**Паралелно и дистрибуирано процесирање на прашалници**

Христијан Шурбески, д-р Горан Велинов

Факултет за информатички науки и компјутерско инженерство

**Апстракт:** Два начини на процесирање на прашалници кои се есенцијални во оптимизацијата и работата со базите на податоците. И двата начини се денес користени од причините на ерата Big data каде имаме голем број на податоци, со кои ние треба брзо и ефикасно да манипулираме, да ги обработуваме и да вадиме заклучоци.

**1 Вовед**

Со развивањето на процесирачката моќ кај машините за пресметување односно кај компјутерите, ширењето на бизнисите на различни геолокациски места и енормното зголемување на податоци, доаѓаме до момент кога нашите пресметувачи перфоманси се многу лоши, а сепак имаме голема процесирачка моќ кај машините за пресметување и обработување на податоците. Тука корисниците и експертите забележале многу големо забавување кога обработуваме и анализираме податоци со помош на прашалници. Во зависност од проблемoт ние пишуваме одреден прашалнил во SQL кој се преведува во рационална алгебра. Сега ќе се навратиме на моделот од насловот т.е како се извршуваат тие прашалници. Пред да почнеме мораме да се запознаеме со два термини пралелено извршен прашалник и дистрибуиранo извршен прашалник. Овие се двата начини со кој ние имаме побрзи резултати од некој прашалник, но тие се однесуваат различно! Па во дистрибуиран прашалник, ние земаме една голема табела и ја распределуваме преку кластер, и потоа го делиме нашиот прашалник на посебни делови за да се оперира на секоја од тие партиции во фајлот, потоа сите резултати од тие индивидуални делови се враќаат назад до главниот јазел, до еден сервер за да го заврши процесирањето. Пример, ако ги броиме сите записи во табелата кои задоволуваат некој услов, па ако имаме многу голем фајл што се дели на неколку машини, тогаш ова се дистрибуирани системи за прашалници. Повеќето SQL сервери како Microsoft SQL и други се пример за ова, тие се доволно паметни за да го разделат прашалникот на повеќе мали делчина кои се обработуваат посебно. Како што почнуваат да генерираат торки, тие одма почнуваат да ги праќаат назад до главниот јазел. Но тука доаѓа идејата дека може овие SQL сервери да се попаметни односно да почнат да ги обработуваат прашалниците паралелно на еден сервер и на крај да ги составиме резултатите. Но треба да знаеме дека е тешко да се состави ваков прашалник каде имаме тесно грло на проток на податоци кои се праќаат до еден сервер. Сега, паралелни прашалници, тие ги извршуваат прашалниците паралелно на еден мултипроцесорски компјутер. Како процесирање на прашалникот, трансформацијата на прашалникот во план за извршување резултатот мора да е точно поставен и да дава ефикасно извршување на истиот. Точноста се одржува со мапирање во рационална алгебра, која дава добар абстракт на системот за извршување. Ефикасно извршување е суштинско за високи перфоманси.

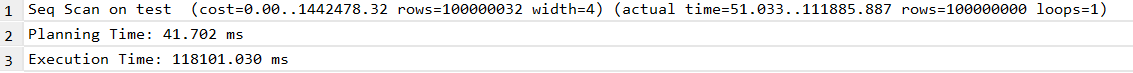
Сега во овој труд ќе направиме истражување, односно ќе преминеме на практичниот дел од оваа тема. Пред се во мојот проект по овој предмет имам потреба од паралелно извршување на прашалници со цел оптимизирање на базата. Јас ќе го разработам и анализирам процесот на паралелизација на прашалниците и делот од дистрибуираното процесирање на прашалниците.

Прво нешто што додадов е нова табела што ќе ги чува сите податоци од сите трансакции од различни продавници. Да претпоставиме дека компанијата ТопШоп работи одреден број на години доволно да достигне 100 000 000 записи во таа табела. Најчесто барање што го правиме до базата е некој секвенцијално скенирање. Прво нешто што ќе направиме е да видиме кој ќе биде резултатот од пребарувањето без паралелизам во прашалникот.

**2 Паралелно процесирање на прашалници**

Ова може да биде побрзо, не поради паралелните читања, туку тоа се должи на тоа што податоците се делат низ многуте јадра што ги има компјутерот. Модерните ОС обезбедуваат добро кеширање за PostgreSQL податочните фајлови. Читањето унапред овозможува земање на блок од меморија повеќе отколку самиот блок побаран од  PG daemon. Како резултат, перформансите на прашалникот не се долшат на дисковите IO. Конзумира CPU циклуси за: читање на редици едена по една од табела, и споредување вредности од редиците и WHERE услови. Сека ќе пробаме еден едноставен прашалник.

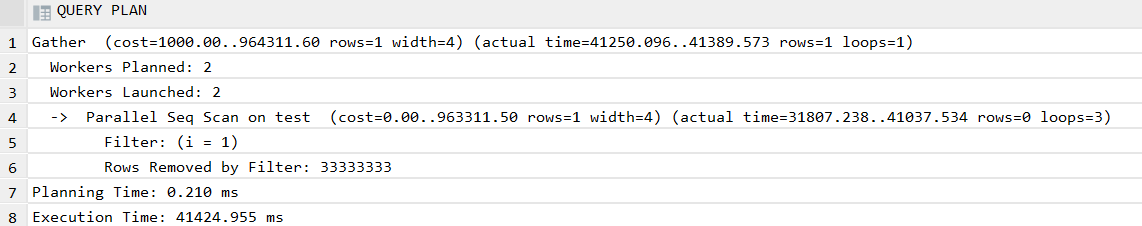
**EXPLAIN ANALYSE SELECT** *\** **FROM** test;



**Слика 1.1)**

Гледаме колкаво е времето за извршување, заклучуваме дека е големо но исто така не забежуваме паралелизам. Тоа е така бидеќи во овој случај само се поделија податоците меѓу јадрата. За да поттикнеме паралелизам треба да додадеме некој услов или агрегатна финкција, па ќе забележиме дека и покрај уште една проверка во прашалникот тој побрзо ќе се изврши.

**EXPLAIN ANALYZE SELECT** *\** **FROM** test **WHERE i**=1;



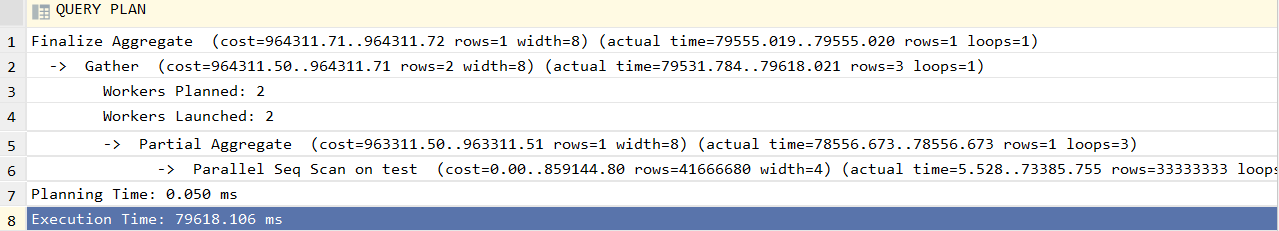
**Слика 2.2)**

Сега гледаме тука многу големо заврзување, односно времето на извршување е многу помало. Пред да продолжиме да се запознаеме со термините кои ги гледаме на сликата.

Gather – е јазел кој ги спојува резултатите добиени од паралелните работници, ако тој е на врвот од планот тогаш тој прашалник се извршува паралелно, ако е подолу тогаш делот и подпрашалникот ќе се изврши со пралалелизам. Со употреба на EXPLAIN, може да ги видиме бројот на работници избрани од планерот. Кога Gather јазелот е достигнат при извршување на прашалникот, процесот во кој ја имплементира корисничката сесија ќе побара број од background worker processes еднаков на бројот од работинци избран од планерот. Сега секако дека имаме ограничување на бројот на работници, максимум е 8 а стандард кај PostgreSQL е 2, но може да се прават модификации со функциите max\_worker\_processes и max\_parallel\_workers. На секој background работинк кој што успешно почнал со работа за даден паралелен прашалник ќе изврши својот дел паралелно, од планот. Е сега тука имаме и водач т.е leader worker, тој исто ќе го извршува својот дел од планот, паралелно, но има и дополнителна одговорност: мора да ги чита сите торки генерирани од останатите работници. Кога паралелниот извршен дел од планот генерира мал број на торки, водачот често ќе се однесува многу повеќе како додатен или помошник работник, кој го забрзува извршувањето на прашалникот. Но во случај кога има голем број на генерирани торки, тогаш лидерот само ги чита, и прави чекори кои се предвидени кои се според планот над нивот на Gather Merge node. Кога најгоре од планот е Gather Merge него Gather, индицира дека секој процес кој се извршува во паралелни делови од планот прави торки во сортиран редослед, и дека лидерот прави подредено спојување. За разлика од Gather каде чита торки од работници во било кој редослед, уништуваќи било каков редослед ако постоел.

Ајде сега да додадеме и агрегација и да го анализираме случајот.

**EXPLAIN ANALYSE SELECT** *sum*(**i**) **FROM** test;



**Слика 2.3**)

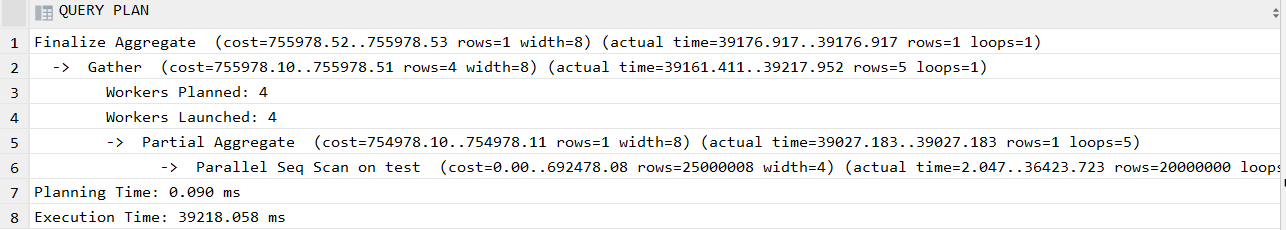
Покомплексниов прашалник е 1,4Х побрз спореден со едноставниот, со една нишка select-от горе. Сега имаме нови работи во овој план. “Parallel Seq Scan” јазелот прави редици за оделни агрегации. “Partial Aggregate” јазелот ги намалува овие редици со SUM(). На крај резултатот се пресметува во “Finalize Aggregate”.

*Број на работници*

Секако дека може да го зголемиме бројот на работници без да го рестартираме серверот.

**SET** max\_parallel\_workers\_per\_gather **TO** 4;

**EXPLAIN ANALYSE SELECT** *sum*(**i**) **FROM** test;



**Слика 2.4**)

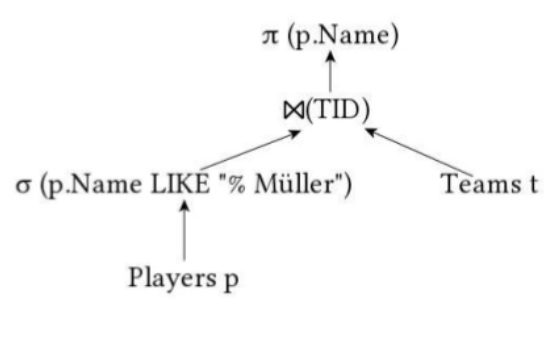
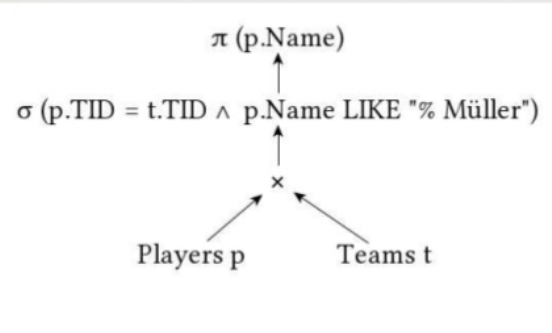
Што се случува сега овде? Ние го сменавме бројот на работници од 2 во 4, но прашалникот сега е 2.03Х побрз. Всушност, скалирањето е супер. Ние имавме 2 работници плус 1 лидер. После смена на конфигурацијата, стана 4+1. Сега следи прашањето како тие работат?!

Извршувањето на прашалникот секогаш почнува со процесот “лидер“. Лидерот ги извршува сите непаралелни активности и тој има посебна соработка со паралелното процесирање. Останатите процеси кои го извршуваат прашалникот се наречени “работници” процеси. Тие комуницираат со лидерот преку редица за пораки. Секој процес има две редици, една за грешки а другата за торки.

**3 Дистрибуирано процесирање на прашалници**

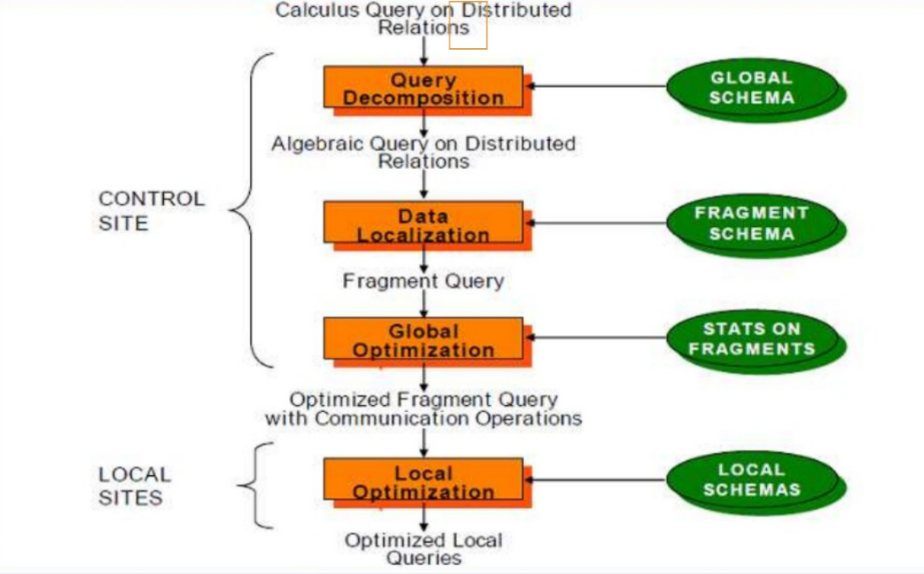
Да почнеме од почеток, мора да имаме дистрибуирана база, односно повеќе, логички меѓусебно поврзани бази, дистрибуирани т.е распределени низа целата мрежа. Процесирањето на овие прашалници би одело во овој редослед: прво се преведува напишаниот прашалник од висок-нивовски јазик како релационен калкулус во секвенца од оператори на базите на податоци како рационална алгебра плус комуникациските оператори. Еден високо нивовски прашалник може да има многу еквивалентни трансформации, главната тешкотија тука е да се одбере најефикасната. Ќе почнеме со пример друга база која ја направив за оваа тема. Треба да ги најдеме сите играчи со име “Muller”, кои играат во тим.

QUERY: SELECT p.Name FROM Players p, Teams t WHERE p.TID = t.TID AND p.Name LIKE “MULLER”



**Слика 3.2**) Решение 2

**Слика** **3.1**) Решение 1



**Слика 3.3**) Земањето на податоци е од различни страници во мрежата. Методологијата за работа е следна:

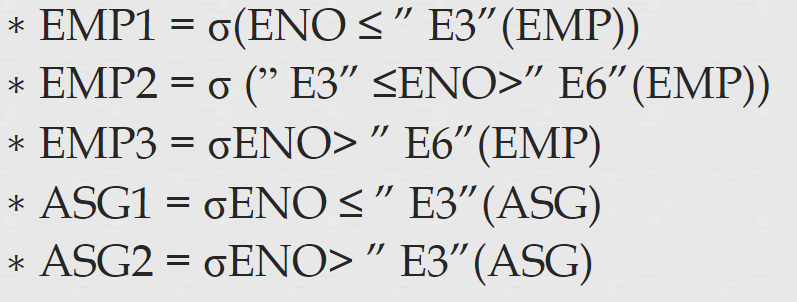
Прво чекор тука е декомпозиција, како што кажавме претходно, преведување од високо ниво во ниско ниво на јазик за манипулација со податоци. Во декомпозицијата имаме 4 чекори а тоа се:

1. Нормализација: Препишување на прашалникот во нормализирана форма, корисна за понатамошна манипулација.
2. Анализа: Прашалниците се анализирани семантички па неточните прашалници се детектираат.
3. Елиминација на редундантност: Елиминираме редундантни предикати.
4. Препишување: Релациониот калкулус (прашалникот) е преведен во еквивалентен израз на алгебра.

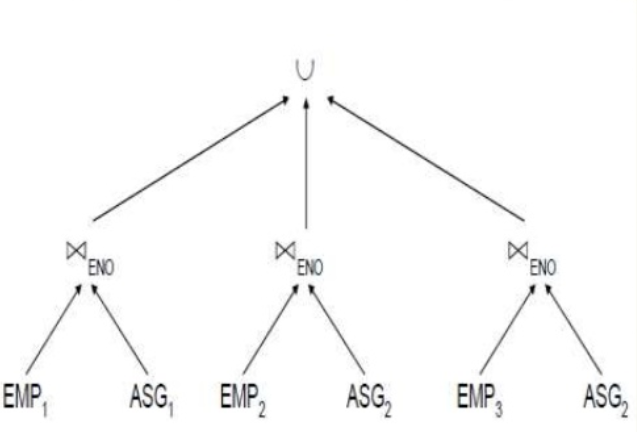
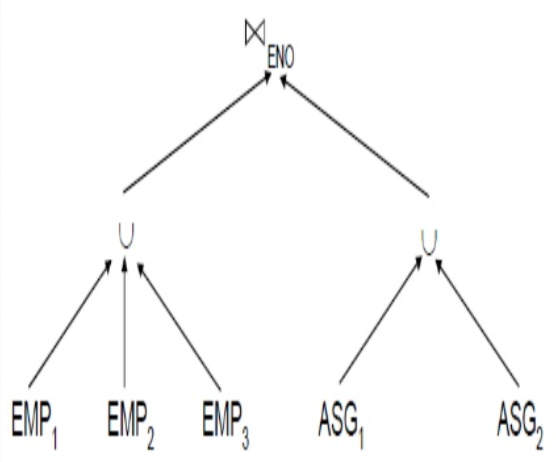
Втор чекор од дистирбуираниот прашалник е локализирање на податоците. Пример имаме за влез некој алгебарски прашалник на распределени релации. Целата е да се аплицира информациите за распределените податоци на алгебарските операции и одреди кои фрагменти се вклучени па потоа замени го глобалниот прашалник со помали фрагментирани прашалници, па оптимизирај го глобалниот прашалник. Пример како се фрагментира еден прашалник.

SELECT \* FROM EMP, ASG WHERE EMP.ENO = ASG.ENO;

Имаме два тиме на фрагментација хоризонтална и вертикална, примерот е со хоризонтална.



**Слика 3.4**) Хоризонтална фрагментација



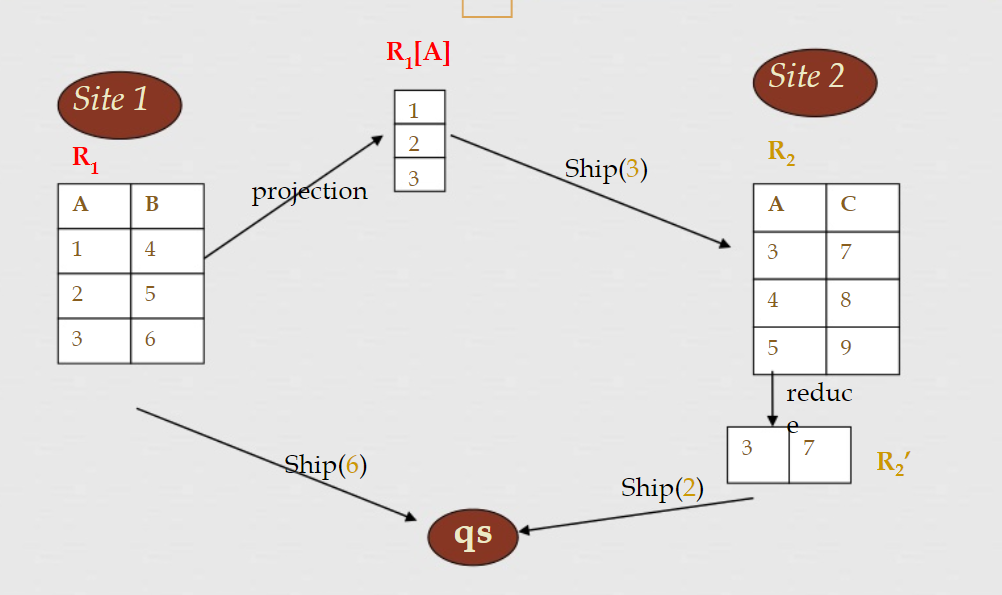
**Слика 3.5**) Генериран прашалник

**Слика 3.6**) Редуциран прашалник

Третиот чекор од методологијата е оптимизација на глобалниот прашалник. Треба да направиме минимизација на cost function. Процесирање на распределите joints односно која релација каде да се испрати, и испрати цела релација vs. испрати колку што треба. И за крај одлука за користење на semi joins. Пример за ист резултат:

1. Префрли Employee на страна 2, изврши Join на страна 2 и прати ги резултатите на страна 3. Резултати од прашалникот = 40 \* 10 000 =400 000 bytes. Total transfer size = 400 000 + 1 000 000 =**14 000 000 bytes**
2. Префрли Department релацијата на страна 1, изврши join на страна 1, и прати ги резултатите на страна 3. Total bytes transferred = 400 000 + 3500 = **403 500 bytes.**

Заклучуваме дека е многу битно како ќе го составиме прашалникот и како ќе ги извршуваме join-тите, и треба исто така да користиме semi joins за да се намали цената на пренос на податоци. Следи илустрација за semi joins:



**Слика 3.6**) Употреба на Semi joins

1. Проектирај ги атрибутите за joins на Department на страна 2, и потоа префрлиги на страна 1. За Q: 4\*100 = 400 bytes се префрлени.
2. Направи јoin на префрлените податоци со релацијата EMPLOYEE на страна 1, и префрлиги бараните атрибути од резлутирачкиот фајл на страна 2. Q: 34\*10000=340 000 bytes are transferred.
3. Изврши го прашалникот со спојување на префрлениот фајл со Department и презентирај го резултат на корисникот на страна 2.

Со употреба на оваа стратегија, ние префрливме оптимални 340,400 бајти за Q.

И за крај ни останува локални оптимизациски техники.

**4 Заклучок**

Двете техники се заслужни за денешното функционирање на системите за масовно опслужување кои се поврзани со базите на податоци. Во ерата на big data овие техники се клучни за успешно работење на нашите системи. Битно е нивното разбирање за нивна соодветна употреба. Во PostgreSQL паралелното процесирање е вклучено од самиот старт, но не значи дека мора да се користи за оптимизирање, ако тоа може да го направиме со индексирање и соодветно делење.

**Референци:**

1. https://www.sciencedirect.com/science/article/pii/S1570866708000749
2. Fundamentals of Database Systems - Elmasri Navathe, 7th Edition
3. Optimization of Parallel Query Execution Plans in XPRS (еXtended Postgres on Raid and Sprite) – Wei Hong and Michael Stonebarker, University of California at Berkeley Berkeley, CA 94720
4. https://docs.oracle.com/cd/A58617\_01/server.804/a58238/ch1\_unde.htm
5. https://www.postgresql.org/docs/