Deliverable 1 2ID35
Database Technology

C. Lambrechts - 0733885 - c.lambrechts@student.tue.nl
K. Triantos - 0852612 - 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 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 [1] 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.

We start by presenting the problem description found in the paper in Section 2. Then the results claimed in the paper are discussed, Section 3. To verify the results we present our methodology in Section 4. Section 5 discusses the current progress and the current state of the implementation.

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 the "Project Pushing Revisited" paper [1], *join* operation is one of the most fundamental and most expensive operations in a database query. The reason is the fact that this kind of operation combines and uses tuples from multiple relations. In the research of the paper [1], 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 the papers [1] 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-COLOUR 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 colouring for these two graphs. This comes down to a table of 6 records, which are all possible combinations of 2 different colours 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-COLOUR problem. When the join results in a table without records, there is no possible colouring, but when there are records, the values in the records give a correct colouring/labelling. Furthermore, there are multiple types of graphs considered: fully random graphs, augmented path graphs, ladder graphs, graphs combining ladder and augmented properties, and augmented circular ladder graphs.

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 naïve 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 chosen join order is apparently no better than the straightforward order.

Projection Pushing is a step further than the 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 as a result reducing execution time of the query. Early projection in SQL can be implemented with the use of sub-queries. This method processes the relations of the query in a linear fashion. However we can advocate that this technique 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, during each step, there is a search 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.

A more heuristical technique regarding reduction of the arity of intermediate relations is the Bucket Elimination method. This method represents the query as a *join*-expression tree which has set of attributes as nodes, in order to describe an evaluation order to the join operation. More specifically, *join* operations are evaluated from the lower to the higher one and projection operations are applied for this specific evaluation as soon as possible. To clarify how this method works let's take an order of n query attributes, for which we build n buckets, one per attribute. Now we consider these buckets iteratively, from the last to the first and eliminate the latest bucket every time (bottom up method). During these iterations, inside the considering bucket there are several relations, where there is a specific attribute, iconic for this bucket, in all these relations. For all these relations, their join is computed and

attributes are projected out, if they do not belong to the target schema, which we want in the end. The result can be empty, and in that case the result of the whole query is empty. If it is not empty, we place the resulting relation in the highest bucket, which has an attribute in the joined relation that is iconic for the bucket. If this process terminates with there still being a relation in the last bucket, we have found the non-empty result of our query.

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 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.

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.

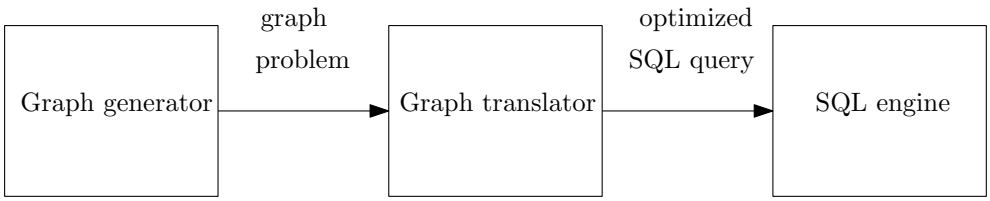

Figure 1: All steps in the process of verification

The queries we can pass to a 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 9.3.4. 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 and no further details about the hardware are available, so we have to use our own machines. The machine we will use also has 4GB of RAM, but we are unsure whether our set-up, 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. We can each implement and test a different algorithm, so we can make progress faster.

We have already installed PostgreSQL, and are looking into how to use it properly. It is not a simple client program, but it assumes that you create a server that holds the database that you want to use/query. We are still looking into the best way of using it with a bigger amount of queries, since we might want to automate the querying process for a set of queries.

References

[1] B. J. Mcmahan, G. Pan, P. Porter, and M. Y. Vardi. Projection pushing revisited. In *In Proc of EDBT04*, pages 441–458, 2004.