CHICAGO: A Test and Evaluation Environment for Coarse-Grained Optimization

Tobias Kraft Holger Schwarz

Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
{Tobias.Kraft, Holger.Schwarz}@informatik.uni-stuttgart.de

Abstract

Relational OLAP tools and other database applications generate sequences of SQL statements that are sent to the database server as result of a single information request issued by a user. Coarse-Grained Optimization is a practical approach for the optimization of such statement sequences based on rewrite rules. In this demonstration we present the CHICAGO test and evaluation environment that allows to assess the effectiveness of rewrite rules and control strategies. It includes a lightweight heuristic optimizer that modifies a given statement sequence using a small and variable set of rewrite rules.

1. Introduction

Many applications, such as information retrieval systems, search engines, and business intelligence tools embed query generators. Some of these applications produce more than one query as a result of an information request that is defined by a user interacting with a graphical user interface. Statement sequences produced by query generators are typically tuned to a certain target database system. However, the response time of such an information request is often far from optimal. One alternative is to rewrite the statement sequence into an equivalent sequence such that the database system consumes far less resources. This is exactly the concept of Coarse-Grained Optimization (CGO). It complements the conventional query optimization phase. The associated rewrite rules exploit the fact that statements of a sequence are correlated. It is possible, e.g., to combine similar statements into a single one, or to move predicates from one statement to a dependent one. An optimizer prototype is introduced in [3].

The CGO approach is not tied to a fixed set of rewrite rules. Several control strategies that determine how these rules are applied to a given statement sequence are possible. Different rulesets and control strategies as well as combinations of them have to be assessed.

Our CGO prototype is based on the results and experience of state-of-the-art optimizer technology as described for example in [2, 5]. Based on few but effective heuristic rewrite rules it produces several SQL statements that are less complex for a single-query optimizer. The ruleset and the control strategies of the CGO optimizer can easily be modified and extended. This allows to flexibly control the optimization process and to identify the appropriate ruleset and control strategy for a given environment.

To further refine the CGO approach, it is crucial to investigate the effectiveness of rewrite rules and control strategies in different settings. The CHICAGO system provides a test and evaluation environment that supports this assessment. The flexibility that is needed is achieved by extending the CGO prototype with additional tools. They allow to define different test scenarios and to measure the effectiveness of several options for combinations of rewrite rules and control strategies in the CGO approach.

In the following section we provide supplementary details regarding the CGO approach. The architecture and the main components of the CHICAGO system are introduced in Section 3. More information on modules that will be demonstrated is provided in Section 4. The paper concludes with remarks on future work.

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

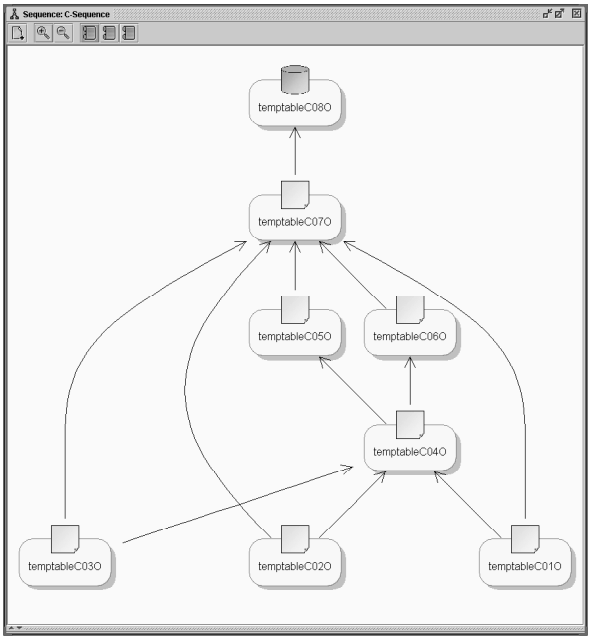

Figure 1: Screenshot of a dependency graph

2. Coarse Grained Optimization

Coarse-grained optimization is a rule-based approach to optimize sequences of SQL statements. The sequences we focus on consist of:

- CREATE TABLE statements that create tables to hold intermediate results or the final result of an information request.
- INSERT statements with a *query* as body that computes an intermediate or the final result by accessing base tables as well as tables storing intermediate results of the sequence. There is exactly one INSERT statement for each table created by a statement sequence.
- DROP TABLE statements that remove tables storing intermediate results.

Such sequences can become very complex. Figure 1 shows a sequence as a dependency graph where each node represents a base table or a statement triple consisting of a CREATE TABLE, INSERT and DROP TABLE statement. This graph shows the direct dependencies and therewith the dataflow between the queries.

We have identified three classes of rewrite rules that can be applied to such statement sequences:

- rules based on similarity among queries,
- rules based on dependencies among queries, and
- rules restricted to a single query.

Figure 2 shows the WhereToGroup rewrite rule being applied to parts of a sample sequence. It belongs to the first class of rewrite rules, because queries *q1* and *q2* only differ in a single predicate of the WHERE clause that compares the same attribute to different constant values. Therefore, these queries can be merged into a new query *q12* that computes and stores the result of both queries. The queries that depend on the result of *q1* and *q2* have to be extended by a predicate that filters out the result of *q1* and *q2*. These optimizations are similar to rewrite rules used in conventional single-query optimization [4] as well as in multi-query optimization [1, 6].

A control strategy is needed to decide on the rewrite rules that should be applied to a given statement sequence. In our prototype we use a priority-based control strategy. The static priority scheme assigns a priority to each rule of the ruleset. The priority assignment is based on experience gained by experiments. If several rules could be applied to the same sequence, the rule of highest priority is picked.

More information on the CGO approach and the optimizer prototype can be found in [3].

```
INSERT INTO q1(a, sum_b)          INSERT INTO q2(a, avg_b)
 SELECT t.a, SUM(s.b)              SELECT t.a, AVG(s.b)
 FROM   s, t                       FROM   s, t
 WHERE  t.c = 1                    WHERE  t.c = 2
   AND  s.d = t.d                    AND  s.d = t.d
 GROUP BY t.a                      GROUP BY t.a

      ↓                                 ↓

INSERT INTO q3(a)                 INSERT INTO q4(a)
 SELECT a                          SELECT a
 FROM   q1                         FROM   q2
 WHERE  sum_b > 100                WHERE  avg_b > 100
```

⇩

```
          INSERT INTO q12(a, c, sum_b, avg_b)
           SELECT t.a, t.c, SUM(s.b), AVG(s.b)
           FROM   s, t
           WHERE  t.c IN (1, 2)
             AND  s.d = t.d
           GROUP BY t.a. t.c

        ↙                                 ↘

INSERT INTO q3(a)                 INSERT INTO q4(a)
 SELECT a                          SELECT a
 FROM   q12                        FROM   q12
 WHERE  sum_b > 100                WHERE  avg_b > 100
   AND  c = 1                        AND  c = 2
```

Figure 2: Example of a rule application

3. System Architecture

Figure 3 shows the overall architecture of the CHICAGO system. A brief description of its components and some implementation issues are provided below.

3.1 Control and Presentation Module

The whole process of generating test cases, optimizing statement sequences and running these sequences on a data warehouse database is controlled by means of a Control and Presentation Module (CPM). Hence, it directly interacts with the CGO Optimizer and the

Sequence Execution Engine. The main features of this module are:

Choose statement sequence: Browsing functionality is provided to the user that allows to view all statement sequences stored in the Test Series Database and choose one of them for optimization. Sequences are visualized as dependency graphs showing the direct dependencies, i.e., the data flow between the queries of a sequence, as shown in Figure 1.

Choose optimizer policy: A set of rewrite rules is provided by the CGO Optimizer. The user may decide to use only a subset of these rules for optimization. The control strategy may also be changed, e.g., by altering the priority assigned to each of the rewrite rules. This allows to adjust and evaluate different optimizer policies.

Control optimization process: The user may leave the entire optimization of a statement sequence to the optimizer. In this case, the series of transformations applied to a statement sequence is shown. Optionally, a single-step mode is available. In this approach only one rewrite rule is applied to the statement sequence. This is useful to explore the benefit of a specific rule in different settings.

Execute statement sequence: As soon as the sequence has been executed, each statement is augmented by its runtime and the cardinality of its result set. This allows the user to identify expensive queries as well as rewrite rules that lead to performance improvements or degradations.

3.2 CGO Optimizer

The CGO Optimizer is the core of the CHICAGO system. It allows to optimize SQL statement sequences by applying rewrite rules. Calls to the optimizer include a statement sequence as well as several parameters that define the ruleset and the control strategy the rule engine of the optimizer should activate. The output of the optimizer is an optimized statement sequence and a list of rewrite rules applied to the original sequence. Optionally, the optimizer can be executed in single-step mode, where only one rule is applied. In this case the output consists of the resulting sequence and a list of all possible rule applications for the next step.

3.3 Sequence Execution Engine

The execution of the statement sequences is controlled by the Sequence Execution Engine. Its input is a configuration that consists of the sequence of SQL statements, name and location of the database that stores the base tables, and other parameters that have to be provided to run the sequence. It provides runtime statistics as the output and stores them in the Test Series Database. These statistics include the runtime for each statement and the cardinality of the temporary tables that are produced by the sequence, as well as errors that occur during execution.

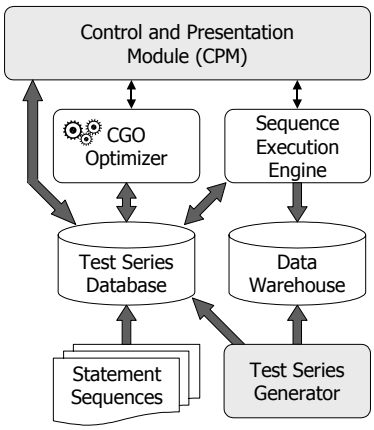

Figure 3: CHICAGO System Architecture

3.4 Test Series Generator

The system offers several options to provide test series. Statement sequences generated by external tools, e.g. commercial OLAP tools, may be provided as well as sequences that are handwritten. Another option is to use the Test Series Generator that is part of the CHICAGO system as well. It provides data sets and statement sequences accessing these data sets. The tool provides a graphical user interface that allows to specify the characteristics of the data sets and the sequence. This serves as a basis for the evaluation of different rewrite strategies with regard to these characteristics.

3.5 Implementation

All components in Figure 3 that are represented as rounded boxes are implemented in Java using Swing for the graphical user interface. Eclipse was chosen as IDE for development of the CHICAGO system.

The SQL statement sequences are parsed and stored as XML documents in the Test Series Database. We use JavaCC as parser generator and DOM to access and manipulate the XML representation of a sequence. XSLT is utilized to retranslate this representation into a SQL statement sequence. The Sequence Execution Engine employs JDBC for access to the data warehouse stored in IBM DB2.

4. Demonstration

The goal of this demonstration is to show the complete evaluation cycle that is supported by the CHICAGO system. This starts with the generation of statement sequences and data sets specifically adjusted to the evaluation purpose. It includes the visualization of the statement sequences as well as the visualization of the optimization process showing how the optimizer works in different settings. Finally, it covers the execution of statement sequences (non-optimized as well as optimized sequences) and shows how the performance gain is

related to the characteristics of rewrite rules, sequences and data.

4.1 Provide Test Sequences and Data

The Test Series Generator includes algorithms to create statement sequences and appropriate data sets. The user may choose from a set of predefined statement patterns. Each pattern corresponds to a single rewrite rule. The main purpose is to provide statement sequences, which allow at least one predefined rewrite step. Additional settings may be used to specify further characteristics of the statement sequence and the underlying data. Hence, the patterns define a subset of interesting test cases for each rule and enable the user to produce test cases for the optimizer in an easy and fast way. This allows to assess the effectiveness of a single rule with respect to the characteristics of statements and data.

Pattern-specific settings for the tables, which are generated, include domain restrictions, value distributions, and indexes on certain attributes. Furthermore, the selectivity of pattern-specific predicates may be defined. There are also some general settings, e.g., the number of columns and the number of rows in the table that has to be generated.

A screenshot of the Test Series Generator is presented in Figure 4. It shows the pattern for the WhereToGroup rule with the INSERT statements of the queries that are produced by the settings. In the demonstration the Test Series Generator will be used to produce simple statement sequences that focus on a special rule, similar to the one in Figure 2.

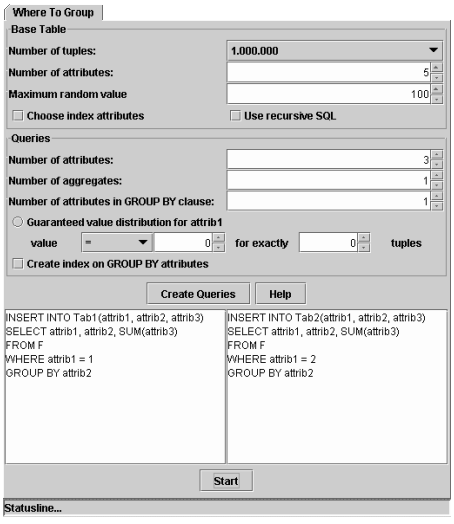

Figure 4: Screenshot of the Test Series Generator

4.2 Load and View Statement Sequences

The Control and Presentation Module offers the basic functionality of loading and viewing statement sequences. It allows to access all sequences in the Test Series Database. This includes statement sequences provided by the Test Series Generator as well as sequences generated by other tools. For each statement sequence that is loaded the corresponding dependency graph is displayed as shown in Figure 1.

4.3 Optimize and Execute Statement Sequences

This part of the demonstration shows how the optimization of statement sequences is managed by the Control and Presentation Module. The single-step optimization mode allows to apply exactly one rewrite rule to the statement sequence, whereas the n-step mode includes several optimization steps according to the control strategy of the CGO Optimizer. For both optimization modes, the set of usable rewrite rules may be restricted.

The Sequence Execution Engine is used to execute statement sequences. In particular, it allows to record the runtime for optimized statement sequences as well as for the corresponding initial sequence.

We will demonstrate the optimization process for a set of statement sequences that includes sequences within a broad complexity range. The effect of different settings will also be demonstrated.

5. Future Work

Planned extensions of the CHICAGO system aim at further exploring and refining the control strategy for CGO. To achieve this, additional control strategies will be provided in the CGO Optimizer. Moreover, the options to control and visualize the optimization process will be enhanced. This includes the navigation in the search space of alternative rewrite processing steps.

References

[1] S. Finkelstein. Common Subexpression Analysis in Database Applications. In Proc. SIGMOD, Orlando, Florida, USA, June 1982.

[2] G. Graefe. The Cascades Framework for Query Optimization. Data Engineering Bulletin, 18(3), 1995.

[3] T. Kraft, H. Schwarz, R. Rantzau, B. Mitschang: Coarse-Grained Optimization: Techniques for Rewriting SQL Statement Sequences. In Proc. VLDB, Berlin, Germany, September 2003.

[4] H. Pirahesh, J. Hellerstein, W. Hasan. Extensible Rule-Based Query Rewrite Optimization in Starburst. In Proc. SIGMOD, San Diego, California, USA, June 1992.

[5] A. Rosenthal, U. Chakravarthy. Anatomy of a Modular Multiple Query Optimizer. In Proc. VLDB, Los Angeles, California, USA, August/September 1988.

[6] T. Sellis. Multiple-Query Optimization. TODS, 13(1), 1988.