CS631 Project
Query Decorrelation for PGSQL

Nikhilesh Sharma - 07305045
Avishek Ghosh - 07305048
Election Reddy - 07305054
Amitraj S. Chouhan - 07305056

November 13, 2007
Contents

0.1 Introduction ............................................. 3
0.2 Description ............................................. 3
0.3 Scope of the Project ................................. 4
  0.3.1 Definite Plans .................................. 4
  0.3.2 How to do ........................................ 4
0.4 Implementation Issues ............................. 6
0.5 Some Implementation Details ....................... 8
0.6 Some More Decorrelation Examples ............... 12
  0.6.1 Query with any comparison operator .......... 12
  0.6.2 Query with Aggregate functions .............. 12
  0.6.3 Query with exists ............................. 13
0.7 Future Work ......................................... 14
List of Figures

1  Query processing by PostGreSQL server . . . . . . . . . . 7
2  Query processing flow by PostGreSQL server after addition of Decorrelation Module . . . . . . . . . . . . . . . . . 8
3  Internal architecture of Decorrelation Module . . . . . . . 9
0.1 Introduction

We have introduced Query Decorrelation in PostgreSQL. At present the Query Decorrelation is not implemented in PostgreSQL. A correlated query is a nested query in which the inner query refers any attribute of outer query. PGSQL evaluates a correlated query by evaluating the subquery separately for each tuple in the outer level query. Due to that a large no of disk I/O may result. Decorrelation involves process of converting a correlated query into an unnested query that gives similar result as the original correlated query gives. Decorrelation is a part of Query Optimization Process.

0.2 Description

A correlated query is a nested query in which the inner query refers any attribute of outer query. Lets take an example:

We have 2 relations one is student that has attributes roll_no, name and cpi these attributes contains information about student’s roll number, name and overall CPI respectively another relation is irdbms having attributes Roll_no and cpi which contains Roll No of the student who enrolled for the irdbms course. Now consider the following query

```
SELECT roll_no,name  
FROM student  
WHERE student.cpi = (SELECT irdbms.cpi  
FROM irdbms  
WHERE student.roll_no = irdbms.Roll_no)
```

this query selects Roll number and Name of those students who have same overall CPI as they got in irdbms. This is an example of correlated query. Here inner query refers attribute roll_no of outer relation in the where clause. When SQL executes this query it executes whole inner query for each tuple present in outer level query. This is a very expensive approach in terms of query processing cost. It may require a large number of Input/Output i.e. large number of disk access. To improve the performance of the Database System it is suggested that we should not allow queries which require such kind of multiple execution for each tuple of outer rela-
tion. So we need a query that give us same result as our previous query gives us, but without involving such a costly execution or in other words we need a query that give us same result but doesn’t involve correlation. Here comes Decorrelation.
Decorrelation is a process of replacing a correlated query by a query with a join(possibly with a temporary relation) that should give the same result as the original correlated query gives. [1] Decorrelation involves manipulation of original correlated query and converting it into a decorrelated form. A decorrelated query is unnested query generated from original correlated query that gives same result as original query gives. For example the given correlated query can be written without correlation as follows:

```sql
SELECT student.roll_no, student.name
FROM student, irdbms
WHERE student.cpi = irdbms.cpi
and student.roll_no = irdbms.Roll_no
```

We can say this is the decorrelated form of the given correlated query. Here we can easily see that this query is more efficient in terms of no of disk I/O as compared to the correlated query. So decorrelation improves the performance of database system.

### 0.3 Scope of the Project

#### 0.3.1 Definite Plans

We planed to implement the query decorrelation by modifying the original correlated query tree structure into desired decorrelated tree structure i.e. we have implemented decorrelation rules on the tree structure of original correlated query that is generated in the query evaluation plan phase, after applying rules and converting it into a decorrelated query tree structure we pass the tree to the optimizer for rest of the processing.

#### 0.3.2 How to do

A correlated query can be decorrelated by converting it into a query with a join(possible with a temporary table). However it is not possible(or very
difficult) to decorrelate all type of queries, but some basic rules can be applied to decorrelate some type of simple queries. First we will see types of Correlated queries. There are basically following types of correlated queries:

1. Queries with comparison operator in where clause
   
   ```sql
   select T_1.a from T_1 where T_1.b \text{ op } (select T_2.c from T_2 where T_2.d = T_1.d )
   ```

   here any comparison operator (like $=,\neq,\leq,\geq,\lt,\gt$) can appear in place of $\text{ op }$.

2. Queries with ALL/ANY in where clause
   
   ```sql
   select T_1.a from T_1 where T_1.b \text{ op all}(select T_2.c from T_2 where T_2.d = T_1.d )
   ```

   ```sql
   select T_1.a from T_1 where T_1.b \text{ op any}(select T_2.c from T_2 where T_2.d = T_1.d )
   ```

   here any comparison operator (like $<,\leq,\geq,\gt$) can appear in place of $\text{ op }$.

3. Queries with IN in where clause
   
   ```sql
   select T_1.a from T_1 where T_1.b \text{ in } (select T_2.c from T_2 where T_2.d = T_1.d )
   ```

4. Queries with EXISTS/NOT EXISTS in where clause
   
   ```sql
   select T_1.a from T_1 where T_1.b \text{ exists}(select T_2.c from T_2 where T_2.d = T_1.d )
   ```

   ```sql
   select T_1.a from T_1 where T_1.b \text{ not exists}(select T_2.c from T_2 where T_2.d = T_1.d )
   ```

   here any comparison operator (like $<,\leq,\geq,\gt$) can appear in place of $\text{ op }$.

We can generalize a decorrelation rule as:

[2] Say there is a relation operator $\circ$ or input $R$, with a scalar argument $e$ using subquery $Q$. We execute the subquery first Apply, such that the subquery result is available as a (new) column $q$; then replace the subquery utilization by such variable:

\[
\circ_{e(Q)} R \rightarrow \circ_{e(Q)}(RA^{\otimes}Q)
\]
In a very basic case this apply operator $A^\otimes$ can be a simple join or a left outer join, so we can replace a correlated query by a decorrelated query using join. To do this we need to analyze basic difference between the tree structures of correlated query and its decorrelated form and need to do desired manipulation in the correlated tree structure to convert it into a decorrelated form.

### 0.4 Implementation Issues

To implement decorrelation we have studied different proposed query decorrelation technique [2] for different types of queries that we have categorised in previous section. We first identify the type of query, then on the basis of the technique that is proposed for decorrelation of that query, we deal with the query and convert it into decorrelated form. PostgreSQL server query processing flow is shown in fig 1. In this query string is passed to Parser module. Parser does the parsing and generates a parse tree which then is passed to the Rewriter module that generates a rewritten parse tree and passes it to Planner, then the planner generates an efficient plan tree using that rewritten parse tree and looking up the statistics. This plan tree is passed to the Executer, then executer executes the query and generates the result. These all phases interacts with database definition, rules and tables for processing.

We have designed a Decorrelation module that is placed between Rewriter and Planner module. This decorrelator deals with the rewritten parse tree. It checks that whether the given parse tree contains a sublink or not and whether the quals attribute of the sub-query contains variable of different levels? i.e. this is a correlated query or not. If yes then it passes the sub query to next condition checking, where the kind of correlated query is checked i.e. the subquery falls under which type among the types we have defined previously. After type identification we pass the subquery to appropriate function and if the query doesn’t contain sublink or query is not correlated then decorrelation module simply passes this query tree to next module without making any changes in it. After Addition of our Decorrelation Module in PostgreSQL Server the query processing flow is shown in figure 2.

We have written several functions in Decorrelation module, each function deals with a fix type of query and decorrelates it. Here different functions
are required for different type of queries because decorrelation process for each type of query is fairly different from the other type of queries. Decorrelation is a very complex process, it is not possible to decorrelate all queries applying same rules. There are some correlated queries (like NOT IN) which haven’t been decorrelated till this day. So there are no general rules for decorrelation which will work correctly for decorrelation on all correlated queries. Figure 3 describes the internal structure of Decorrelation Module.

As shown in figure 3 there is a check for correlation, if correlation is not present then it simply bypasses the whole remaining decorrelation module and passes the query tree to Planer but if decorrelation is present then there is another check for the type of correlation this is like a case construct, on the basis of type it passes the query to appropriate function (represented by Function 1, Function 2 etc). Each function is designed to deal with a specific type of query. The function changes the tree structure into the decorrelated tree structure that gives same result as the original query gives. Now the resultant tree structure is passed to the next phase.
0.5 Some Implementation Details

We have analysed the query trees of both correlated and decorrelated form of each type of query we are going to decorrelate. Postgresql represents the query tree as a linkedlist. Some parts of query tree of one of the query we decorrelated are explained below.

Query: select T1.c from T1 where T1.a in (select T2.a from T2 where T1.b = T2.b)

Query tree description:

```
:commandType 1
:resultRelation 0
:into <>
:hasAggs false
:hasSubLinks true
```

commandType: This value tells which command (SELECT, INSERT, UPDATE, DELETE) has produced this query tree. Its value is 1 for SELECT command.

resultRelation: This is an index into the range table that identifies the relation where the results of the query go. SELECT queries normally
don’t have a result relation.

**hasSubLinks:** This value tells whether there is a subquery or not. Its value is **true** for correlated queries and **false** for decorrelated queries.

```plaintext
:rtable (  
:alias <> 
:eref 
   :aliasname t1 
   :colnames ("a" "b") 
:rtekind 0 
:relid 16390 
:inh true 
:inFromCl true 
:requiredPerms 2 
:checkAsUser 0
```
The range table is a list of relations that are used in the query. In a 
**SELECT** statement these are the relations given after the **FROM** key word. Every range table entry identifies a table or view and tells by which name it is called in the other parts of the query. In the query tree, the range table entries are referenced by number rather than by name.

In correlated query tree there is only one table entry i.e, outer table in **rtable** list. But in decorrelated query, as the inner table is taken out, its entry has to be entered into rtable. This we have done using following code.

```c
List * rt = ourSubQuery->rtable;
ListCell *ourList_item;
foreach(ourList_item, rt)
{
    void * ourItem = lfirst(ourList_item);
    ours->rtable = lappend(ours->rtable, ourItem);
    tableNum++;
}
```

```
:jointree
 :fromlist
 :quals
  :groupClause <>
  :havingQual <>
  :distinctClause <>
  :sortClause <>
  :limitOffset <>
  :limitCount <>
  :setOperations <>
  :resultRelations <>
 :rowMarks <>
 :forUpdate false
 :rowNoWait false
 :targetList ( 
   :expr
   :resno 1
   :resname a
   :ressortgroupref 0
```
The query’s join tree shows the structure of the FROM and WHERE clauses. As there is change of subquery to join while decorrelation, we have to change jointree. The code for the jointree creation is

```c
SubLink * ourSubLink = ourFromExpr->quals;

for(j = 0; j < tableNum; j++)
{
    RangeTblRef *rtr = makeNode(RangeTblRef);
    rtr->rtindex =  j + 2;
    ourFromExpr->fromlist = lappend(ourFromExpr->fromlist, rtr);
}

createBoolExpr->args = lappend(createBoolExpr->args, createOpExpr1);

createBoolExpr->args = lappend(createBoolExpr->args, createOpExpr2);
```
targetList
The target list is a list of expressions that define the result of the query. In the case of a SELECT, these expressions are the ones that build the final output of the query. They correspond to the expressions between the key words SELECT and FROM.

0.6 Some More Decorrelation Examples

0.6.1 Query with any comparison operator
This is the most basic form correlated query. The correlated query structure would be

select $T_1.a$ from $T_1$
where $T_1.b$ $op$ (select $T_2.b$
from $T_2$ where $T_1.c = T_2.c$ )

Here any comparison operator can appear in place of $op$. To decorrelate this query we have taken the inside condition and anded it with the outside. We have also taken the column name of the inside select clause and applied the operator present in the outside where clause. We also had to add another table in the form clause. So the decorrelated form of the given query will be:

select $T_1.a$ from $T_1$, $T_2$ where $T_1.c = T_2.c$ and $T_1.b$ $op$ $T_2.b$

0.6.2 Query with Aggregate functions
This is a correlated query in which there is an aggregate function in the subquery. This aggregate function may be any function (like SUM, AVG, COUNT, MIN, MAX). The correlated query structure would be

select $T_1.a$ from $T_1$
where CONST $op$(select $\text{AggF}(T_2.b)$
from $T_2$ where $T_1.c = T_2.c$ )
Here `CONST` is a constant and any comparison operator can appear in place of `op`. To decorrelate this query we will create a temporary relation that will specify the condition that is present in outer where clause into its having clause and groupby the result on the attribute of inner relation that is considered in the inner where clause. We are naming this resultant temporary relation as `aggresult`, now we can perform the join of the outer table and `aggresult` on the condition that is present in the inner where clause. So the decorrelated form of the given query will be:

```sql
select T1.a from T1, (select T2.c
from T2 groupby T2.c
having CONST op AggF(T2.b)) as aggresult
where T1.c = T2.c )
```

### 0.6.3 Query with `exists`

This is a correlated query that contains `EXISTS` in the where clause of outer query. The correlated query structure would be

```sql
select T1.a from T1
where exists (select T2.b from T2 where T1.c = T2.c )
```

To decorrelate this query we will simply convert it into join on the condition that is present in the inner where clause, also to insure that those tuple should not be included for which value of inner select result is empty we are including one more condition in where clause that is null test. So the decorrelated query structure would be

```sql
select distinct T1.a from T1, T2
where T1.c = T2.c and
T2.b is NOT NULL
```

This decorrelation involves a new condition in where clause that is null test.
0.7 Future Work

In our project we have laid the foundation for decorrelation in PGSQL. We have considered the most common types of correlated queries in practice today and have decorrelated them for faster processing by PGSQL. More or less all the basic kind of queries have been considered.

Complete implementation of Query Decorrelation in PGSQL is not something that can be done as a course project in a semester. So there remains a lot that can be added to our basic foundation. First of all we have considered only queries that contain only select clause. Other types of queries like update, delete, etc that contains correlated parts can be decorrelated as well by adding a few lines and changing just a few conditional statements in our code.

Though we have considered almost all types of queries, we have left out a few of them like queries that contain not exists. These queries can be taken into consideration in the next phase.

We have only considered queries that contain correlation just up to one level. If there are multiple levels of correlation then our system will fail to decorrelate it. This has to be taken into account when work begins on the next level.
Bibliography

