Deliverable 1 2ID35
Database Technology

C. Lambrechts - 0733885 - c.lambrechts@student.tue.nl
K. Triantos - ?? - k.triantos@student.tue.nl
J. Wulms - 0747580 - j.j.h.m.wulms@student.tue.nl

May 13, 2014

Abstract

This report contains the first deliverable of the project for the course 2ID35 Database Technology. We provide some background for the paper we are studying, work out the research problems that have been addressed and what results have been claimed by the paper. We then proceed by giving an overview of what we are going to do in order to verify the research that has been done by the authors of the paper, and a discussion of the results so far.

1 Introduction

Optimizing a query can be done in several ways. The operation that is the computational most expensive is the join operation. So it makes sense to try to make this operation faster. Changing the order in which joins are performed is a common approach. For this approach cost approximations are computed before actually performing the joins. This approach requires the development of a cost model, an assignment of an estimated cost to each query processing plan and searching in the (huge) search space for the cheapest cost plan. There are also approaches that do not use a cost plans. One of the approaches focusses on minimizing the number of joins instead of using a optimal join order. This approach requires a homomorphism test, which is NP-complete. An other approach focusses on the structural properties of the queries. It will try to find a project-join order that will minimize the size of the intermediate results during query evaluation. Practical it means, use only the data that you need and remove the rest as soon as possible.

The paper we are reviewing develops such a structural approach. The authors of that paper try to push down the projections, such that the attributes that are not needed are projected out as early as possible. The projection pushing strategy has been applied to solve constraint satisfaction problems in Artificial Intelligence with good experimental results. The input to a constraint-satisfaction problem consists of a set of variables, a set of possible values for the variables, and a set of constraints between the variables; the question is to determine whether there is an assignment of values to the variables that satisfies the given constraints. IN the database area, this means that you are searching for entries that satisfy the constraints on their attributes.

Optimizing the query manually, often focusses on reducing the search space before the join operation. In this fashion the number of entries used in the join is less, which can result in a huge performance gain. Most of the time the projections are pushed down by selecting

as soon as possible or pushing the projection to a sub query. It is possible to make a sub-query that projects out all irrelevant information and tries to reduce the intermediate results. Optimizing the query in an automated and structural way looks promising and practical.

2 Problem Description

In this section we describe the problem that is addressed in the paper, but we also state the solution that the authors proposed. We choose to do this, since the solution in the paper is actually part of our own validation problem.

As mentioned in "Project Pushing Revisited" paper, *join* operation is one of the most fundamental and most expensive operations in a database query. The reason is the fact that it combines and uses tuples from multiple relations. In McMahan, Pan, Porter and Vardi research, an attempt is made, which focuses on structural query properties, to find the *project - join* order, which will utilize the size of intermediate results during query evaluation, in the terms of minimization.

In general, almost every database query can be expressed as a *select - project join* query, which combine *joins* with *selections* and *projections*. The main idea of the research is to choose a *project - join* order, which will establish a linear bound on the size of intermediate results. More specifically, it is proven experimentally by the authors that a standard SQL planner spends an exponential amount of time on generating plans for such queries, with rather dismal results in terms of performance and without taking advantage of projection pushing. So the main focus of whole project is to study the scalability of various optimizing methods and to compare the performance of different optimization techniques when the size of the queries is increased.

Below we give a description of the techniques, which were tested during McMahan, Pan, Porter and Vardi research, in order to clarify what will be verified in our own research. The following techniques must be implemented for verification. More specifically, the techniques, which will be tested, are the following:

- Naïve approach
- Straightforward approach
- Projection Pushing and Join Reordering
- Bucket Elimination

These methods will be tested on queries that solve the 3-COLOR graph problem. The goal is to find a label out of a set of three different labels, such that each pair of vertices that have an edge between them, do not have the same label. This is realised by creating a table for each possible edge, which has a record for each possible, valid coloring for these two graphs. This comes down to a table of 6 records, which are all possible combinations of 2 different colors out of the 3 possible ones. If we now try to join all the tables for which there are edges in a graph, we solve the 3-COLOR problem. When the join results in a table without records, there is no possible coloring, but when there are records, the values in the records give a correct coloring/labeling.

The Naïve approach constructs a query where all the tables are listed in the FROM section and in the WHERE section conditions are stated that show which attributes should be equal. The Straightforward approach suggests a specific query structure, which takes advantage of join operation, in order to minimize compile time. In the paper is stated that naive queries are exceedingly difficult to compile and compile time is four orders of magnitude bigger than execution time. In order to get around this ineffectiveness, researchers propose to explicitly list the *joins* in the FROM section of the query, instead of using equalities in the WHERE section as in the naive approach. In this case, the order in which the relations are listed then becomes the order that the database engine evaluates the query. This technique effectively limits what SQL Planner can do and therefore drastically decreases compile time. However, the straightforward approach still does not take advantage of projection pushing. Consequently, it was found that query execution time for the naive and straightforward approaches are essentially identical; the join order chosen by the genetic algorithm is apparently no better than the straightforward order.

Projection Pushing is a step further than Straightforward approach. The idea is to produce an early projection, which would reduce the size of intermediate results by reducing their arity, making further joins less expensive, and thus reducing execution time of the query. Early projection in SQL can be implemented with the use of subqueries. This method processes the relations of the query in a linear fashion, but it can be optimized further. Since the goal of early projection is to project variables as soon as possible, reordering the relations may enable us to project early more aggressively. So, it would be more effective if there is a search at each step for an atom that would result in the maximum number of variables to be projected early. According to Join Reordering, a permutation could be chosen so as to minimize the number of live variables into intermediate relations. Once this permutation is computed, the same SQL query can be constructed as before, but this time with permutation order.

3 Claimed results

In order to verify the paper we are looking into, we need a precise overview of the results that are claimed by the authors of the paper. This way we are able to make clear comparisons between the claimed results and our own.

3.1 Paper results

We first look at the results for the 3-COLOR graphs where order is fixed and the density scales:

- The curves for Boolean and non-Boolean queries have roughly the same shape.
- At first the running time increases as density increases, becaue of an increased number of joins.
- Eventually the size of intermediate results becomes small or empty, and additional joins have little effect on overall running time.

- At low density each optimization method improves upon the previous. For denser instances, optimizations using early projection lose their effectiveness.
- Bucket elimination completely dominates the greedy methods.

Proceeding with the results for 3-COLOR graphs where density is fixed and the order scales. The density is fixed at 2 values, and the authors assume that the lower value is most likely associated with 3-colorable graphs, and the higher density with non-3-colorable graphs:

- All methods show exponential increase in running, when order is increased. (This is shown by a linear slope in logscale.)
- Bucket elimination maintains a lower slope in logscale, in comparison to the other optimazations. This lower slope in logscale translates to a strictly smaller exponent, so we have an exponential improvement.

The next focus of the report was order-scaling experiments with structured queries. We first look into augmented path instances:

- Bucket elimination is again the best, but early projection is competitive for these instances, because the problem has a natural order that works well for early projection.
- For non-Boolean graphs the optimizations do not scale as well as for Boolean graphs. This is due to the fact that there are $\geq 20\%$ less vertices to exploit in the optimization. Early projection and bucket elimination still dominate the other optimizations in this case.

The final claims are about the results for ladder graph instances, and augmented ladder instances:

- For ladder instances, the heuristic for reordering is not only unable to find a better order, but actually finds a worse one.
- Furthermore, ladder instances give results very similar to augmented path instances.
- Augmented ladder instances shows even more differences between optimization methods.
- Non-Boolean cases for augmented ladder instances struggle to reach order 20 with the faster optimizations.

The conclusion for all these results is that bucket elimination dominates the field at every turn with an exponential improvement. In a discussion about future research areas, the authors also claim that they found results consistent with 3-COLOR queries, when using 3-SAT and 2-SAT to construct queries.

3.2 Verification

In our verification, the main claims we want to verify are the domination of bucket elimination and the exponential improvement it shows in the paper. The next step is going into the details of all the different query types (random and structured), and getting consistent results there, or finding out why we have different results. When everything works out as planned, we can further look into queries constructed from other sources than 3-COLOR, such as 3-SAT or 2-SAT.

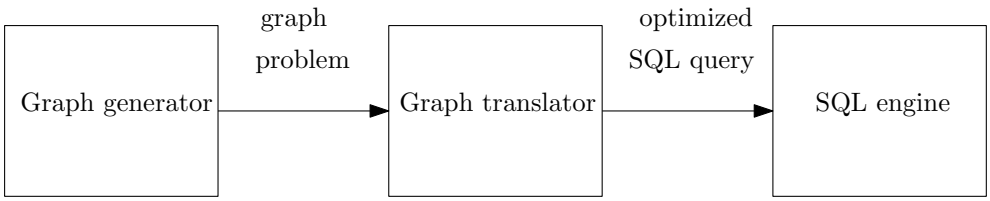

Figure 1: All steps in the process of verification

4 Methodology

This section will elaborate on the steps we are going to take to verify the results in the paper.

The steps we are going to take are as follows:

- Generate graphs similar to the ones used in the paper.
- Implement the proposed algorithms to create optimized SQL queries from the graphs.
- Send query to SQL engine and measure the execution time

Figure 1 gives an overview of the process and how the results are passed. The generation and translation of graphs will be implemented in Java. The idea is to have a graph generator, which outputs the graph we want to solve. Specifically, we pass a list of graphs, which will be the graphs used for a single experiment, varying in graph order or density.

The graph translation part takes the graphs as input and generates SQL queries for the graphs, according to the algorithms proposed in the paper. This part thus returns several SQL queries, in a text-file or on console. This is the biggest part of our assignment, so we want to work in parallel on the different algorithms.

The queries we can pass to an SQL engine. We have chosen to use PostGreSQL, to stay close to the tools that were used in the paper. However, we no longer have access to the older version of PostGre that is used by the authors of the paper, so we choose to use a newer version, namely The newer version can be more optimized, so we have to take this into account when comparing results. Furthermore, we know that the authors used a Linux cluster of Itanium II, processors with 4GB of memory. We have no access to the cluster so we have to use our own machinesThe machine we will use also has 4GB of RAM, but we are unsure whether our setup, using an Intel Core i5 (2,53 GHz), can match the computational power of the cluster.

5 Discussion of progress

Currently, the graph generator is mostly finished. We can generate graphs having scaling graph order or density, while maintaining constant values for the variables we do not want to vary. What remains to be done, is extending the graph generator with an option to generate graphs that have a specific structure, like the augmented and ladder graphs.

The graph translator has been set up in such a way that we can start programming on the different algorithms in parallel. This is important, since it is the biggest part of the implementation of the process.