, об’єднаними за принципом SN (рисунок 1.2 г);

- CD (Clustered-Disk) – архітектура із SD кластерами, об’єднаними за принципом SN (рисунок 1.2 д). Кордон SD кластерів на малюнку поширені на загальну сполучну мережу, тому що в них може бути власна сполучна мережа.

SE архітектура представляє системи баз даних, де всі диски безпосередньо доступні всім процесорам з однаковим часом доступу і всі процесори поділяють загальну оперативну пам’ять. Міжпроцесорні комунікації в SE системах здійснюються через загальну оперативну пам’ять. Доступ до дисків SE системах зазвичай здійснюється через загальний буферний пул. У цьому слід зазначити, кожен процесор в SE системі має власну кеш-пам’ять. Існує велика кількість паралельних систем баз даних із SE архітектурою. По суті, всі провідні комерційні СКБД сьогодні мають реалізацію на базі SE архітектури. Як один з перших прикладів переходу з

однопроцесорної системи на SE архітектуру можна навести реалізацію DB2 на IBM3090 з 6 процесорами. Слід зазначити, однак, що переважна більшість комерційних SE систем використовує тільки міжтранзакційний паралелізм (тобто внутрішньотранзакційний паралелізм відсутній). Тим не менш, зараз створено кілька дослідницьких прототипів SE систем, що використовують внутрішньозапитовий паралелізм.

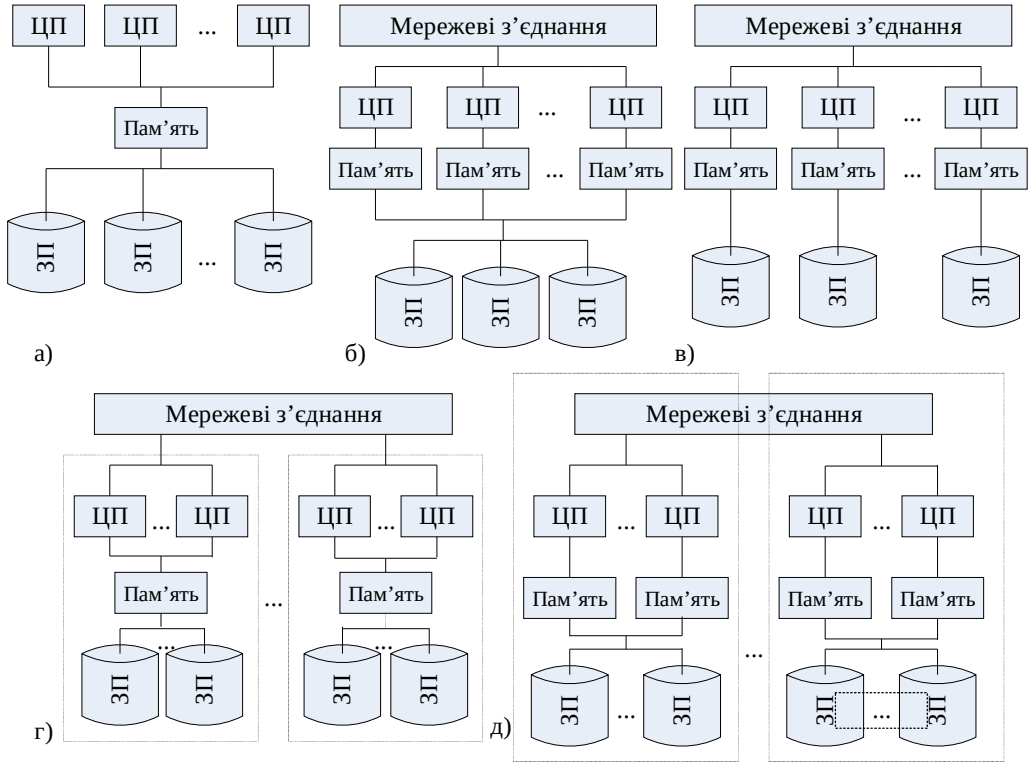

Рисунок 1.2 – Класифікація Стоунбрейкера для паралельних систем: а) архітектура з пам’яттю і дисками, що розділяються; б) архітектура з дисками, що розділяються; в) архітектура без спільного використання ресурсів; г) архітектура із SE кластерами; д) архітектура із SD кластерами

Базовою апаратною платформою для реалізації систем з SE архітектурою зазвичай служить SMP, хоча потенційно SE системи можна будувати на платформах з архітектурою NUMA і навіть MPP з віртуально загальною, фізично розподіленою пам’яттю [1]. SD архітектура представляє системи баз даних, у яких будь-який процесор має доступом до будь-якого

диску, проте кожен процесор має власну оперативну пам’ять.

Процесори в таких системах з’єднані за допомогою деякої високошвидкісної мережі, що дозволяє здійснювати передачу даних. Як приклад паралельної системи баз даних, що працює з SD архітектурою, можна навести Oracle Parallel Server. SN архітектура характеризується наявністю у кожного процесора власної оперативної пам’яті та власного диска. Як і в SD системах, процесорні вузли з’єднані деякою високошвидкісною мережею, що дозволяє організовувати обмін повідомленнями між процесорами.

На даний момент створено велику кількість дослідницьких прототипів і кілька комерційних систем з SN архітектурою, що використовують фрагментний паралелізм. Як приклади комерційних систем з SN архітектурою можна навести Teradata [21], IBM DB2 PE [45]. PЄ та CD системи є прикладами змішаних архітектур. На рисунку 1.2 наведено приклади дворівневих ієрархій. Однак класифікаційний підхід легко може бути поширений на архітектури з трьома та більше рівнями ієрархії. Таким чином, проведений аналіз підходу до класифікації архітектур паралельних систем баз даних показує, що розширена класифікація Стоунбрейкера дозволяє класифікувати всі відомі на сьогодні типи архітектур. З аналізу даних архітектур випливає, що найбільш підходящою для паралельних СКБД архітектурою є SN архітектура [8] та системи, побудовані з використанням змішаних архітектур.

1.3 Масштабованість паралельних систем обробки даних

Важливою властивістю паралельних платформ є можливість їх динамічного нарощування з метою адаптації до збільшення розміру бази даних або зростаючим вимогам продуктивності. Це досягається шляхом поступового додавання до конфігурації системи додаткових процесорів, модулів пам’яті дисків та інших апаратних компонентів. Цей процес

називається масштабуванням системи. При подвоєнні апаратної потужності системи ми можемо очікувати, що її продуктивність також зросте вдвічі, проте на практиці реальне збільшення продуктивності часто виявляється істотно нижче. Наприклад, масштабованість SE систем обмежена 20-30 процесорами. При подальшому нарощуванні SE системи продуктивність зростає дуже слабко або навіть починає відбуватися її падіння. Це пов’язано з тим, що процесори починають все більше і більше простоювати в очікуванні доступу до ресурсів, що розділяються. Відповідно до цього масштабованість будь-якої багатопроцесорної системи визначається ефективністю розпаралелювання.

Існують дві основні якісні характеристики ефективності розпаралелювання: прискорення та розширюваність. Дамо наступне формалізоване визначення зазначених понять. Нехай дано дві різні конфігурації A і B паралельних систем баз даних із заданою архітектурою, що відрізняються кількістю процесорів і асоційованих з ними пристроїв (ми припускаємо, що всі конфігурації припускають пропорційне нарощування модулів пам’яті та дисків). Нехай заданий деякий тест Q. Коефіцієнт прискорення aAB, що отримується при переході від конфігурації A до конфігурації, визначається формулою 1.1:

$$a_{AB} = \frac{d_A t_{QA}}{d_B t_{QB}}, \qquad (1.1)$$

де dA – рівень паралелізму (наприклад – кількість процесорів) конфігурації A;

dB – рівень паралелізму зміни У;

tQA – час, витрачений конфігурацією A виконання тесту Q;

tQB – час, витрачений конфігурацією на виконання тесту Q.

Кажуть, що система демонструє лінійне прискорення, якщо коефіцієнт прискорення залишається рівним одиниці всім конфігурацій цієї системи.

Нехай тепер заданий набір тестів Q1, Q2, ..., кількісно перевершують певний фіксований тест Q в і раз, де і - номер відповідного тесту. Нехай задані зміни віртуальної паралельної машини баз даних A1, A2, ..., що перевершують за ступенем паралелізму деяку мінімальну конфігурацію А в j раз, де j - номер відповідної конфігурації. Тоді коефіцієнт розширюваності ekm, що отримується при переході від конфігурації Ак до конфігурації Am (k < m), задається формулою 1.1:

$$e_{km} = \frac{t_{ok}}{t_{QA}} \ . \qquad (1.2)$$

Кажуть, що система демонструє лінійну розширюваність, якщо коефіцієнт розширюваності залишається рівним одиниці всім конфігурацій цієї системи.

Прискорення дозволяє визначити ефективність нарощування системи на сумісних задачах. Розширюваність дозволяє виміряти ефективність нарощування системи великих завдань. Кажуть, що паралельна система добре масштабується, якщо вона демонструє прискорення та розширюваність, близькі до лінійних. Основним фактором, що перешкоджає хорошій масштабованості системи, є перешкоди, що виникають при конкурентному доступі процесорів до ресурсів, що розділяються, і необхідність збереження порядку ряду виконуваних операцій.

1.4 Методи пошуку субоптимальних алгоритмів використання запиту

Процес компіляції запиту, як показано рисунку 1.3, можна як трьох основних етапів:

- синтаксичний аналіз – конструювання дерева аналізу, що описує запит та його структуру;

- перезапис запиту – перетворення дерева розбору на початковий

огический план запиту, який зазвичай відображається як алгебраїчного представлення запиту, а навіщо перетворюється на еквівалентний ефективніший план;

- генерація фізичного плану – перетворення абстрактного логічного плану запиту, отриманого на етапі 2, у фізичний план запиту за допомогою вибору алгоритмів, що реалізують кожен оператор логічного плану, та завдання порядку виконання цих операторів.

Фізичний план, подібно до результату синтаксичного аналізу та логічного плану, представляється у формі дерева виразів. Крім того, фізичний план містить опис деталей реалізації, які, наприклад, регламентують способи доступу до відносин, згаданим у запиті, або визначають умови та черговість сортування вмісту відносин.

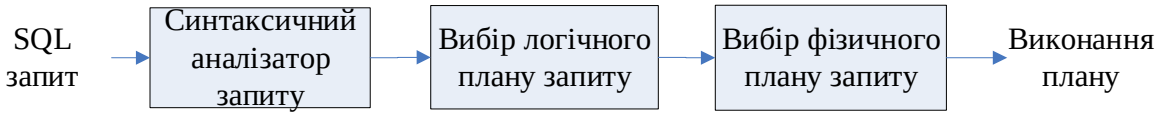

Рисунок 1.3 – Схема компіляції SQL-запиту

Різні фізичні плани однієї й тієї ж запиту виконуються за час. Природне завдання, яке випливає з цього факту – пошук плану запиту, який виконується за мінімальний час. Вичерпний підхід до цієї проблеми міг би полягати в аналізі всіх можливих планів виконання запиту. Однак такий підхід не прийнятний, тому що навіть окремі підзавдання знаходження оптимального плану виконання запиту оцінюється як NP – складні. На рисунку 1.4 представлені існуючі методи пошуку запиту у просторі можливих планів виконання запиту.

Евристичний вибір плану виконання запиту полягає у використанні фіксованого набору правил, згідно з якими, за синтаксисом та семантикою запиту будується план виконання. Спосіб гілок і кордонів початку роботи

передбачає використання певних евристичних правил для пошуку деякого “хорошого” плану виконання. Нехай ефективність виконання знайденого плану дорівнює С. Якщо в міру розгляду планів виразів з’ясовується, що будь-який приватний план має меншу ефективність, то такий план виключається з числа потенційних кандидатів на формування повного плану, інакше такий повний план приймається за поточний оптимальний план виконання.

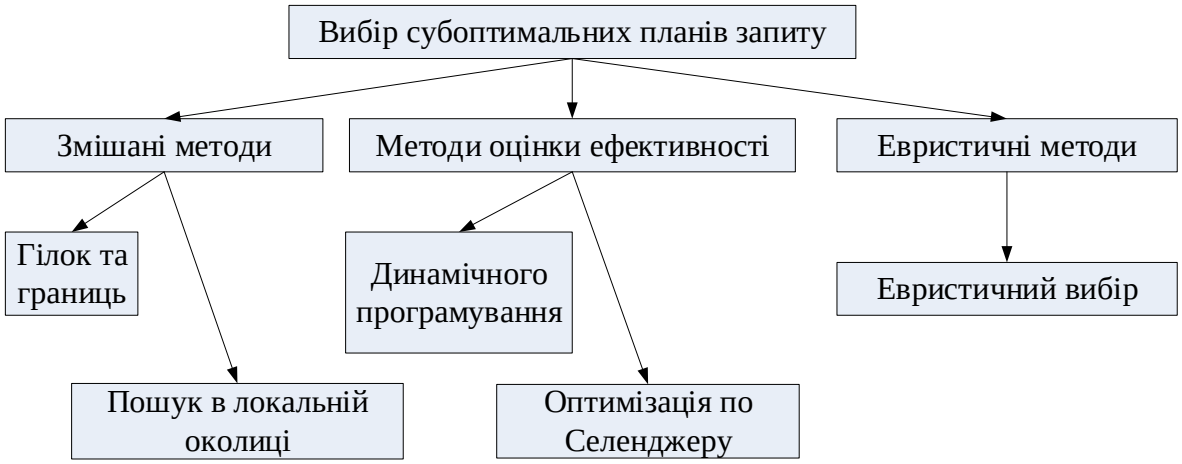

Рисунок 1.4 – Методи вибору субоптимального запиту

Важлива перевага методу полягає в можливості швидкого і контрольованого завершення пошуку. Наприклад, якщо поточне значення вже нас влаштовує, процедуру доречно завершити, навіть якщо існують і кращі плани виконання. Метод пошуку локальної околиці спочатку з допомогою будь-якої евристики вибирає певний фізичний план. Потім з метою відшукання "сусіднього" плану, що володіє меншим очікуваним часом виконання, в поточний план вносяться локальні зміни. В якості шуканого приймається такий план виконання запиту, який не може бути збільшений за рахунок будь-яких локальних змін.

Метод динамічного програмування, використовуваний оптимізатором запитів, передбачає використання оптимальних планів кожного окремого подвыражения. Центральна ідея, що лежить в основі методу, полягає у

заповненні таблиці оцінок зі збереженням мінімального обсягу даних, необхідних для ухвалення рішення [3]. Оптимізація по Селенджеру є вдосконаленою версією алгоритму динамічного програмування, здатний враховувати кожному за висловлювання як плани з найменшими значеннями вартості, а й деякі інші, менш оптимальні плани, здатні знизити вартість планів виразів вищого рівня. Оптимальна підструктура означає, що оптимальне рішення підзадач меншого розміру може бути використане як метод вирішення вихідної задачі.

1.5 Методи обробки даних реляційної структури

Розглянемо формальну систему побудови запитів засновану на реляційній алгебрі, використовувану за підтримки систем баз даних. Опишемо певні припущення, у яких використовується. Це доводиться робити у зв'язку з тим, що у різних джерелах відсутня однаковість в записі алгебри операторів реляційної алгебри. Здебільшого, ми керуватимемося позначеннями. У класичній реляційній алгебрі всі оператори та результати обчислення виразів є безліччю. Як операнди і результати операцій ми маємо на увазі мультимножини, іншими словами - безлічі, що допускають включення в себе кортежів дублікатів. Як значення атрибутів кортежів можна використовувати значення спеціального виду null – "невизначена" величина. Оператори реляційної алгебри. Оператор проекції застосовується до відношення з метою отримання нового відношення, що містить лише деякі з атрибутів. Значенням виразу служить відношення $\pi_{A_1,A_2,...,A_n}(R)$, що містить явно перераховані атрибутит $A_{1,}A_{2,...,}A_n$ відношення R. Схема підсумкового відношення охоплює безліч атрибутів $\{A_{1,}A_{2,...,}A_n\}$, що відображаються у вказаному порядку. Як атрибути можуть використовуватися різні вирази виду $E \to A_i$, де E - вираз, що включає найменування атрибутів відношення R, константи та арифметичні та рядкові

оператори, а A_i- нове ім’я атрибута. Оператор вибору $\sigma_c(R)$, що застосовується до відношення, дає в результаті нове відношення, що містить підмножину кортежів R, що задовольняють певною умовою, яка накладається на атрибути відношення R. Схеми підсумкового та вихідного відношення збігаються, і порядок проходження атрибутів у них зберігається.

Декартовий добуток відносин і є набір кортежів R,S, що є безліч пар кортежів таких, що першим елементом пари є будь-який елемент відносини R, а другим – будь-який елемент відносини S.

Оператор позначається як R*S. Для перейменування відносини R використовується оператор $\rho_s(A_1, A_2, .., A_n)(R)$. Підсумкове ставлення має ті ж кортежі, що й R, але отримує ім’я S, яке атрибути – імена $A_{1,} A_{2,...,} A_n$. Оператор видалення дублікатів перетворює відношення мультимножина у відношення безліч шляхом видалення кортежів, що дублюються.

Оператор сортування – $\tau_L(R)$ повертає список кортежів, відсортований у порядку зростання значень. Зазначимо, що результат оператора не є мультимножиною, тому цей оператор може використовуватися тільки як "зовнішній". Якщо після оператора застосовується інший оператор, відношення, що служить результатом обчислення , інтерпретується як безліч або безліч кортежів, і який би там не було порядок їх прямування до уваги не приймається.

1.6 Постановка завдання

Проведений аналіз існуючих паралельних архітектур, здатних підтримувати функціонування СКБД, показав перспективи розробки паралельних СКБД, орієнтованих на роботу в різних варіантах масово-паралельних системах MPP. Досліджено завдання, які необхідно вирішувати при паралельному виконанні запитів та існуючі методи їх вирішення.

Розглянуті відомі алгоритми з’єднання відносин у паралельних СКБД у

певних випадках вимагають інтенсивної взаємодії між обчислювальними вузлами і не дозволяють взаємодіяти з іншими джерелами даних, тому що їх реалізації знаходиться в тій частині системи, яка не має стандартного (зазвичай взагалі ніякого) інтерфейсу. Т.ч. існуючі СКБД недостатньо використовують можливості паралельного виконання запитів у гетерогенних реляційних базах даних.

У результаті проведеного аналізу, мета роботи полягає у дослідженні методів паралельного виконання SQL - запитів у великомасштабних системах для підвищення продуктивності розподілених СКБД та зменшення часу виконання запиту. Отже, для досягнення поставленої мети потрібно виконати докладний аналіз ефективності існуючих методів розпаралелювання запитів у гетерогенних середовищах. Дослідити алгоритми та програмні засоби для паралельного виконання SQL – запитів, які забезпечують сумісність із існуючими СКБД. Отримати апріорні оцінки часу виконання запитів. Спираючись на результати отриманих досліджень, виконати експериментальну оцінку методів та алгоритмів використаних для розпаралелювання запитів у паралельних системах баз даних.

2 ПРИНЦИПИ ПОБУДУВАННЯ СИСТЕМ ДЛЯ ПАРАЛЕЛЬНОЇ ОБРОБКИ ЗАПИТІВ

2.1 Дослідження архітектур систем паралельного виконання запитів з використанням надмірного представлення інформації

Як випливає з роботи [7] єдиним рішенням, що дозволяє обробляти запити до надвеликих баз даних, є тиражування даних з метою їхньої паралельної обробки.

Надвеликі бази даних характеризуються тим, що обсяг даних, що зберігаються в них, перевищує обсяг оперативної пам’яті, доступної для системи на багато порядків. В результаті чого, для ефективного завантаження обчислювальних потужностей системи потрібно забезпечити високу пропускну здатність дискової підсистеми. Для завантаження обчислювальних потужностей системи є ефективним спосіб поділу оброблюваних даних на основі діапазонів оброблюваних значень.

Методи поділу, використовувані для зберігання даних, на основі діапазонів не дозволяють проводити цю операцію (поділу) динамічно (для вирішення цього завдання довелося б зберігати копії даних розділені по всіх можливих діапазонах розбиття). Крім того, можливий дисбаланс навантаження на вузли системи, що не допускає програмне регулювання. Метод поділу даних на основі хешування не дозволяє ефективно працювати з даними, що розбиваються на діапазони. Розміщення даних на основі хешування також здійснюється з урахуванням апріорно заданого набору атрибутів. Застосування цього методу породжує дисбаланс навантаження під час проведення операції з’єднання відносин. Здебільшого ці методи орієнтовані застосування у системах, заснованих на принципах симетричної мультипроцесорної обробки (SMP системи).

Відомо, що системи, засновані на масовій мультипроцесорній обробці

(MPP), дозволяють досягати великих обчислювальних можливостей при менших витратах, і забезпечують кращу масштабованість щодо SMP систем. Основним недоліком MPP систем [15] є невисока (порівняно з SMP системами) швидкість взаємодії між окремими обчислювальними вузлами. Для вирішення цієї проблеми пропонується забезпечити кожен обчислювальний вузол MPP системи власною копією даних. З урахуванням цього факту, можна реалізувати операції з паралельної обробки даних, мінімізуючи взаємодію між вузлами системи.

Слід зазначити, що при даному підході як вузли MPP системи можуть виступати окремі SMP системи. Така архітектура називається змішаною або гібридною, і дозволяє досягати більш високої продуктивності, за рахунок додаткового застосування методів оптимізації запитів, орієнтованих на SMP системи, які на сьогоднішній момент досить розвинені.

Опишемо систему, засновану на MPP архітектурі, яка дозволяє здійснювати паралельну обробку SQL-запитів [23]. Нехай система складається з N вузлів, у яких встановлено SQL сервера, причому дані SQL сервера нічого не винні бути однорідними, тобто. вони не обов’язково повинні бути копіями того самого програмного продукту. У такому разі, для звернення до подібної системи необхідно використовувати підмножину мови SQL, загальне для всіх SQL серверів системи. Зазначимо, що у використовуваних СКБД є оператори мови SQL, які відрізняються лише синтаксичною формою, то з допомогою застосування тривіальних перетворень стає можливим використання синтаксичних конструкцій мови SQL, які є спільними всім SQL серверів системи.

До прикладів таких конструкцій можна віднести оператор вилучення перших N рядків результату запиту. У той же час існують конструкції, які не допускають тривіального перетворення з одного діалекту SQL в інший. Наприклад - використання ієрархічних запитів, якщо хоча б один із SQL серверів не підтримує цієї можливості. Звичайно, якщо вимагати однорідності SQL серверів системи, то такі обмеження на використання

операторів у запитах не виникають. Припустимо наявність бази даних, створеної з урахуванням MPP системи. Також припустимо, що цікавий для нас набір даних реплікований на N серверів розподіленої системи. Під узгодженістю набору даних на двох вузлах ми розумітимемо ідентичність результату будь-якого запиту на читання даних, виконаного як на першому, так і на другому вузлі.

Для реалізації паралельної обробки даних, аналізований набір може бути узгоджений у час роботи системи, будь-яких комбінацій вузлів. Зокрема, це означає, що на момент старту роботи системи набори даних на її вузлах мають бути синхронізовані. У цій системі можлива реалізація кількох алгоритмів поповнення даних.

У найпростішому випадку, оновлення даних може відбуватися шляхом призупинення роботи системи на певний проміжок часу, протягом якого проводиться додавання або зміна вже існуючого набору даних. При цьому найбільш просто реалізується поповнення інформації, характерне для класичних сховищ даних (data warehouse), в яких відсутня необхідність оновлення раніше накопичених даних. Можлива реалізація поповнення інформації без припинення функціонування системи. Тоді оновлення даних у всіх вузлах розподіленої системи має здійснюватися в рамках єдиної розподіленої транзакції з тією метою, щоб внесені зміни в різних вузлах системи ставали доступні користувачам, які роблять читання, в синхронному режимі. Така поведінка системи може бути досягнуто при використанні механізму блокувань, реалізованого в СКБД, або на підставі концепції тимчасових міток, коли при обробці запиту враховується час занесення запису в базу, і пізніше визначеного часу ігноруються.

Можна відмовитися від ідентичності всіх копій на будь-який момент часу, однак, у цьому випадку буде можливість для роботи з базою в цілому тільки в режимі, еквівалентному брудному читанню (dirty read), що може виявитися достатнім для роботи низки додатків. І тут можливе виникнення проблеми неузгодженої обробки, відомої як проблеми не повторюваності

читання [8]. Відзначимо, що виникнення проблеми подібного роду при виконанні паралельних запитів не залежить від рівня ізоляції транзакцій, що модифікують, на локальних вузлах системи. Для забезпечення вищого рівня ізоляції користувацьких транзакцій (узгодженості копій даних) ця вимога є необхідною.

У системі можлива реалізація змішаної концепції обробки даних, як OLAP (Online Analytical Processing), так і OLTP (Online Transaction Processing). Для цього доцільно виділити один "потужний" сервер для підтримки OLTP частини бази даних, інформація якого не піддаватиметься тиражуванню, а решту системи, що забезпечує функціонування OLAP компоненти реалізувати за запропонованою вище схемою. Тоді повна інформація для бази даних буде складатися з однієї копії даних з OLAP частини системи та даних OLTP сервера. Отримання даних як з OLAP частини системи, і з OLTP частини можна виконувати засобами СКБД з урахуванням концепції поділу даних.

Точку поділу даних можна вибрати, наприклад, як дату закриття періоду, за умови, що дані після закриття відповідного періоду не підлягають редагування. Узгоджені зміни довільних даних у системі, загалом, представляє складне (з погляду витрат ресурсів) завдання у кожному з запропонованих варіантів реалізації системи, і реалізація цієї можливості виглядає недоцільною.

Слід розуміти, що запропонована схема побудови системи дозволяє прискорювати виконання SQL-запитів на читання даних за рахунок зниження швидкості виконання запитів на вставку/зміну/видалення даних (далі – модифікацію). Відомо, що необхідність оновлення реплікованих даних у рамках транзакції вимагає більшої кількості витрат і вносить свої обмеження. Останній із запропонованих варіантів реалізації системи представляє компромісний варіант між підвищенням накладних витрат на модифікацію даних та збільшенням швидкості обробки даних. Для описаної вище розподіленої системи пропонується наступний метод паралельного

виконання SQL-запитів. Запит користувача, що спочатку надійшов на один з вузлів системи, готується для паралельного виконання. Можливість такого паралельного виконання забезпечується за рахунок перетворення вихідного SQL-запиту до набору запитів, результат виконання яких еквівалентний вихідному [5] (після об’єднання результатів виконання нових запитів, можливого додаткового обчислення агрегатних функцій і групування, а також переупорядкування результату).

Як буде показано пізніше, набір запитів, отриманий в результаті перетворення, можна виконувати будь-якому вузлі системи незалежно один від одного. Набір правил перетворення запитів заздалегідь фіксований, і допустимість застосування тієї чи іншої правила перетворення визначається синтаксисом вихідного запиту та статистичною інформацією, доступною для атрибутів відносин, що використовуються у запиті. Такий підхід дозволяє динамічно вибирати алгоритм паралельного виконання запиту залежно від змісту таблиць бази даних.

Ключовим моментом при реалізації запропонованого підходу є додавання проміжної ланки до послідовності дій, що відповідають виконання запиту. Фактично, при практичної реалізації запропонованого рішення, можливе побудова системи, заснованої на трирівневої архітектурі (рисунок 2.1). При цьому з точки зору клієнтської програми процес обробки запиту залишається незмінним незалежно від реальної кількості серверів, виділених для обробки запиту. Усі функції з координації роботи окремих серверів баз даних перебирає програмне забезпечення проміжного рівня.

Апаратною платформою для роботи проміжної ланки можуть виступати як спеціально виділені для цієї мети вузли системи, так і вузли, які безпосередньо забезпечують виконання користувальницьких запитів до СКБД.

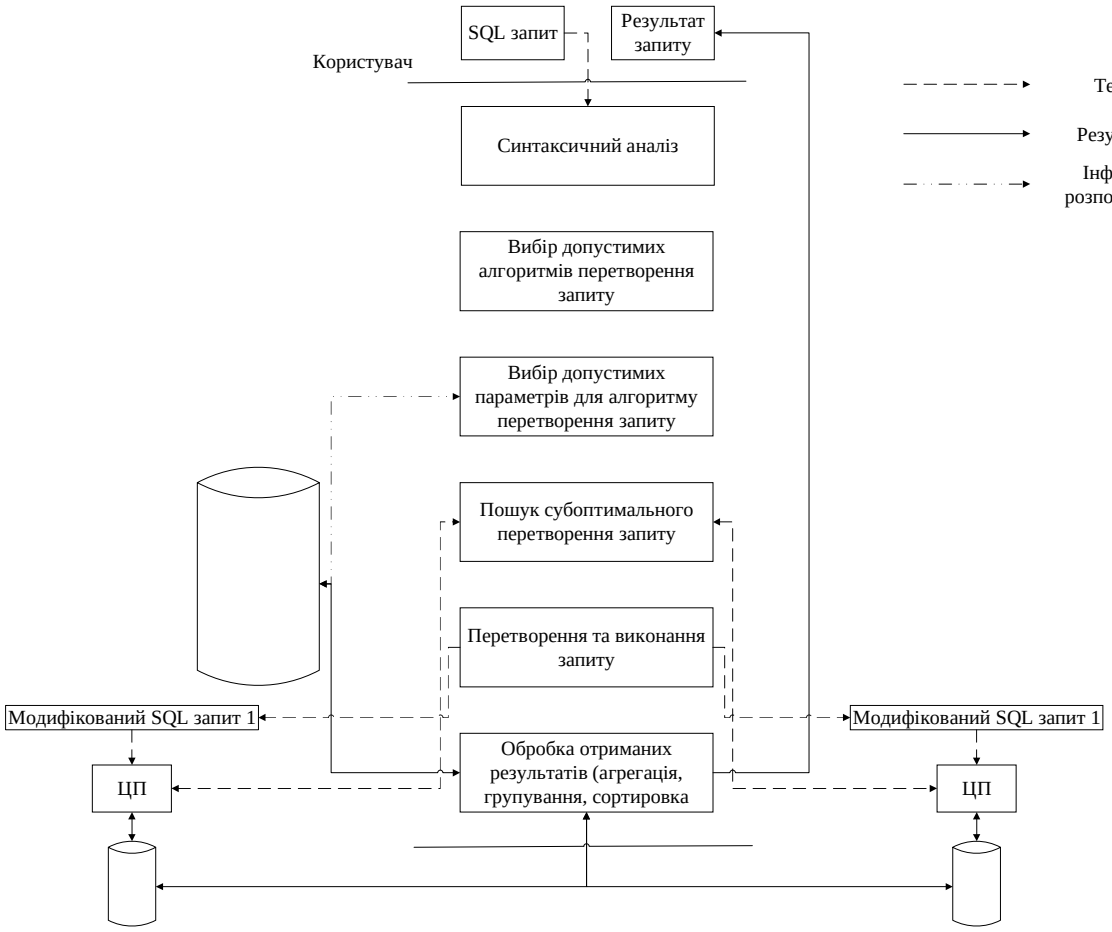

Рисунок 2.1 – Архітектура системи для паралельного використання запитів

Підсумовуючи вище сказане, сформулюємо переваги, які досягаються при реалізації цього підходу. Вони полягають у наступному:

- забезпечується робота з неоднорідними СКБД;

- реалізується можливість досягнення високої продуктивності під час обчислення результату запита;

- досягається відносна простота реалізації системи з паралельною обробкою запитів. Фактично, у системі застосовується подвійний процес оптимізації – для паралельного виконання запиту між вузлами системи та процесу локального виконання запиту кожному з вузлів;

- є можливість реалізації запропонованих принципів паралельної обробки в системах з класичними методами розпаралелювання (операцій сканування, з’єднання та сортування);

- динамічна зміна кількості вузлів, здатних обробити запит у процесі роботи системи;

- динамічна зміна алгоритмів перетворення запиту. Оперуючи статистичною інформацією за даними, що зберігаються в системі, є можливість динамічної зміни алгоритмів перетворення запиту. Таким чином, ми не відходимо від декларативних принципів, закладених в основу мови SQL, яка визначає, що необхідно зробити, а як це потрібно зробити – вирішує програма, що обробляє SQL-запит.

2.2 Дослідження та розробка методу декомпозиції запиту

Початкова стадія компіляції запиту складається з чотирьох етапів: запит (текстове уявлення) – синтаксичний аналізатор – препроцесор – генератор логічного плану запиту – переписувач логічного плану запиту. Доповнимо цю схему двома етапами – синтаксичний аналізатор паралельного запиту та препроцесор паралельного запиту, які передуватимуть чотирьом класичним етапам компіляції. Синтаксичний аналізатор паралельного запиту, власне, виконує ту ж функцію, як і синтаксичний аналізатор у класичній схемі компіляції – перетворює текст вихідного запиту мовою SQL в дерево розбору. Деталі реалізації, способи, позначення та особливості процесу синтаксичного аналізу наводяться в [2]. Препроцесор паралельного запиту, на відміну від класичної схеми (де він призначений для заміни позначень деревами розбору та семантичного контролю), у запропонованій новій схемі модифікує дерево запиту з метою виділення під дерев запиту придатних для паралельного виконання. В результаті роботи паралельного препроцесора формується набір нових запитів, обробка яких, надалі, будується за класичною схемою. Перетворення дерев розбору запиту проводяться препроцесором з використанням наперед відомого набору правил, з метою отримання еквівалентного запиту.

Як приклад розглянемо запит, який повертає кількість відвантаженого товару за період відвантаження (лістинг 2.1):

Лістинг 2.1 – Запит для знаходження кількості відвантаженого товару

```
SELECT sum (QUANTITY)
FROM LINEITEM
WHERE SHIPDATE>=’ 20211711’ and SHIPDATE<=’ 20213011’
```

У деяких випадках після проведення перетворень можуть бути потрібні додаткові операції над наборами відносин, що повертаються запитами. Дерево розбору, яке відповідає даному запиту, буде виглядати наступним чином (рисунок 2.2).

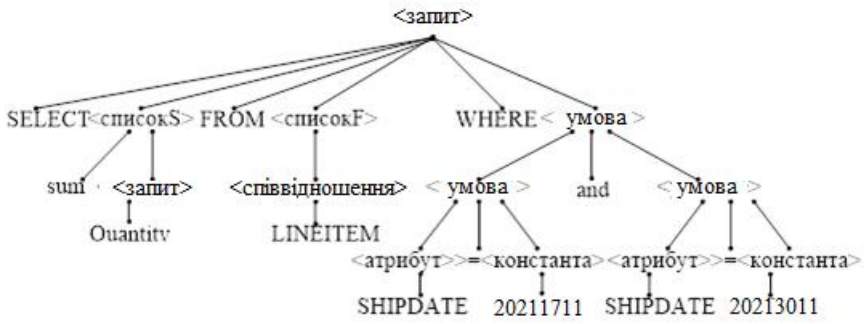

Рисунок 2.2 – Дерево розбору SQL-запиту

Під еквівалентністю двох запитів тут і далі ми розумітимемо такі запити, в результаті виконання яких формуються однакові у всіх атрибутах кортежів відносини з точністю до порядку слідування кортежів, якщо не задана інструкція сортування, і з урахуванням порядку слідування в іншому випадку.

Слід зазначити, що в загальному випадку подальше обчислення запиту згідно з деревом розбору можна проводити тільки при отриманні результатів всіх підзапитів і виразів, що нижчі в ієрархії.

Виходячи з вищевикладеного зауваження, можна сформулювати цілі,

які мають досягатися за допомогою еквівалентних перетворень запитів [22]:

- правило перетворення з вихідного запиту має формувати новий запит, що містить апріорно задану кількість некорельованих підзапит;

- отримані підзапити повинні володіти приблизно рівною вартістю виконання, так як подальше обчислення запиту можливе тільки після обчислення відповідних підзапитів, і у разі істотного перевищення часу виконання одного підзапиту над іншими, інші вузли системи (не зайняті обчисленням підзапиту) можуть простоїти. Таким чином, перетворення запиту має контролювати баланс навантаження між вузлами системи шляхом відповідного формування підзапитів;

- на верхніх рівнях дерева аналізу запиту перетворення має залишати найбільш "дешеві" операції.

Під терміном "дешеві" тут маються на увазі операції, для реалізації яких не потрібно обробки великої кількості записів, так як, наприклад, при їх обчисленні вже буде неможливо скористатися інформацією, що міститься в індексах;

- перетворення, по можливості, не повинно збільшувати обсяг відносин, що виходять при обчисленні підзапитів для того, щоб виключити передачу великих обсягів даних між вузлами системи. Значний обсяг передачі може серйозно уповільнити виконання запиту і зменшити виграш від паралельної обробки. Очевидно, що в загальному випадку існує більше одного правила для еквівалентного перетворення запиту, які можуть його готувати для паралельного виконання, і при виборі різних правил можна досягати різної підсумкової швидкості виконання запиту.

Таким чином, препроцесор паралельного запиту повинен, по заданому дереву запиту, формувати набір дерев для еквівалентних запитів. У цих деревах запиту має міститися необхідна кількість піддерев, виконання яких можливо проводити паралельно. З цих дерев, на основі оцінки ефективності обчислення [8], вибирається дерево, згідно з яким виконання запиту буде виконано з найменшими накладними витратами.

2.3 Дослідження та модифікація методу паралельного виконання запитів

Реалізація запиту наведено в лістингу 2.2.

Лістинг 2.2 – Реалізація SQL-запиту

```
SELECT * FROM Al JOIN A2 ON Al.B = A2.C
```

При цьому припустимо, що значення B і C упорядковані – це не має значення для реалізації даного алгоритму, але знадобиться нам у подальших міркуваннях. Відповідно до методу та термінології дослідження можливості паралельного виконання програм, пропонованих у [4], необхідно приведення програми до лінійного класу. Наведена вище реалізація такого класу не відноситься через наявність обчислення індексів k для масивів RB і RC. Зауважимо, однак, що нам не важливий порядок проходження індексів у масивах RB і RC через належність результату обчислень до мультимножин.

Лістинг 2.3 – Алгоритм програми

```
k=1
FOR i=1,n
   FOR j=1,m
   If (A1B(i)<>A2C(j))
   RB(k)=A1B(i)
   RC(k)=A2C(j)
   K=k+1
  END
END
FOR 1=1,k
  CALL OUTPAT(RB(1),RC(1))
END
```

Тому ми можемо помітити рядки 5-7 з прикладу 2.3 підпрограмою виведення значень A1B(i) та A2C(i) безпосередньо на якийсь зовнішній пристрій, минаючи фазу проміжного збереження результату, яку, у свою

чергу, згідно з рекомендаціями [4] слід замінити на наступні вираз для приведення програми до лінійного вигляду, як показано на прикладі 2.5.

Лістинг 2.4 – Лінійний вигляд програми

```
FOR i=1,n
  FOR j=1,m
   If(A1B(i)<>A2C(j))
   A1B(i)=A1B(i)
   A2C(j)=A2C(j)
  END
END
```

Ця програма вже може бути віднесена до лінійного класу. Вона визначає дві простих програми, кожна з яких виходить винятком із вихідного п’ятого або четвертого рядка відповідно. Розглянемо простір ітерацій циклу FOR. Всі його точки можна розбити на множини, що не перетинаються, якщо віднести до кожної множини ті з них, які відповідають одній і тій же ітерації циклу. Позначимо ці множини через M1, ..., MP. Кажуть, що цикл має тип ParFOR за графом G, якщо множини M1,…..,Mp паралельні цьому графу при будь-яких допустимих, але фіксованих значеннях зовнішніх змінних.

У тілі циклу типу ParFOR не існує жодної пари операцій, що відносяться до різних ітерацій, в яких та сама змінна або перераховується або перераховується і використовується. Ітерацію такого циклу можна виконувати в будь-якому порядку на комп’ютерах будь-якої архітектури. При цьому гарантується правильність одержуваних результатів. З [4] відомо достатню умову для визначення належності циклу DO до типу ParFOR: нехай граф G заданий функціями виду . Для того, щоб цикл мав тип ParDO за графом G, достатньо, щоб граф G був порожнім, або для всіх покриваючих функцій перша рівність виразу $x = Jy + \varphi$ мала вигляд $x_1 = y_1$.

З огляду на вигляд покриває функції Ф і наведене вище твердження, отримуємо, що зовнішній цикл має тип ParFOR. Тоді вибравши N-1 точку, яка відповідає умовам:

$$\forall(k,1\in[1...N],k\neq l)\left[[A_{k-1},A_k]\cap[A_{l-1},A_l]=\varnothing\right];$$
$$\forall(k,1\in[0...N],k\neq l)[A_k\neq A_l];\qquad(2.1)$$
$$A_0=1;\ A_N=n$$

Можна перетворити наведену вище програму до виду, в якому буде N циклів типу ParFOR з параметром циклу, що змінюється в межах $A_{i-1}\le i\le A_i$. Для цього необхідно згрупувати відповідні ітерації зовнішнього циклу. Як тільки було показано, такі цикли можуть виконуватися паралельно.

Зауважимо, що кожен з N циклів ParFOR, що представляє собою групу ітерацій зовнішнього циклу FOR, реалізує запит, представлений на лістингу 2.5.

Лістинг 2.5 – Лінійний вигляд програми

```
SELECT * FROM Al JOIN A2 ON Al.B = A2.C
and (A1.B >= A1B(A_{i-1}) and A1.B <= A1B(A_i))
```

Отже, запити цього типу допускають паралельне виконання.

2.4 Апріорні оцінки часу виконання запитів

Отримаємо тимчасові оцінки виконання запитів, заснованих на принципах поділу по діапазонам. При їх виведенні ми будемо дотримуватися спрощеного припущення, що дані зберігаються і зчитуються з дискових накопичувачів послідовно, по одному блоку в кожний момент часу. Визначимо чотири сімейства параметрів, назвемо їх B і T, V і L - застосовувані для опису відносин-аргументів операцій, що використовуються для отримання даних:

- кількість блоків, необхідне зберігання всіх кортежів відношення R ми позначимо як B(R). Зазвичай передбачається, що дані відносини R компактно згруповані в (або приблизно в) блоків.

- кількість кортежів R позначимо як T(R). Тоді кількість кортежів, які частішають у межах одного блоку, виражається як T(R)/ B(R).

- позначимо за $V(R,(a_1,...,a_n))$ – кількість різних значень у межах певних стовпців $(a_1,...,a_n)$ відношення R, інакше кажучи $V(R,(a_1,...,a_n)) = \gamma_{count(*)\to cnt}(\delta(\pi_{a_1,...,a_n}(R)))$;

- позначимо за $L(R,n)$ – різницю в часі витрачаному на передачу дискових блоків відношення R від одного обчислювального пристрою іншому та часом читання з диска одного блоку. Як загальний випадок розглянемо запит з операцією групування, представлений з лістингу 2.6.

Лістинг 2.6 – Запит з операцією угруповування

```
SELECT R.A, sum(R.B) as B FROM R WHERE R.C>=C1 and R.C<=C2 GROUP
BY R.A
```

Проведемо оцінку витрат на виконання даного запиту у разі його виконання лише на одному вузлі системи та у разі паралельного виконання на вузлах. При аналізі ми враховуватимемо лише час дискових операцій вводу-виводу та операцій з мережевої передачі даних як істотно більш витратних, порівняно з обробкою кортежів в оперативній пам’яті.

Спочатку проведемо аналіз спрощених запитів, розглянувши запит, показаний з лістингу 2.6. Якщо вміст відношення згрупований по блоках компактним чином, для його завантаження в оперативну пам’ять оператору табличного сканування знадобиться виконати приблизно операцій дискового введення-виводу, тоді загальний час виконання запиту можна оцінити так:

$$S_1 = B(R) + L(R, B(R)) \qquad (2.2)$$

Для випадку виконання запиту з використанням передбачуваних методів отримуємо, підсумовуючи витрати по вузлам-виконавцям і координуючому вузлу:

$$S_1^{'} = \frac{B(R)}{N} + \sum_{i=1}^{N} \frac{L_i(R, B(R))}{N} + L(R, B(R)) + \Omega = \frac{B(R)}{N} + 2L(R, B(R)) + \Omega, \quad (2.3)$$

де Ω позначено час, необхідне підготовки запиту до паралельного виконане виражене в еквівалентному кількості дискових блоків, які обробляються за час. Ми отримуємо лінійне прискорення операції табличного сканування, але більш ніж двократне уповільнення операції передачі. Слід очікувати, що операція сканування таблиці значно більш витратна, ніж операція передачі даних. При значеннях очікується істотного прискорення під час виконання операції. Оскільки запит не виробляє жодної обробки та агрегації даних, ми не можемо отримувати суттєвий виграш зі збільшенням кількості вузлів, оскільки необхідно буде передавати по мережі подвоєний обсяг даних. Можна запропонувати способи дворазового скорочення обсягу мережного обміну, але ми не будемо на цьому зупинятись, оскільки запити такого роду не є типовими для аналітичних систем.

Якщо розподіл вмісту відносин по блоках не можна вважати компактним, кількість необхідних дискових операцій, взагалі кажучи, може бути значно більшою. У цьому випадку, для виконання табличного сканування, можливо, доведеться звертатися до такої кількості блоків, яка дорівнює кількості кортежів, тобто. Витрати виконання операцій дискового вводу-вивода будуть оцінюватися як $T(R)$ (а чи не як використовувалося вище – $B(R)$. Зазначимо, що у разі виграш від паралельної обробки запитів збільшується.

Всі відомі ефективні алгоритми для виконання цього запиту передбачають неявне сортування відношення за значеннями атрибута. Відповідно до [8], якщо відношення зберігається компактно і його

впорядкування пов’язане з використанням злиттів, то системі доведеться зробити 3B дискові операції. Тоді ми можемо отримати наступну оцінку швидкості виконання запиту:

$$S_2 = 3B(R) + L(R, V(R,B) \cdot \frac{B(R)}{T(R)}). \quad (2.4)$$

У разі паралельного виконання отримаємо:

$$S_2^{‘} = \frac{3B(R)}{N} + 2L(R, V(R,B) \cdot \frac{B(R)}{T(R)}) + \Omega. \quad (2.5)$$

Аналізуючи час виконання другого запиту можна зробити висновок, що прискорення, яке буде досягатися у разі його паралельного виконання, виявиться дещо більшим, за рахунок множника 3 у першого доданку, і того факту, що

$$L(R, V(R,B) \cdot \frac{B(R)}{T(R)}) < L(R, B(R)). \quad (2.6)$$

Розподіл різних значень атрибуту, згідно з рядом досліджень, у загальному випадку підпорядковується закону розподілу Зіпфа [8], відповідно до якого, часто зустрічаемого за ступенем популярності значення пропорційна величині $1/\sqrt{i}$. Оцінимо величину $V(R,B)$, використовуючи це припущення про характер розподілу значень атрибутів. Позначимо за m – кількість значень B, що найбільш часто зустрічаються, а за $n = B(R)$. Тоді, розглядаючи таку систему:

$$\begin{cases} \dfrac{m}{\sqrt{x}} = 1 \\ \sum\limits_{i=1}^{x} \dfrac{m}{\sqrt{i}} = n \end{cases} \tag{2.7}$$

Перетворюючи вираз 2.7 отримуємо вираз 2.8:

$$\begin{cases} m = \sqrt{x} \\ \sqrt{x} \cdot \sum\limits_{i=1}^{x} \dfrac{1}{\sqrt{i}} = n \end{cases} \tag{2.8}$$

Оцінимо друге рівняння системи за допомогою інтегральних оцінок зверху і знизу (відповідно):

$$\sqrt{x} \int_{1}^{x} (i)^{-1/2} \, di = n \tag{2.9}$$

$$\sqrt{x} \int_{0}^{x} (i)^{-1/2} \, di = n \tag{2.10}$$

Тоді з (2.9) та (2.10) справедливо вважати, що

$$V(R, B) = \frac{B(R) + \sqrt{B(R)}}{2}. \tag{2.11}$$

Тепер є інформація щодо аналізу первинного запиту з даного розділу. Припустимо, що з полю C побудований індекс. Розглянемо окремо випадок кластерного та не кластерного індексів. При використанні не кластерного індексу отримуємо наступні оцінки (позначаючи $Z = \sigma_{C>C_1 \text{and} C<C_2}(R)$):

$$S_3 = T(Z) + B(\pi_C(Z)) + L(Z, \frac{[B(Z) + \sqrt{B(Z)}]B(Z)}{2T(Z)}) \quad (2.12)$$

І для паралельно випадку, відповідно, отримуємо вираз 2.13:

$$S_3^{'} = \frac{T(Z) + B(\pi_C(Z))}{N} + L(Z, \frac{[B(Z) + \sqrt{B(Z)}]B(Z)}{T(Z)}) + \Omega \quad (2.13)$$

У разі кластерного індексу чисельник першого доданка у виразах оцінок слід замінити на B(Z). Якщо індекс відсутній або не буде використовуватися для запиту, але по відношенню R визначено інший індекс, отримуємо наступні оцінки:

$$S_3 = T(R) + B(\pi_C(R)) + L(Z, \frac{[B(Z) + \sqrt{B(Z)}]B(Z)}{2T(Z)}),$$

$$S_3^{'} = \frac{T(R) + B(\pi_C(R))}{N} + L(Z, \frac{[B(Z) + \sqrt{B(Z)}]B(Z)}{T(Z)}) + \Omega. \quad (2.14)$$

Припустимо, що значення і компактно згруповані за атрибутами і тоді для часу виконання запиту можна навести наступну оцінку:

$$S_4 = B(R1) + B(R2) + L(R1, B(R1)) + L(R2, B(R2)) \quad (2.15)$$

та, відповідно, для випадку параллельного виконання

$$S_4^{'} = B(R1) + B(R2) + 2[L(R1, B(R1)) + L(R2, B(R2))] + \Omega. \quad (2.16)$$

У разі наявності індексів та відсутності компактної згрупованості відношення перший доданок буде мати інший вигляд, що істотно залежить

від типу індексу. У разі також є можливість зменшення величини другого доданку до $L(R1, B(R1)) + L(R2, B(R2))$. У таблиці 2.1-2.3 представлені зведені оцінки часу виконання запитів, зокрема, у таблиці 2.1 наведено апріорні оцінки часу виконання запитів на N вузлах, а в таблиці 2.2 ці-же оцінки на одному вузлі; у таблиці 2.3 наведено формули для очікуваного коефіцієнта прискорення системи.

Таблиця 2.1 – Апріорні оцінки часу виконання запитів на вузлах

Тип запиту	Оцінки швидкості виконання
SELECT	$\frac{B(R)}{N} + 2L(R, B(R)) + \Omega$
GROUP	$\frac{3B(R)}{N} + 2L(R, \frac{B(R) + \sqrt{B(R)}}{2} \cdot \frac{B(R)}{T(R)} + \Omega$
JOIN	$\frac{B(R1) + B(R2)}{N} + 2[L(R1, B(R1)) + L(R2, B(R2))] + \Omega$

Таблиця 2.2 – Оцінки часу виконання запитів на одному вузлі

Тип запиту	Оцінки швидкості виконання
SELECT	$B(R) + L(R, B(R))$
GROUP	$3B(R) + L(R, \frac{B(R) + \sqrt{B(R)}}{2} \cdot \frac{B(R)}{T(R)}$
JOIN	$B(R1) + B(R2) + L(R1, B(R1)) + L(R2, B(R2))$

У цьому розділі розглянуті оцінки часу виконання базових типів SQL-запросов. При побудові оцінок більш складних SQL-запросів необхідно використовувати різні комбінації наведених вище методів оцінок.

Таблиця 2.3 – Коефіцієнт прискорення

Тип запиту	Оцінки швидкості виконання
SELECT	$1-\frac{(2N-1)L(R,B(R))+N\Omega}{B(R)+2NL(R,B(R))+N\Omega}$
GROUP	$1-\frac{(2N-1)L(R,\frac{B(R)+\sqrt{B(R)}}{2}\cdot\frac{B(R)}{T(R)}+N\Omega}{3B(R)+2NL(R,\frac{B(R)+\sqrt{B(R)}}{2}\cdot\frac{B(R)}{T(R)}+N\Omega}$
JOIN	$1-\frac{(2N-1)(L(R1,B(R1))+L(R2,B(R2)))+N\Omega}{B(R1)+B(R2)+2NL(R1,B(R1))+2NL(R2,B(R2))+N\Omega}$

З аналізу відносин часу виконання запитів випливає, що не мають прискорення лише витрати на підготовку запиту до паралельного виконання та час передачі між вузлами системи.

3 МЕТОДИ ТА ЗАСОБИ ПЕРЕТВОРЕННЯ ЗАПИТІВ ДЛЯ ПАРАЛЕЛЬНОГО ВИКОНАННЯ

3.1 Перетворення запитів за умов фільтрації в інструкції WHERE

Застосовується для запитів, у яких відсутня інструкція GROUP BY. При цьому у запиті допускається наявність кількох таблиць та їх з'єднання, застосування оператора сортування ORDER BY. Пошук атрибута (атрибутів) поділу запитів доцільно виробляти у такому порядку:

- у разі присутності інструкції ORDER BY, доцільно проводити розбиття за першими (першими) атрибутами даної інструкції (якщо вони входять до відповідного індексу) та присутні за умови фільтрації WHERE;

- розділення діапазонів на основі індексу, за яким проводиться фільтрація в інструкції WHERE.

Застосовувати таке розбиття доцільно, так як у будь-якому випадку проводитиметься відбір або фільтрація за атрибутом при виконанні запиту;

- у разі присутності інструкції ORDER BY, поділ за діапазонами по першим (першим) атрибутам даної інструкції, якщо відсутні відповідні атрибути в інструкції фільтрації WHERE;

- розділення діапазонів на основі індексу за атрибутом, з відношення, що використовується в запиті.

У разі наявності поняття кластерного індексу в системі доцільно вибирати його атрибути, на основі яких він побудований, для поділу оброблюваного відношення на діапазони.

Вибір певного методу поділу можливий лише тоді, коли для цього є підходящий атрибут (використовується у запиті, або є у базі – залежно від методу). Будемо називати атрибут відповідним, якщо в результаті його вибору як параметра для алгоритму поділу, можливе скорочення обсягів відносин, оброблюваних кожному з вузлів системи, тобто. за значеннями

атрибута побудований індекс і кількість різних значень атрибуту дозволяє рівномірно розділити відношення по заданій кількості серверів. Можливі випадки, коли атрибут буде підходящим, але вибір його не є оптимальним – наприклад, якщо існує інший атрибут, який забезпечує ефективніші умови фільтрації (тобто в результаті фільтрації за цим атрибутом підсумкова кількість кортежів щодо буде менше).

У такому разі, необхідно проводити вибір атрибуту, що забезпечує найкращі умови фільтрації. В іншому випадку можлива ситуація, коли дискретність зміни значень атрибуту не дозволяє забезпечити завантаження всіх доступних ресурсів розподіленої системи, навіть якщо умова фільтрації є ефективною. Наприклад, якщо ми маємо цілісний атрибут, що приймає значення в діапазоні від 10 до 15, то можна паралельно проводити виконання запиту не більше ніж на 6 серверах системи. У цьому випадку доцільно проводити поділ за декількома атрибутами відношення. Для ефективного поділу відносин за декількома атрибутами вони повинні входити в єдиний індекс як його перші ключі. Тоді система зможе розділяти відносини на основі кількох атрибутів.

Позначаючи за zi - діапазон зміни значень і-го атрибута з мінімальним кроком дискретності отримуємо, що кількість серверів, які можна задіяти при виконанні запиту, буде дорівнює $\prod_{i=1}^{I_n} z_i$, де I_N- кількість ключів в індексі. Таким чином, наявність двох-трьох ключів в індексі має забезпечити можливість повного завантаження системи, навіть за малих значень zi. Створення таких індексів залишається за розробником бази даних. Вибір, як атрибути поділу атрибутів не є першими ключами одного індексу, не забезпечує завдання ефективних умов фільтрації кортежів [8].

3.2 Перетворення запиту з агрегуючими функціями та оператором групування атрибутів

Застосовується для запитів, в яких присутня інструкція GROUP BY або функції, що агрегують. У цьому випадку пошук методів поділу для паралельного виконання запиту доцільно проводити в наступному порядку, враховуючи специфіку запиту:

- у разі присутності інструкції ORDER BY, намагатися провести поділ на основі першого атрибута (у разі, якщо він входить в атрибути інструкції GROUP BY);

- поля інструкції GROUP BY потрапляють до умов фільтрації інструкції WHERE;

- поля поділу присутні в інструкції WHERE, але вони відсутні в інструкції GROUP BY (в даному випадку потрібно додаткове агрегування);

- поля поділу присутні в інструкції GROUP BY, але не присутні в умові фільтрації WHERE;

- поле з таблиці, яким є індекс, але поле немає у інструкціях GROUP BY і WHERE (у разі потрібно додаткове агрегування результату).

Обробка списку атрибутів угруповання, якщо потрібно додаткове агрегування, повинна відбуватися на сервері, який повертає результат запиту. Для того, щоб було можливо проводити угруповання атрибутів, необхідно наявність цих атрибутів на машині, що проводить угруповання. Таким чином, при модифікації запиту для паралельного виконання повинен бути модифікований і список атрибутів в інструкції SELECT. До нього мають бути додані відсутні поля угруповання. Звичайно, додані атрибути повертати у підсумковому відношенні не потрібно. Як зазначалося вище, для ефективної обробки групуючих атрибутів, результати виконання запитів на розподілених серверах повинні повертатися відсортованими, як слідування угурупованих атрибутів. Агрегацію атрибутів можна проводити, користуючись алгоритмом, заснованим на ідеї алгоритму сортування злиттям.

Дійсно, оскільки у підсумковому відношенні буде знаходитися рівно кортежів з різним набором атрибутів, що групують, то і на кожному з серверів розподіленої системи їх кількість не може перевищити.

Рисунок 3.1 – Агрегація атрибутів проміжних співвідношень

У середньому, не вводячи додаткових припущень про зв’язок атрибуту розбиття з групуючими атрибутами, можна вважати, що розмір

$S_i = S(N_f + 1)/2$. Таким чином, можна говорити про збільшення проміжного відношення в $(N_f + 1)/2$ раз. У цей же раз зростають витрати на передачу даних між компонентами розподіленої системи. Проте, слід зазначити, що ця операція не вимагатиме збільшення обсягу оперативної пам’яті для зберігання проміжного відношення на вузлі, який повертає підсумковий результат, так як користуючись алгоритмом угруповання атрибутів на кінцевому вузлі системи можна або зберігати підсумковий результат роботи запиту для його подальшого переупорядкування, або повертати його, залежно від структури вихідного SQL-запиту.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ МЕХАНІЗМУ РОЗПАРАЛЮВАННЯ ЗАПИТІВ

4.1 Структура, особливості реалізації, методика вибору алгоритму перетворення

Розглянемо структуру інформаційної системи підприємства, представлену рисунку 4.1. Кожен із відділів підприємства пов’язаний з виділеним сервером, через який здійснюється введення оперативної інформації у систему. Для побудови аналітичних звітів за великими обсягами даних використовується запропонований метод розпаралелювання запитів. У цій конфігурації мережі другий сервер одночасно виступає у ролі вузла для розпаралелювання запитів. Оперативна інформація реплікується по всіх трьох серверах системи.

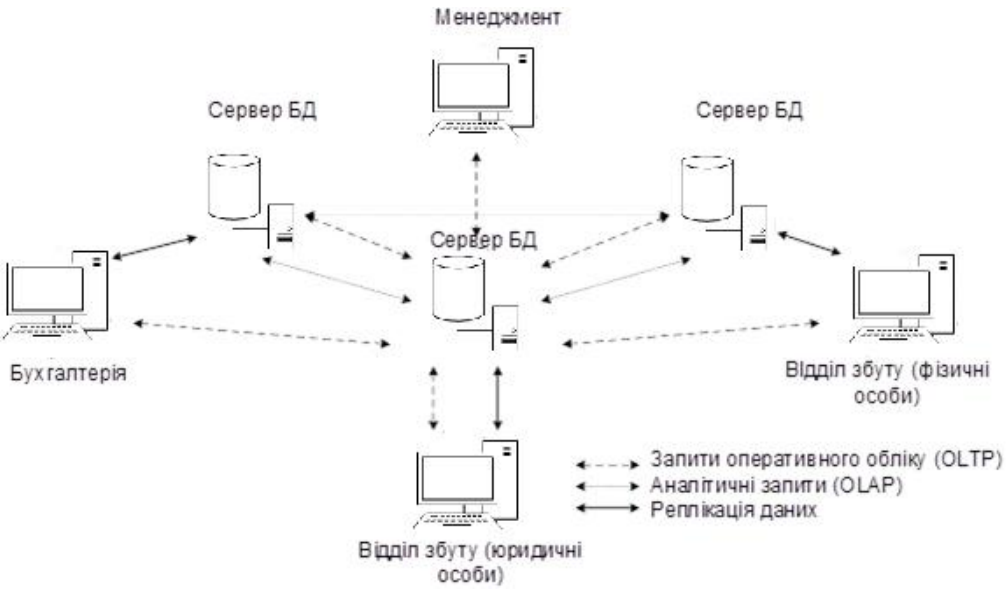

Рисунок 4.1 – Структура інформаційної системи підприємства

Застосування розробленого методу розпаралелювання SQL-запитів на основі їх попередньої модифікації дозволило прискорити виконання

аналітичних запитів в інформаційній системі підприємства на 30-90% (за даними дослідної експлуатації, залежно від типу запиту).

При надходженні SQL виразу програма проводить побудову дерева аналізу даного запиту. В результаті чого програма отримує список, що містить набір відносин, що бере участь у запиті. Програмою визначаються звичайні та кластерні індекси, які побудовані за атрибутами відносин, що беруть участь у запиті. На основі цих даних формується список "потенційних кандидатів для атрибуту розбиття". Нижче описується структури даних таблиць та подано коментарі за значеннями їх атрибутів (таблиці 4.1, 4.2).

Таблиця 4.1 – Структура службової інформації _sysstatind

Назва атрибуту	Тип	Коментар
SysID	int	Унікальний ключ опису індексу
Tablename	varchar(255)	Ім’я відношення, якому належить атрибут
FieldName	varchar(255)	Назва атрибуту в таблиці
IndexName	varchar(255)	Назва індексу
IndexType	int	Тип индексу: 0 = B-Tree, 1= Cluster
FieldType	varchar(255)	Тип атрибуту
MinValue	varchar(255)	Мінімальне значення атрибуту на всьому співвідношеніні
MaxValue	varchar(255)	Максимальне значення атрибуту на всьому співвідношеніні

Для коректного функціонування програми паралельного виконання SQL-запросов потрібно підтримувати у узгодженому стані спеціальні службові таблиці, містять статистичну інформацію з атрибутам відносин бази даних. Таблиця _sisstatindinfo є підпорядкованою таблиці _sysstatind. У ній міститься характеристики розподілу значень атрибутів по діапазонах значень.

Таблиця 4.2 – Структура службової інформації _sisstatindinfo

Назва атрибуту	Тип	Коментар
SysinfoID	int	Унікальний ключ рядку
SysID	int	Унікальний ключ опису індексу (посилання на таблицю _sysstatind)
MinValue	varchar(255)	Мінімальне значення, що характеризує даний діапазон статистичної інформації
MaxValue	varchar(255)	Максимальне значення, що характеризує даний діапазон статистичної інформації
TotalCount	bigint	Число кортежів, що потрапляє до поточного діапазону статистичної інформації
DistrictCount	bigint	Число різних кортежів, що потрапляє до поточного діапазону статистичної інформації

В даний момент підтримуються тільки індекси, що складаються зі значень одного атрибуту, або за значеннями першого атрибуту індексу, що складається з кількох атрибутів. Інформація про статистичний розподіл значень індексу може надаватися як СКБД, так і збиратися будь-якими сторонніми засобами. Наголосимо на важливості підтримки даних таблиць в актуальному стані. Якщо в таблиці _sisstatindinfo виявлятиметься інформація, що не відповідає поточному стану бази даних, то може призвести до значного дисбалансу при розподілі навантаження між серверами системи. Зазначимо, що система може успішно користуватися даними з _sisstatindinfo, якщо зміни з моменту останнього збору статистики були незначними.

Однак якщо значення атрибутів Min Value і MaxValue відношення _sysstatind будуть містити не актуальну інформацію, то в цьому випадку система повертатиме спотворені результати виконання SQL-запиту. Тому

система додавання інформації до бази даних має дбати про підтримку даних атрибутів у актуальному стані. Це не є серйозною проблемою навіть при видаленні даних, оскільки за атрибутами вихідного відношення побудовані індекси.

Після побудови списку "потенційних кандидатів для атрибуту розбиття" на його основі формується список "кандидатів для атрибуту розбиття". Формування даного списку проводиться шляхом виключення з первинного списку атрибутів, якими неможливо розділити відношення, відповідно до значень атрибута, на задану кількість вузлів.

Відповідно до виду запиту, сформований список “кандидатів для атрибуту розбиття” подається на вхід відповідних алгоритмів перетворення запиту. Після чого, проводиться оцінка ефективності виконання модифікованих запитів. У цій реалізації програми використовується процедура отримання оцінки, реалізована в СКБД для якої проводиться підготовка запиту до паралельного виконання. На підставі цієї оцінки здійснюється вибір найефективнішого алгоритму паралельного виконання запиту на основі одного з атрибутів відносин, що знаходиться у списку "кандидатів для атрибуту розбиття".

Після цього проводиться перетворення запиту до набору запитів, придатного для паралельного виконання і власне здійснюється виконання запитів на вузлах системи. Виконання запиту здійснюється шляхом відправки модифікованого запиту на вузол кожної із систем, та подальшого отримання результату його виконання. По закінченні виконання запиту створюється файл, що містить результати виконання запиту. У файлі групуються результати виконання запитів із різних вузлів системи. Поява вихідного файлу є сигналом закінчення виконання запиту.

Ставлення REGION містить 5 кортежів. Вміст рядків відношення наведено у додатку Б. Атрибут відношення R REGIONKEY є первинним ключем відношення. На рисунку 4.2 наведено схему тестової БД.

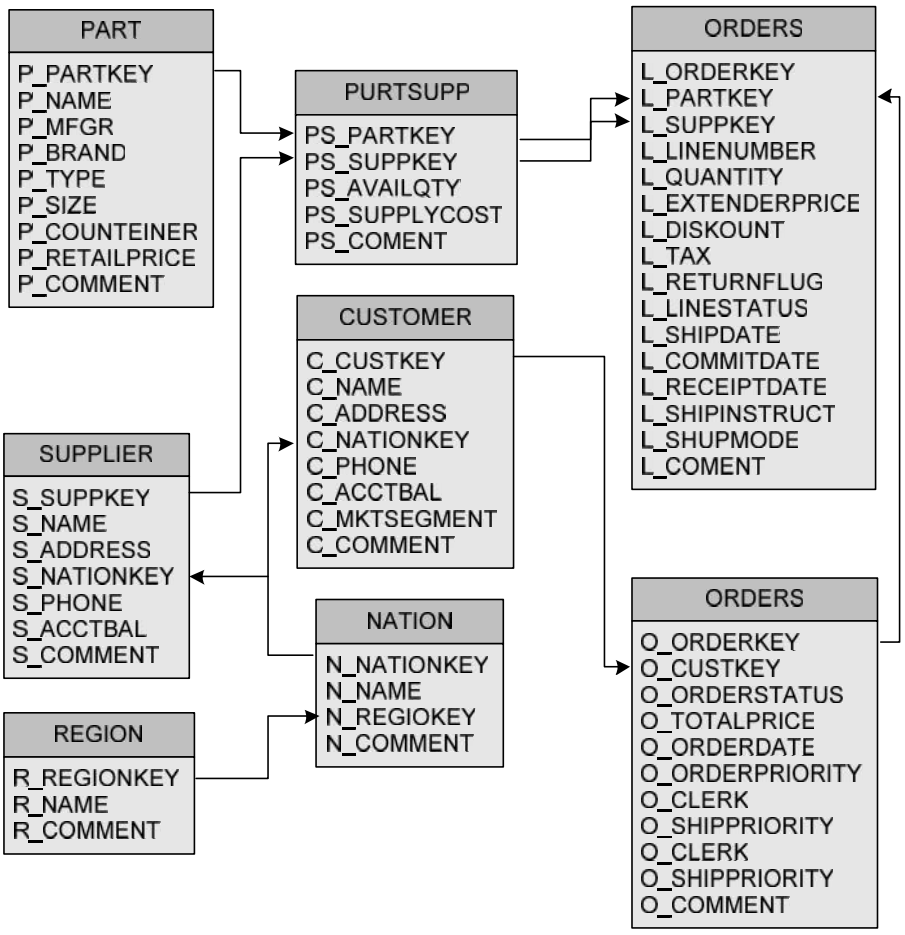

Рисунок 4.2 – Схема тестової бази даних

Стрілки на рисунку 4.2 позначені зв’язок між таблицями в напрямку один до багатьох. Вміст бази створювалося як текстових файлів, які потім були імпортовані в БД, створену згідно зі схемою бази TPC-D. При створенні текстових файлів для завантаження, відповідно до рекомендацій ради TPC, використовувалася програма, розроблена на основі програми з відкритими вихідними текстами DBGEN.

Програма була адаптована для створення файлів у форматі, зручному для імпортування до Microsoft SQL Server. Під час проведення тесту вихідні запити модифікувалися з метою їхньої підготовки для паралельного виконання. Робота багатомашинної конфігурації емулювалася за допомогою одного обчислювального пристрою. Отримані модифіковані варіанти запитів виконувались послідовно. Для того щоб виключити можливість впливу кеша

запитів та дискової підсистеми на час виконання запиту проводився їхнє скидання перед кожним виконанням запиту. Як час, що витрачається виконання запиту в MPP системі, брався максимум за часом, протягом якого виконувались послідовні запити. Як час, що витрачається на підготовку до паралельної обробки запиту, фіксувався загальний час побудови додаткових планів запиту. Час передачі між обчислювальними пристроями по мережевим каналам не враховувалася. Як можна бачити з підрозділу 4.3, таке припущення цілком виправдане, оскільки час виконання запиту кожному з вузлів системи набагато більше часу передачі.

4.2 Результати експериментального дослідження розпаралелювання запитів

Розглянемо з прикладу тестової бази, розробленої підрозділі 4.2, процес паралельного виконання запитів, з урахуванням алгоритмів, наведених у розділі 3. Як запити було обрано перші п’ять запитів, які пропонуються для дослідження продуктивності творцями тесту TPC-D. При виконанні запитів застосовувалися всі способи перетворення запитів, писані в 3-му розділі. Інші запити тесту (їх використовується понад 20, не включаючи різних варіантів одного і того ж запитання) не містять жодних специфічних особливостей, через які до них неможливо було б застосувати досліджувані методи паралельного виконання. Синтаксис запитів був змінено, з метою приведення їх у відповідність з діалектом мови T-SQL, придатному для виконання на SQL сервері компанії Microsoft [6, 7].

Пояснимо вміст таблиць, у яких наведено результати виконання запитів:

- кількість вузлів - позначимо за - кількість обчислювальних пристроїв, виділених для виконання запиту;

- рядків у вибірці – кількість кортежів у результуючій вибірці;

- проміжна вибірка – загальна кількість кортежів у проміжних вибірках

для всіх серверів системи;

- додаткові плани запиту – кількість планів запиту, які були додатково побудовані та проаналізовані під час підготовки до вибору алгоритму для паралельного виконання запиту.

Перед початком тесту наводяться описи запитів, що використовувалися для генерації параметрів. Усі запити під час експериментального дослідження виконувалися при фіксованих значеннях параметрів. Явне завдання параметрів виконання запиту дозволяє системі більш точно підготувати план виконання запиту. При виконанні запитів у системах з різною кількістю обчислювальних пристроїв значення параметрів запитів залишалися незмінними. При виконанні запитів дисковий кеш SQL сервера був порожнім на момент початку запиту. Запити, наведені в додатку 1, є типовими для системи підтримки прийняття рішень [15]. Перший запит обчислює кількість та ціну виробів, які були оплачені, доставлені чи повернені. Запит проводить сканування 95%-97% кортежів відношення lineitem. У кожній з таблиць наводяться результати виконання запитів, розділених за атрибутами відношення lineitem, які були відібрані в результаті евристичної оцінки алгоритму розділу 3.2. Зауважимо, що ці евристичні оцінки не видаляють з подальшого аналізу атрибутів, якими можна проводити розбиття ні першому запиті, ні інших чотирьох випадках. Однак, як можна побачити, вони не видаляють усіх атрибутів, за якими немає сенсу проводити розбиття відносин.

На рисунку 4.3 наведено діаграму з результатами виконання запиту. Тут і далі пунктиром позначено час, який буде потрібно розробленій системі виконання запиту. Як слід апріорно очікувати, воно має бути більше мінімального часу паралельного виконання запиту за фіксованим атрибутом (через необхідність проведення аналізу запиту). Існують шляхи зменшення цієї різниці в часі, наприклад – механізм кешування паралельних запитів.

Також на рисунку 4.3 представлена діаграма, побудована на основі результатів загального часу виконання запиту Q2 для різних атрибутів

розподілу. Зазначимо, що у разі не досягається лінійне прискорення. Це відбувається через те, що вихідні відносини містять мале число кортежів, і час виконання запиту не дозволяє повною мірою компенсувати час, що витрачається на підготовку запиту до паралельного виконання.

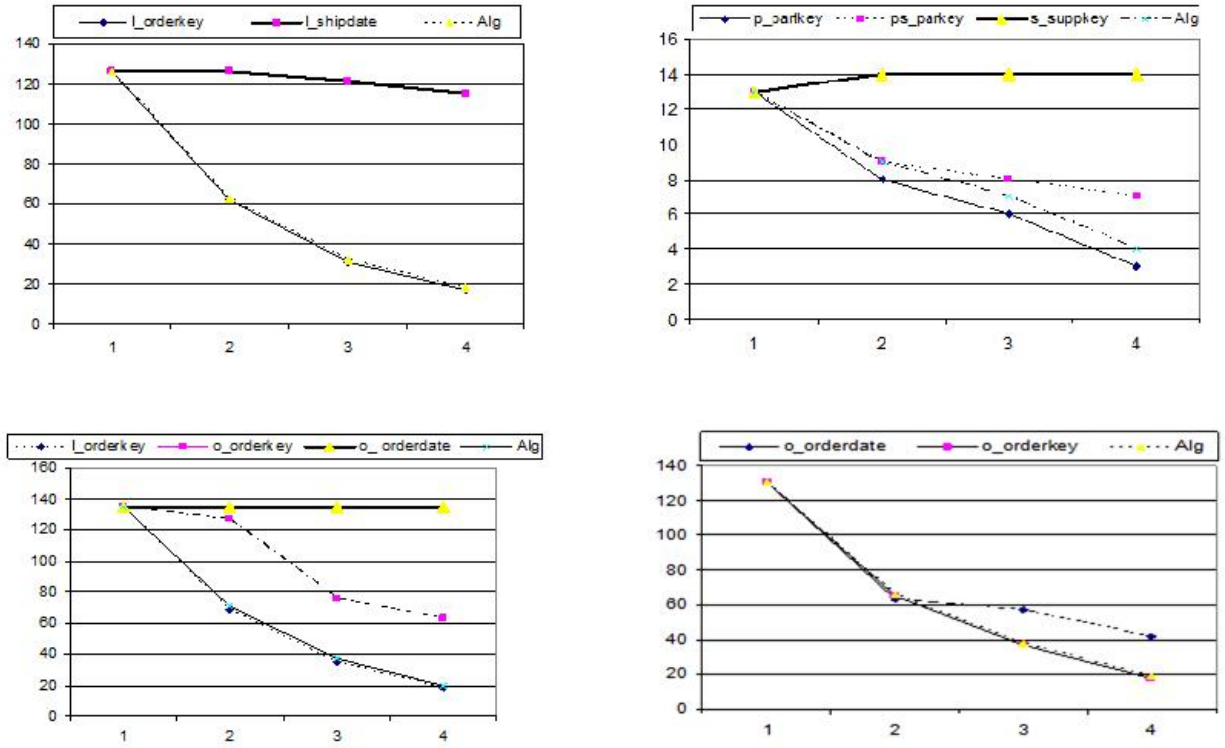

Рисунок 4.3 – Результати роботи

З отриманих результатів видно, що в даному випадку система демонструє прискорення, близьке до лінійного. Зауважимо, що для системи, що складається з двох вузлів алгоритм паралельного виконання запиту, вибирає не найкращий варіант, однак і в цьому випадку Ку залишається близьким до одиниці.

ВИСНОВКИ

З використанням різних архітектур систем для паралельного виконання запитів з використанням надлишкової інформації було розглянуто та досліджено методи розпаралелювання SQL-запитів у СКБД на основі еквівалентного перетворення запитів до форми, що допускає паралельне виконання для застосування в гетерогенних середовищах. Внаслідок чого були отримані апріорні оцінки загального часу паралельного виконання перетворених SQL-запитів, що відрізняються урахуванням особливостей виконання запитів у гетерогенних системах з реплікацією даних, і дозволяють швидко оцінити доцільність досліджуваних методів розпаралелювання запитів на етапі конфігурування системи. Найкращі загальні показники продуктивності системи можна досягти, розробляючи систему на основі MPP архітектури, допускаючи використання SMP машин як вузлів системи, що фактично перетворює її на гібридну.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Варшавский П.Р., Еремеев А.П. Моделирование рассуждений на основе прецедентов в интеллектуальных системах поддержки принятия решений // Искусственный интеллект и принятие решений. 2009. № 2. С. 45-47.

2. Ахо А. Структуры данных и алгоритмы [Текст] / А. Ахо, Дж. Хопкрофт, Дж. Ульман – М.: Издательский дом “Вильяме”, 2000. – С. 384.

3. Воеводин В. В. Параллельные вычисления [Текст] / В.В. Воеводин -СПб.: БХВ-Петербург, 2002. – 608 с.

4. Волков В.В. Тесты TPC [Текст] / В.В. Волков СКБД. – 1995. – №2. – С. 70-78.

5. Вьейра Р. SQL Server 2000. Программирование. Часть 1 [Текст] / Р. Вейра – М.: Бином, 2004. – 736 с.

6. Вьейра Р. SQL Server 2000. Программирование. Часть 2 [Текст] / Р. Вейра -М.: Бином, 2004. – 808 с.

7. Ульман Д. Системы баз данных. Полный курс [Текст] / Г. Гарсиа-Молина, Д.Ульман, Д.Уидом. – М. “Вильямс”, 2003. – 1088 С.

8. Дейт К. Введение в системы баз данных [Текст] / К. Дейт – М. Вильямс, 2000 – 846 с.

9. Зильбершатц А. Базы данных: достижения и перспективы на пороге 21-го столетия [Текст] / А. Зильбершатц, М.Стоунбрейкер, Д. Ульман. – СКБД. – 1996. – №3. – С. 103-117.

10. Игнатович Н. Семейство реляционных баз данных IBM DB2 [Текст] / Н. Игнатович. – СКБД. – 1997. – №2. – С. 5-17.

11. Кнут Д.Э. Искусство программирования, т. 3. Сортировка и поиск [Текст] / Д.Э. Кнут. – М.: Издательский дом “Вильяме”, – 2000. – 832 с.

12. Когаловский М.Р. Энциклопедия технологий баз данных [Текст] / М.Р. Когаловский – М.: Финансы и статистика, 2002. – 800 с.

13. Когаловский М.Р. Электронные библиотеки – новый класс информационных систем [Текст] / М.Р. Когаловский. – Программирование, 2000. – №3. – С. 3-8.

14. Коннолли Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика [Текст] / Т.Конноли, К.Бегг, А.Страчан – М.: Вильяме, 2000. – 1120 с.

15. Корнеев В.В. Архитектуры с распределенной разделяемой памятью [Текст] / В.В. Корнеев. – Открытые системы, 2001. – №3. – С. 15-23.

16. Корнеев В.В. Параллельные вычислительные системы [Текст] / В.В. Корнеев – М.:"Нолидж", 1999. – 320 с.

17. Кузнецов С.Д. Развитие идей и приложений реляционной СКБД System R [Текст] / С.Д. Кузнецов. – М.:ВИНИТИ, 1989. –с. 3-75.

18. Кузьминский М. Современные суперкомпьютеры: состояние и перспективы [Текст] / М. Кузьминский, Д. Волков. – Открытые системы, 1995. – с. 33-40.

19. Кузьминский М. Архитектура S2MP – свежий взгляд на cc-NUMA [Текст] / М.Кузьминский. – Открытые системы, 1997. – № 2. – С. 14-21.

20. Лисянский К. СКБД Teradata для ОС UNIX [Текст] / К.Лисянский, Д.Слободяников – СКБД, 1997. – №5-6. – С. 25-46.

21. Локшин М.В. Модификация деревьев разбора для параллельного исполнения запроса [Текст] / М.В. Локшин. – Успехи современного естествознания, 2004. – №10. – С. 70-71.

22. Локшин М.В. Об одном методе параллельного выполнения запросов в распределенных СКБД [Текст] / М.В. Локшин. Современные проблемы информатизации в системах моделирования, программирования и телекоммуникациях: Сб. трудов. Вып. 9. Под ред. д.т.н., проф. О.Я. Кравца – Воронеж: Издательство “Научная книга”, 2004. – С. 360-361.

23. Локшин М.В. Обработка запросов к большим объемам данных в СКБД [Текст] / М.В.Локшин. – Актуальные проблемы информатики и информационных технологий: Материалы Международной (VIII Тамбовской

межвузовской) научно-практической конференции (9-10 сентября 2004 г.). Тамбов: Издательство ТГУ им. Державина, 2004 – С. 69.

24. Локшин М.В. Система для параллельного исполнения запросов к базам данных [Текст] / М.В.Локшин. – Седьмая региональная молодежная научная и инженерная выставка “Шаг в будущее, Центральная Россия”: Сборник тезисов докладов. Липецк: Липецкий государственный технический университет – С. 25-26.

25. Локшин М.В. Эквивалентное преобразование SQL запроса к запросу с диапазонами в условии фильтрации [Текст] / М.В.Локшин. - Современные проблемы информатизации в системах моделирования, программирования и телекоммуникациях: Сб. трудов. Вып. 10. Под ред. д.т.н., проф. О.Я. Кравца – Воронеж: Издательство “Научная книга”, 2005. – С. 255-256.

26. Кузьома Т.М., Кулик С.В., Лебедєв О.Г. Методи доступа до сховищ даних в паралельних розподілених системах// Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. Матеріали одинадцятої міжнародної НТК. – Баку: ВА ЗС АР; Харків: НТУ «ХПІ»; Харків: ДП«ПДПРОНДІАВІАПРОМ»; Жиліна: УмЖ, 2021. – 8-9 квітня 2020. – С. 56.