An Adaptive Probe-Based Technique to Optimize Join Queries in Distributed Internet Databases

LATIFUR KHAN
University of Texas at Dallas, USA

DENNIS MCLEOD, AND CYRUS SHAHABI
University of Southern California, Los Angeles, USA

An adaptive probe-based optimization technique is developed and demonstrated in the context of an Internet-based distributed database environment. More and more common are database systems, which are distributed across servers communicating via the Internet where a query at a given site might require data from remote sites. Optimizing the response time of such queries is a challenging task due to the unpredictability of server performance and network traffic at the time of data shipment; this may result in the selection of an expensive query plan using a static query optimizer. We constructed an experimental setup consisting of two servers running the same DBMS connected via the Internet. Concentrating on join queries, we demonstrate how a static query optimizer might choose an expensive plan by mistake. This is due to the lack of a priori knowledge of the run-time environment, inaccurate statistical assumptions in size estimation, and neglecting the cost of remote method invocation. These shortcomings are addressed collectively by proposing a probing mechanism. Furthermore, we extend our mechanism with an adaptive technique that detects sub-optimality of a plan during query execution and attempts to switch to the cheapest plan while avoiding redundant work and imposing little overhead. An implementation of our run-time optimization technique for join queries was constructed in the Java language and incorporated into an experimental setup. The results demonstrate the superiority of our probe-based optimization over a static optimization.

A distributed database is a collection of partially independent databases that share a common schema, and coordinates processing of non-local transactions. Processors communicate with one another through a communication network (Silberschatz, Korth, and Sudarshan, 1997; Yu and Meng, 1998). We focus on distributed database systems with sites running homogeneous software (i.e., database management system, DBMS) on heterogeneous hardware (e.g., PC and Unix workstations) connected via the Internet. The Internet databases are appropriate for organizations consisting of a number of almost independent sub-organizations such as a University with many departments or a bank with many branches. The idea is to partition data across multiple geographically or administratively distributed sites where each site runs an almost autonomous database system.

In a distributed database system, some queries require the participation of multiple sites, each processing part of the query as well as transferring data back and forth among themselves. Since usually there is more than one plan to execute such a query, it is crucial to obtain the cost of each plan, which highly depends on the amount of participation by each site as well as the amount of data shipment between the sites. Assuming a private/dedicated network and servers, this cost can be computed \textit{a priori} due to the predictability of servers and network conditions and availability of effective network bandwidth. However, in the Internet environment, which is based on a best effort service, there are a number of unpredictable factors that make the cost computation complicated (Paxson and Floyd, 1997). A static query optimizer that does not consider the characteristics of the environment or only considers the a priori knowledge on the run-time parameters might end up choosing expensive plans due to these unpredictable factors. In the following paragraph, we explain some of these factors via simple examples.
Participating sites (or servers) of Internet database systems might have different processing powers. One site might be a high-end multiprocessor system while the other is a low-end PC running (say) Windows NT. In addition, since most queries are I/O intensive, a site having faster disk drives might observe a better performance. In an Internet-based environment these sites might be dedicated to a single application or multiple simultaneous applications. For example, one site might only run a database server while the other is a database server, a web server, and an e-mail server. Moreover, the workload on each server might vary over time. A server running overnight backup processes is more loaded at night as compared to a server running 8 a.m.-5 p.m. office transactions. Due to time differences, a server in New York might receive more queries at 5 a.m. in pacific standard time as compared to those received by a server in Los Angeles. The network traffic is another major factor. It is not easy to predict network delay in the Internet due to variability of effective network bandwidth among the sites. A query plan which results in less tuple shipments might or might not be superior to the one preferring extensive local processing, depending on the network traffic and server load at the time of query processing. Briefly, there is just too much uncertainty and a very dynamic behavior in an Internet-based environment that makes the cost estimation of a plan a very sophisticated task.

Although we believe our probe-based run-time optimization technique is applicable to multi-databases with sites running heterogeneous DBMS, we do not consider such a complex environment in order to focus on the query processing and optimization issues. There has been an extensive research in query processing and optimization in both distributed databases and multi-databases (Amsaleg, Bonnet, Franklin, Tomasic, and Urban, 1997; Apers, Hevner, and Ya, 1983; Bernstein, Goodman, Wong, Reeve, and Rothnie, 1981; Bodorik, Riordo, and Jacob, 1989; Bodorik, Riordo, and Pyra, 1992; Chen and Yu, 1992; Evrendile, Dogac, Nural, and Ozcan, 1997; Kambayaashi, Yoshikawa, and Yajima, 1983; Roussopoulos and Kang, 1991; Zhu and Larson, 1994). Among those, only a few considered run-time parameters in their optimizers. We distinguish these studies from ours in Related Work section. Briefly, most of these studies propose a detection approach to compensate for lack of run-time information while our approach is first predictive and prevents the selection of expensive queries at run-time and then becomes adaptive to adapt itself with run-time variations.

This paper demonstrates the importance and effectiveness of an adaptive probe-based optimization technique for join queries in the Internet databases. We focused on join queries because join operation is not only frequently used but also expensive (Yu and Meng, 1998).

In order to demonstrate the importance of run-time optimization, we implemented an experimental distributed database system connected through the Internet. Our setup consists of two identical servers both running the same object-relational DBMS (i.e., Informix Universal Server, (Informix, 1997)) connected via the Internet. We then split the BUCKY database (from the BUCKY benchmark (Carey et al., 1997)) across the two sites. We implemented a probe-based run-time optimization module for join queries in Java language. The optimizer first issues two probe queries each striving to estimate the cost of either semi-join or simple join plans. Consequently, the cheapest plan will be selected. The query optimizer of a distributed database system can be extended with our probe queries to capture run-time behavior of the environment. Furthermore, as a byproduct, the result of the probe queries can be utilized for estimating the size of intermediate relations in a join plan. This estimation is shown to be less sensitive to statistical anomalies as compared to that of static optimizers. Finally, the probe-based technique identified some hidden costs (e.g., the cost of remote invocation of methods with RMI) that should be considered in order to select the cheapest plan. That is, our probing mechanism can capture any surprises associated with specific implementations (e.g., RMI in our case) which can never be accounted for by static optimizers. The experiments show that for expensive queries processing many tuples the response time can be improved on the average by 32.5% over a static optimizer while the probing overhead only results in an average of 6.4% increase in response time. We also discuss an enhanced version of our optimizer, which reduces the overhead by an average of 45% (i.e., observing 3.5% increase in response time due to overhead) by utilizing the results of the probe queries. Obviously, these numbers depend on the number of tuples sampled by the probe queries and the size of relations.

In addition, we propose an adaptive optimization technique that copes with sudden changes of run-time environment on the fly during the execution of query. However, we show that adaptive optimization incurs either no overhead or a little overhead (only in a few cases).

The remainder of this paper is organized as follows. Related Work section covers some related work on query processing and optimization in both distributed databases and multi-databases. Run-Time Optimization for Join Queries section states the problem, reviews a conventional solution, and finally explains our proposed extensions to capture run-time parameters and utilize them to improve the optimizer. Performance Evaluation section consists of a performance study to compare the performance of our run-time optimization technique with that of a static optimizer. Finally, Conclusions and Future Directions section concludes the paper and provides an overview on our future plans.

**RELATED WORK**

There have been various studies on query processing and optimization in distributed, federated, and multidatabase systems (Apers, Hevner, and Ya., 1983; Bernstein et al. 1981; Bodorik, Pyra, and Riordo, 1990; Chen and Yu, 1992; Kambayaashi, Yoshikawa, and Yajima, 1983; Roussopoulos et al. 1981; Bernstein, Goodman, Wong, Reeve, and Rothnie, 1981; Bodorik, Pyra, and Riordo, 1990; Chen and Yu, 1992; Kambayaashi, Yoshikawa, and Yajima, 1983; Roussopoulos, Evrendile, Dogac, Nural, and Ozcan, 1997; Kambayaashi, Yoshikawa, and Yajima, 1983; Roussopoulos and Kang, 1991; Zhu and Larson, 1994). Among those, only a few considered run-time parameters in their optimizers. We distinguish these studies from ours in Related Work section. Briefly, most of these studies propose a detection approach to compensate for lack of run-time information while our approach is first predictive and prevents the selection of expensive queries at run-time and then becomes adaptive to adapt itself with run-time variations. In this paper, we demonstrate the importance and effectiveness of an adaptive probe-based optimization technique for join queries in the Internet databases. We focused on join queries because join operation is not only frequently used but also expensive (Yu and Meng, 1998).

In order to demonstrate the importance of run-time optimization, we implemented an experimental distributed database system connected through the Internet. Our setup consists of two identical servers both running the same object-relational DBMS (i.e., Informix Universal Server, (Informix, 1997)) connected via the Internet. We then split the BUCKY database (from the BUCKY benchmark (Carey et al., 1997)) across the two sites. We implemented a probe-based run-time optimization module for join queries in Java language. The optimizer first issues two probe queries each striving to estimate the cost of either semi-join or simple join plans. Consequently, the cheapest plan will be selected. The query optimizer of a distributed database system can be extended with our probe queries to capture run-time behavior of the environment. Furthermore, as a byproduct, the result of the probe queries can be utilized for estimating the size of intermediate relations in a join plan. This estimation is shown to be less sensitive to statistical anomalies as compared to that of static optimizers. Finally, the probe-based technique identified some hidden costs (e.g., the cost of remote invocation of methods with RMI) that should be considered in order to select the cheapest plan. That is, our probing mechanism can capture any surprises associated with specific implementations (e.g., RMI in our case) which can never be accounted for by static optimizers. The experiments show that for expensive queries processing many tuples the response time can be improved on the average by 32.5% over a static optimizer while the probing overhead only results in an average of 6.4% increase in response time. We also discuss an enhanced version of our optimizer, which reduces the overhead by an average of 45% (i.e., observing 3.5% increase in response time due to overhead) by utilizing the results of the probe queries. Obviously, these numbers depend on the number of tuples sampled by the probe queries and the size of relations.

In addition, we propose an adaptive optimization technique that copes with sudden changes of run-time environment on the fly during the execution of query. However, we show that adaptive optimization incurs either no overhead or a little overhead (only in a few cases).

The remainder of this paper is organized as follows. Related Work section covers some related work on query processing and optimization in both distributed databases and multi-databases. Run-Time Optimization for Join Queries section states the problem, reviews a conventional solution, and finally explains our proposed extensions to capture run-time parameters and utilize them to improve the optimizer. Performance Evaluation section consists of a performance study to compare the performance of our run-time optimization technique with that of a static optimizer. Finally, Conclusions and Future Directions section concludes the paper and provides an overview on our future plans.