Analysis of Query Evaluation Techniques for Large Databases

HIMANSHU VERMA
Research Scholar, MBU
Presently working as Asstt. Prof. Computer Science at IIPM Ghaziabad

Abstract

A "complex" query is one that requires a number of query processing algorithms to work together, and a "large" database uses files with sizes from several megabytes to many terabytes, which are typical for database applications at present and in the near future.

Introduction

Database management systems will continue to manage large data volumes. Thus, efficient algorithms for accessing and manipulating large sets and sequences will be required to provide acceptable performance. This survey provides a foundation for the design and implementation of query execution facilities in new database management systems. It describes a wide array of practical query evaluation techniques for both relational and post-relational database systems, including iterative execution of complex query evaluation plans, the duality of sort- and hash-based set matching

algorithms, types of parallel query execution and their implementation, and special operators for emerging database application domains.

Literature Review

While database management systems are standard tools in business data processing, they are only slowly being introduced to all the other emerging database application areas. In most of these new application domains, database management systems have traditionally not been used for two reasons. First, restrictive data definition and manipulation languages can make application development and maintenance unbearably cumbersome. Second, data volumes might be so large or complex that the real or perceived performance advantage of file systems is considered more important than all other criteria, e.g., the higher levels of abstraction and programmer productivity typically achieved with database management systems. The purpose of this paper is to survey efficient algorithms and software architectures of database query execution engines for executing complex queries over large databases.

User Interface
Database Query Language
Query Optimizer
Query Execution Engine
Files and Indices
I/O Buffer Disk

Figure 1. Query Processing in a Database System.

As shown in Figure 1, query processing fills the gap between database query languages and file systems. It can be divided into query optimization and query execution. A query optimizer translates a query expressed in a high-level query language into a sequence of operations that are implemented in the query execution engine or the file system. The query execution engine is a collection of query execution operators and mechanisms for operator communication and synchronization — it employs concepts from algorithm design, operating systems, networks, and parallel and distributed computation. After a query or request has been entered into the database system, be it interactively or by an application program, the query is parsed into an internal form. Next, the query

is validated against the meta-data to ensure that the query contains only valid references to existing database objects.

Parsing
Query Validation
View Resolution
Optimization
Plan Compilation
Execution

Figure 2. Query Processing Steps.

The query optimizer then maps the expanded query expression into an optimized plan that operates directly on the stored database objects. The optimizer's output is called a query execution plan, query evaluation plan, QEP, or simply plan. Query processing focuses on extracting information from a large amount of data without actually changing the database.

1. Need

The larger amount of data you store, the longer it takes to access, and the heavier the load on the machine. This couples with the larger number of people trying to access the system at one time causes very poor access times. By splitting the data into logical

sections and spreading it around several machines, you only need a small amount of indexing data to point people at the appropriate servers for their requirements.

2. Scope

The SCOPE project has been developing and deploying distributed systems software and services that enable the sharing of large numbers of heterogeneous computing resources across the Internet. These services enable cooperating organizations to help one another in supplementing their computing capacities. In addition, these services enable individual users and organizations to purchase computing cycles from service providers on an as-needed basis, when the frequency of such demands does not justify investment in permanent capacity. SCOPE provides a comprehensive environment for sharing computing resources by integrating the elements of **resource allocation** with **resource discovery**, **resource selection** based on assurance credentials, **distributed authorization**, and online **payments**.

Resource Discovery

PRM allows users to execute applications on processors managed by the users organization, or by a small set of organizations known by the user. To enable the job manager to find resources in a large, geographically and administratively distributed system, mechanisms are needed that will allow the job manager to discover appropriate resources. This is the "resource discovery" problem applied to processing resources and the same techniques used for information discovery can be applied.

Authorization and Payments

When computing resources must be made accessible across organizational boundaries, distributed authorization is used to restrict resource access to particular users. To obtain access to resources, the job manager present authorization credentials to the system manager, proving the user's membership in a group listed in the service provider's authorization database. In this case, a distributed accounting server issues credentials certifying that the client can pay for the service, and can provide payment to the service provider when the service is complete.

Objectives

Transparencies. Distributed database systems are to provide the image to the users of a centralized database system through a series of transparencies that shield the user of such a system from the complex details of providing the transparencies. There are listed seven transparencies that are considered to encapsulate the desired functions of a distributed database system: Location Transparency, Performance Transparency, Copy Transparency, Transaction Transparency, Fragment Transparency, Schema Change Transparency, and Local DBMS Transparency.

Free object naming.

Free object naming means that it allows different users the ability to access the same object with different names, or different objects with the same (internal) name. The tradeoff here is to allow the users as much freedom as possible in naming data objects, yet still allow for the sharing of data without naming conflicts. This kind of strategy can make scale up of distributed databases less of a burden.

Concurrency Control.

Concurrency control is the activity of coordinating concurrent accesses to a database in a multi-user database management system (DBMS). Concurrency control permits users to access a database in a multiprogrammed fashion while preserving the illusion that each user is executing alone on a dedicated system. However, the sub algorithms used by all practical DDBMS concurrency control algorithms are variations of just four basic techniques: two-phase locking, timestamp ordering, multiversion timestamp ordering and optimistic concurrency control.

Several problems can occur when concurrent transaction execute in an uncontrolled manner:

1. The lost update problem. This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.
2. The temporary update problem. This occurs when one transaction updates a database item and then the transaction fails for some reason. The updated items are accessed by

another transaction before it is changed back to its original value.

3. The incorrect summary problem. Another problem occurs if one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records.

Conclusion

This survey focuses on useful mechanisms for processing sets of items. All query processing algorithm implementations iterate over the members of their input sets; thus, sets are always represented by sequences. Sequences can be used to represent not only sets but also other one-dimensional "bulk" types such as lists, arrays, and time series, and many database query processing algorithms and techniques can be used to manipulate these other bulk types as well as sets. A complete query execution engine consists of a collection of operators and mechanisms to execute complex expressions using multiple operators, including multiple occurrences of the same operator.

Query optimization is the mapping from logical to physical operations, and the query execution engine is the implementation of operations on physical representation types and of mechanisms for coordination and cooperation among multiple such operations in complex queries. The mapping process is guided by the meta-data, i.e., the schema information that describes the data in the database.

References

[1] Adam and Wortmann 1989: N. R. Adam and J. C. Wortmann, Security-Control Methods for Statistical Databases:A Comparative Study, ACM Computing Surveys 21, 4 (December 1989), 515.

[2] Agrawal and DeWitt 1984: R. Agrawal and D. J. DeWitt, Whither Hundreds of Processors in a Database Machine, Proc. Int'l. Workshop on High-Level Architecture, Los Angeles, CA, 1984.

[3] Ahn and Snodgrass 1988: I. Ahn and R. Snodgrass, Partitioned storage for temporal databases, Inf. Sys. 13, 4 (1988), 369.

[4] Albert 1991: J. Albert, Algebraic Properties of Bag Data Types, Proc. Int'l. Conf. on Very Large Data Bases, Barcelona, Spain, September 1991, 211.

[5] Analyti and Pramanik 1992: A. Analyti and S. Pramanik, Fast Search in Main Memory Databases, Proc. ACM SIGMOD Conf., San Diego, CA, June 1992, 215.

[6] Anderson, Tzou, and Graham 1988: D. P. Anderson, S. Y. Tzou, and G. S. Graham, The DASH Virtual Memory System, November 1988.

[7] Antoshenkov 1993: G. Antoshenkov, Dynamic Query Optimization in Rdb/VMS, Proc. IEEE Int'l. Conf. on Data Eng., Vienna, Austria, April 1993, 538.

[8] Astrahan et al. 1976:Watson,System R: A Relational Approach to Database Management, ACM Trans. on Database Sys. 1, 2 (June 1976), 97. Reprinted in M. Stonebraker, Readings in Database Sys., Morgan-Kaufman, San Mateo, CA, 1988.