Holistic Optimization of Database Applications

Karthik Ramachandra

March 11, 2011

Advisor: Prof. S Sudarshan

Database Applications today

With the advent of the Internet, in the last decade, database applications have acquired a new meaning. Internet applications generate huge amounts of data of different kinds, and generate huge numbers of requests against that data. Adding to this, the recent trends of multi-core processors and cloud computing, have pushed concurrent and distributed programming to the mainstream. A lot of traditional database and programming techniques were not designed to deal with these data management challenges. Therefore, there is a clear need for new data models, query processing models, and programming models that (a) aid the programmer by providing the right high level abstractions, (b) ensure that the underlying mechanism is able to perform smart optimizations, and (c) ensure high scalability.

Traditional Query and Program Optimization

Database query optimization and program optimization techniques have evolved independently over a long time thanks to the extensive research that has been happening in these areas. Though there are still many hard unsolved problems in those areas, we can safely claim that the techniques that have evolved are mature and robust. However, in practice, we see that there is a large class of 'database applications' i.e., applications which operate and maintain data from one or more data sources such as relational databases, web services etc. Applications written in various programming languages access such data using SQL, JSON or other query languages as supported by the data source. These applications are built in a modularized manner, and various parts of the application logic are placed in different modules and are written in different languages. For example, consider a web application which interacts with a relational database. Here, all the data access logic is written in SQL (SQL queries, PL/SQL procedures, UDFs etc.), and they run only on the database system. The application logic which operates on this data, is written in a general purpose language such as Java, and runs on the application server. The presentation logic is written in Javascript, and runs on the browser. This separation is done for reasons such as (a) Maintainability and readability of the source code, (b) Security of data and source code, (c) Expressive power of languages (eg. SQL is suitable for expressing data access queries, Javascript for event handling and so on).

The typical pattern of data access for such applications is as follows: (a) Dynamically construct and manipulate queries (in the target query language) using the application programming language, (b) Submit the queries to the database for execution (typically over the network), (c) Subsequently,

map the results of the query from the database to objects of the programming language for further usage. This results in complex interactions between these systems, and can lead to a large number of correctness, security and performance issues which can go unnoticed during development time. Traditionally, many of the attempts that have been made to identify such issues have been in two directions: (a) DBMS centric - involving query optimizations, caching, constraint enforcements etc., or (b) Application centric - involving source code optimizations, data caching, data authorization, validation, etc. Optimization of such applications is typically done using the above independent techniques.

Example 1 *A Program Snippet with JDBC Calls [4]*

```
Connection con = DriverManager.getConnection(url);
PreparedStatement pstmt = con.prepare(
    "SELECT count(partkey) FROM part WHERE category=?");
while(category != -1) {
    pstmt.setInt(1, category);
    ResultSet rs = pstmt.executeQuery();
    if (rs.next()) {
        partCount = rs.getInt(0);
        total += partCount;
    }
    category = getParent(category);
}
```

Holistic Optimization

We argue that a lot of correctness, performance, and security issues arise due to the complexity and overhead in the interactions, and not just due to problems within a system. This can be illustrated with an example. Consider the Java program snippet, which interacts with a relational database, shown in Example 1 [4]. The program computes the total number of parts in a given category and all its parent categories. Note the repeated execution of a parameterized aggregate query inside the *while* loop. This results in a lot of random IO, and network round trips. This can be optimized to use set oriented query execution which saves on both disk IO as well as network round trips. An equivalent program that uses set oriented execution is shown in Example 2.

It can be observed from Example 2 that automatically performing such an optimization requires a rewriting of both the query as well as the program. It can neither be conceived of as a purely query optimization technique nor as a purely program optimization technique. Therefore, in order to achieve optimal, correct and safe execution of the application as a whole, there have to be approaches which consider the application as a unit. In essence, this is what we mean by a *holistic approach*. The boundaries still exist in terms of the source code as well as deployment, but these holistic optimization and robustness approaches span across these separations.

Let us consider another scenario. Synchronous execution of queries or Web service requests forces the calling application to block until the query/request is satisfied. The performance of applications can be significantly improved by asynchronous submission of queries, which allows the application

Example 2 *JDBC Example with set oriented query execution [4]*

```
Connection con = DriverManager.getConnection(dbrUrl);
PreparedStatement pstmt = con.prepare(
  "SELECT count(partkey) FROM part WHERE category=?");
while(category != -1) {
    pstmt.setInt(1, category);
    pstmt.addBatch();
    category = getParent(category);
}
// SELECT pb.category, count(partkey)
// FROM pbatch pb LEFT OUTER JOIN part p
// ON pb.category=p.category GROUP BY pb.category"
pstmt.executeBatch();
while (pstmt.getMoreResults()) {
    ResultSet rs = pstmt.getResultSet();
    if (rs.next()) {
        partCount = rs.getInt(0);
        total += partCount;
    }
}
```

to perform other processing instead of blocking while the query is executed, and to concurrently issue multiple queries. Concurrent submission of multiple queries can allow the query execution engine to better utilize multiple processors and disks, and to reorder disk IO requests to minimize seeks. Concurrent submission also reduces the impact of network round-trip latency and delays at the database, when processing multiple queries. This overlapping of query executions improves the efficiency of the interactions between the application and the database and hence is a holistic optimization technique. However, manually writing applications to exploit asynchronous query submission is tedious. We address the issue of automatically transforming a program written assuming synchronous query submission, to one that exploits asynchronous query submission [3]. Our program transformation method combines the data-flow analysis of the program with the additional knowledge about the query being submitted to the database. We have built a tool that implements our transformation techniques on Java code that uses JDBC calls [2]; our tool can be extended to handle Web service calls.

Holistic approaches therefore, exploit the knowledge of the application program source code, the interaction patterns between the application and the database, the meta-data of the database, and maybe even traces and other runtime information, and try to make sure that the interactions between these two systems are optimal, correct, and secure. This is primarily done by applying source code transformations and query transformations which guarantee equivalence with the previous state while fixing the identified issues. Holistic approaches can be entirely static, i.e. those that operate on the source code as mentioned above, or they can have a runtime component as well, which speculatively performs safe optimizations.

Conclusion and Future work

Though there has been some earlier work in performing such optimizations that span across system boundaries, these are still far from being applied in real world internet scale database applications. This is due to the fact that these are inherently hard problems that hide beneath the surface, and involves combining and extending the ideas from the extensive research work done both in the areas of traditional query optimization as well as in the area of compiler design and program analysis. By observing the trends in terms of the research work, we could possibly claim that the area of query optimization seems to be naturally evolving into considering factors beyond the database.

The main reason for this is probably the exponential growth of large scale distributed asynchronous systems with heterogeneous constraints. In order to effectively build software to handle the scale and the complexity, trends also show the increase in tendency towards declarative styles of programming. The Claremont report on database research [1] highlights this as a key research area where work needs to be done. We conclude by quoting from the report:

> "Although developing new programming paradigms is not a database problem per se, ideas of **data independence**, **declarative programming** and **cost-based optimization** provide a promising angle of attack. There is significant evidence that data-centric approaches can have major impact on programming in the near term."
>
> "For enterprise applications, a key distributed design decision is the partitioning of logic and data across multiple tiers: web clients, web servers, application servers, and a back-end DBMS. Data independence is particularly valuable here, to allow programs to be specified without making a priori, permanent decisions about physical deployment across tiers. **Automatic optimization** processes could make these decisions, and move data and code as needed to achieve efficiency and correctness."
>
> "Among the research questions we face are language design, efficient compilers and runtimes, and techniques to **optimize code automatically across both the horizontal distribution of parallel processors, and the vertical distribution of tiers**. It seems natural that the techniques behind parallel and distributed databases partitioned data-flow, cost-based query optimization should extend to new environments."

References

[1] R. Agrawal, A. Ailamaki, P. A. Bernstein, E. A. Brewer, M. J. Carey, S. Chaudhuri, A. Doan, D. Florescu, M. J. Franklin, H. Garcia-Molina, J. Gehrke, L. Gruenwald, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. F. Korth, D. Kossmann, S. Madden, R. Magoulas, B. C. Ooi, T. O'Reilly, R. Ramakrishnan, S. Sarawagi, M. Stonebraker, A. S. Szalay, and G. Weikum. The claremont report on database research. *Commun. ACM*, 52(6):56–65, 2009.

[2] M. Chavan, R. Guravannavar, K. Ramachandra, and S. Sudarshan. Dbridge: A program rewrite tool for set-oriented query execution. In *IEEE Intl. Conf. on Data Engineering (to appear)*, 2011.

[3] M. Chavan, R. Guravannavar, K. Ramachandra, and S. Sudarshan. Program transformations for asynchronous query submission. In *IEEE Intl. Conf. on Data Engineering (to appear)*, 2011.

[4] R. Guravannavar and S. Sudarshan. Rewriting Procedures for Batched Bindings. In *Intl. Conf. on Very Large Databases*, 2008.