Heterogeneous Database Query Optimization in DB2 Universal DataJoiner

Shivakumar Venkataraman
IBM Santa Teresa Labs
San Jose, CA 95141
shiv@us.ibm.com

Tian Zhang
IBM Santa Teresa Labs
San Jose, CA 95141
tzhang@us.ibm.com

Abstract

DataJoiner (DJ) is a heterogeneous database system that provides a single database image of multiple databases. It provides transparent access to tables at remote databases through user defined aliases (nicknames) that can be accessed as if they were local tables. DJ is also a fully functional relational database system. A couple of salient features of the DataJoiner query optimizer are: (1) A query submitted to DataJoiner is optimized using a cost model that takes into account the remote optimizer's capabilities in addition to the remote query processing capabilities and (2) If a remote database system lacks some functionality (eg: sorting), DataJoiner compensates for it. In this paper, we present the design of the Datajoiner query optimizer.

Proceedings of the 24th VLDB Conference
New York, USA, 1998

1 Introduction

DataJoiner (DJ) is a heterogeneous database system [AL90] that provides a single database image of multiple databases. The database clients access remote data through user defined aliases (or nicknames). The presence of remote tables is completely transparent to the user. The Cost based query optimizer in DataJoiner explores the capabilities of the remote optimizer and remote query processor to determine the portions of the query/updates to be executed at the remote databases. DataJoiner uses an optimizer morphing technique that enables the optimizer to model the remote optimizer characteristics (left deep plan vs bushy plans) in addition to the remote query processing capabilities (joins, selects, subqueries, nested table expressions, common subexpressions) to determine the globally optimal plan. This enables DataJoiner to generate global plans that closely reflect the actual plans generated by the remote database systems when segments of the queries are shipped to the remote database for execution.

Figure 1 illustrates the architecture of DataJoiner. DataJoiner is based on the DB2/CS code base, and is a fully functional relational database system supporting access to data at heterogeneous databases. Data-

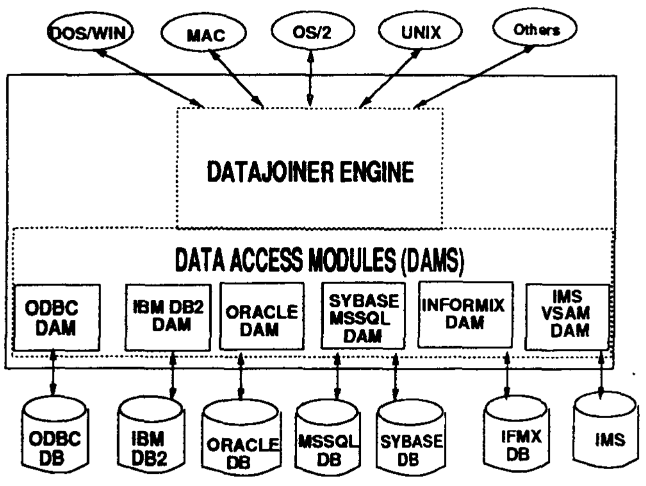

Figure 1: DataJoiner Architecture

Joiner communicates with remote databases through Data Access Modules (DAM). Each DAM understands the SQL dialect and communication protocol of the remote database that it communicates with. Figure 1 shows the remote databases supported by DataJoiner. Associated with each DAM is a Server Attributes Table (SAT). The SAT records important information about the remote database system, specifically about its remote query optimizer and its SQL dialect. This information is used by the DataJoiner optimizer to model the remote query optimizer characteristics. Changes to the values in the SAT can be accomplished through data definition language (DDL) statements. This provides a powerful mechanism by which global plans produced by DataJoiner can be tuned, without having to make changes to the code. For example, if a database were to come out with support for merge join, when it previously did not support it, DDL statements can be used to inform DJ about this feature, and the optimizer will take this information into account while generating a global plan. The following are some of the salient features of DataJoiner (DJ):

- Single database image of multiple databases
- Global catalog
- Local storage of data
- Cost-based optimization of heterogeneous database queries
- Modeling remote query optimizer characteristics
- Insert, Update, and Delete to remote databases
- DDL transparency
- Heterogeneous multi-site updates with two-phase commit
- Global stored procedures

2 Query Optimizer

Most relational database systems that DataJoiner supports access to have cost-based query optimizers. However, these databases differ significantly in their query processing and optimization capabilities. For example, while most databases support some form of joins, some databases only support nested loop join while others support both merge and nested-loop join. Some query optimizers consider a left-deep plan space, while others consider a bushy plan space. We take these differences into account while generating global query plans in DataJoiner. It is important for DataJoiner to model the capabilities of the remote optimizer, since the global plan generated mostly depends on how the remote query optimizer plans the SQL query sent to it.

Earlier work on heterogeneous query optimization have studied how to model the costs of remote plans [DKS92]. Recently, Haas et al. [HKWY97] and Tork-Roth et al. [TRS97] have studied the problem of modeling the query processing capabilities of non-relational data sources. In DataJoiner, in addition to modeling the capabilities of the remote database, we provide a framework for modeling the remote query optimizer capabilities. A comparative study published by Rezende et al [RH98] shows the performance of of gateway products.

The query optimizer in DataJoiner is based on the DB2/CS Starburst optimizer [PHH92] [GLS93] technology. The relational query optimizer of DB2/CS is extended to model the remote query optimizers. The information in the SAT is used to parametrically model the remote query optimizers. The intention of using the SAT is to enable DataJoiner to generate query plans that closely reflect the plans that the remote query optimizers generate. Accurate statistics for tables at remote database is maintained in the local DataJoiner system catalogs. Catalog refreshes is either done by fetching the information from the catalogs of the remote database, or by using the runstats utility to scan the remote table, to generate the detailed statistics used by the DB2 query optimizer.

Figure 2 shows the six stages of the DataJoiner query optimizer. The non-shaded boxes correspond to the stages in the base DB2 query optimizer. The shaded stages correspond to the new stages introduced by DataJoiner. Each stage is described in detail here:

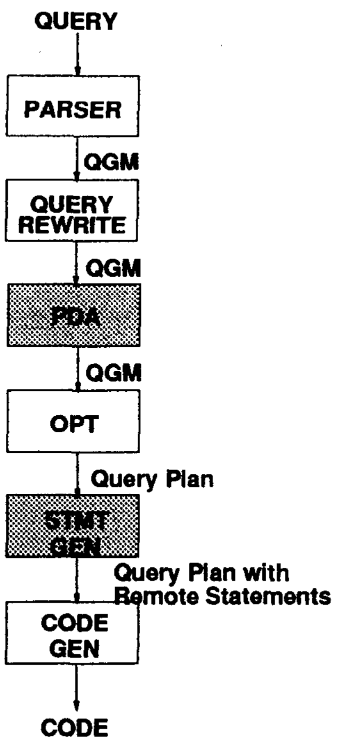

Figure 2: Query Optimizer

- **Parser:**The query from the user is parsed, semantically analyzed, and translated into an internal data structure that reflects the query. This data structure is referred to as the Query Graph Model (QGM). The QGM semantically represents the query.
- **Pushdown Analysis (PDA):** Pushdown analysis is used to determine as well as maximize the portions of the query that can be evaluated by the remote databases. By using the knowledge of the remote databases, PDA prepares the QGM for the optimizer with useful information about the portions of the query that can be evaluated at the remote databases, however it still leaves the decision of whether the evaluation should take place remotely to the DJ query optimizer.
- **Query Rewrite:** The QGM is transformed by a set of heuristic rewrite rules [PHH92] to enable the optimizer to generate better plans. Since the capabilities of the remote databases are different, the rewrite rules have to be sensitive to the capabilities of the remote databases so that we can send a significant portion of the query to the remote database. For example, scalar subquery to join rewrite rule may transform a query containing a scalar subquery into a join query with a

nested table expression. The original query usually can be sent in its entirety to the remote database because most relational databases support scalar subqueries, whereas the transformed query might not be processed completely at the remote database because some databases do not support nested table expression. In order to avoid this, we use the capabilities dictionary of the remote database stored in the SAT to conditionally invoke the query rewrite rules. In addition, we also added some new rewrite rules that are targeted towards reducing the communication costs in a heterogeneous environment. For example, UNION operands are grouped by their data sources so that instead of opening a connection for each UNION operand, we only open a connection for each group of UNION operands. This rule, while reducing the number of requests, does not limit the optimizer's capabilities in generating the best global plan.

- **Query Optimizer:** The optimizer uses the QGM as input to generate query plans. It uses the PDA markings to determine if a remote plan can be generated. If a query block is pushdownable to a remote database, both, remote and local plans are generated. The SAT table is used by the optimizer to determine the capabilities of the remote database. For example: if the property in the SAT indicates that a remote database does not support merge joins then the DataJoiner optimizer will not consider merge joins for the remote plan. In addition to the query processing capabilities, the optimizer models the characteristics of the remote database optimizer. An example is the plan space search. Some optimizers may generate plans containing bushy joins, while others only generate plans with left-deep joins. DataJoiner's optimizer is sensitive to the abilities of the remote database optimizer and generates a global plan accordingly. The query optimizer computes the cost of the various plans based on the statistics maintained in the catalogs. The optimizer's cost model takes into account the communication cost, the difference between the local and the remote CPU and I/O speeds.

- **Statement Generation:** The best plan generated by the query optimizer is then fed to the statement generator. The statement generator, traverses the portion of the query plan that needs to be evaluated remotely and generates SQL statements for these portions.

- **Code Generation:** The best plan is then converted into executable code.

The code generated is interpreted by the runtime engine during query execution. The SQL statements generated are sent to the remote databases for execution. The query results obtained are combined and returned to the user.

3 Performance

Preliminary performance results with the industry standard benchmarks indicates that queries executed through DataJoiner performs extremely well, since the global plan produced by DataJoiner contains subplans that closely reflect the plan produced by the remote optimizer. In many instances, queries directed to a single database system sent through DataJoiner performed much better than queries executed on the remote database system directly. The aggregate timing measurements on the seventeen queries of the industry standard benchmark executed through DataJoiner on database systems of commercial database vendors showed that the execution times that were either comparable or significantly faster than the queries that were executed directly at the remote database system. shows the superiority of DataJoiner query optimizer over those of other gateway products.

4 Conclusions

The DataJoiner query optimizer models the query optimizers of the remote database systems. It recognizes the sensitivity of the remote database optimizers to the query syntax. Knobs provided in the form of catalog updates in DataJoiner enables one to easily model changes to the remote database system capabilities, without requiring changes to the code. This enables Datajoiner to generate query plans that closely reflect the actual plan that get generated at the remote databases during query execution. This enables DataJoiner to generate very efficient plans and this in turn leads to high performance. DataJoiner version 2.1 is now generally available and more information on DataJoiner is available from the DataJoiner web site at:**www.software.ibm.com/data/datajoiner**

References

[AL90] Sheth A.P and J.A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, March 1990.

[DKS92] W Du, R Krishnamurthy, and M.C Shan. Query Optimization in a Heterogeneous DBMS. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 277–291, 1992.

[GLS93] P. Gassner, G.M. Lohman, and Y Schiefer, B. Wang. Query Optimization in the IBM DB2 Family. *Data Engineering Bulletin*, 16(4), 1993.

[HKWY97] L.M. Haas, D. Kossmann, E.L. Wimmers, and J. Yang. Optimizing Queries Across Diverse Data Sources. In *Proceeding of the VLDB Conference*, Aug 1997.

[PHH92] H Pirahesh, J.M Hellerstein, and W. Hassan. Extensible/Rule Based Query Rewrite Optimization in Starburst. In *Proceedings of the ACM SIGMOD Conference*, pages 39–48, 1992.

[RH98] Rezende, F. and Hergula, K. The heterogenity problem and middleware technology: Experiences with and performance of database gateways. In *Proc. of the Intl Conf on Very Large Databases (VLDB)*, New York, 1998.

[TRS97] M. Tork-Roth and P. Schwarz. Dont Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources . In *Proceeding of the VLDB Conference*, Aug 1997.