

**AN OVERVIEW OF
DISTRIBUTED DATABASE MANAGEMENT**

BY

AYSE YASEMIN SEYDIM

**CSE 8343 - DISTRIBUTED OPERATING SYSTEMS
FALL 1998 TERM PROJECT**

TABLE OF CONTENTS

INTRODUCTION.....	2
1. WHAT IS A DISTRIBUTED DATABASE SYSTEM?.....	3
2. ADVANTAGES AND DISADVANTAGES.....	6
3. TRANSPARENCY.....	7
4. DISTRIBUTED DBMS ISSUES.....	8
5. PARALLEL DATABASE SYSTEMS.....	10
CONCLUSION.....	12
REFERENCES.....	13

INTRODUCTION

Distributed database technology is one of the most important developments of the past decades. The maturation of data base management systems - DBMS - technology has coincided with significant developments in distributed computing and parallel processing technologies and the result is the emergence of distributed DBMSs and parallel DBMSs. These systems have started to become the dominant data management tools for highly intensive applications. The basic motivations for distributing databases are improved performance, increased availability, shareability, expandibility, and access flexibility. Although, there have been many research studies in these areas, some commercial systems can provide the whole functionality for distributed transaction processing. Important issues concerned in studies are database placement in the distributed environment, distributed query processing, distributed concurrency control algorithms, reliability and availability protocols and replication strategies.

As the subject is broad in a sense that the recent studies are all theoretical and most of the work being done in the past, only an overview of distributed database management could be made and presented in this report. Starting from the definition of a distributed databases, advantages, disadvantages, the main concern of promises and related transparency constraints are given. The main issues dealt in a distribution of data and applications are briefly discussed. The current trends and developments according to the researchers are summarized in the conclusion along with the self ideas.

1. WHAT IS A DISTRIBUTED DATABASE SYSTEM?

There is considerable confusion in the marketplace concerning the definition of a distributed DBMS. At the very least, it must provide a 'seamless' interface to data that are stored on multiple computer systems. As an example, if a personnel file is stored on a machine located in London and the department data is stored on a machine in Dallas in a company working worldwide, it must be possible to join these relations without explicitly logging on to both sites and assembling the data manually at some processing location, such as in their head quarter in New York. Instead by the use 'location transparency', one could simply state a query as selecting the employee names who are working on the first floors of their departments.

Several vendors are said to be marketing software as distributed databases that cannot run such a query and instead provide only a micro-to-mainframe connection. Location transparency can be provided either by a network file system (NFS) or a distributed database management system (DBMS). For a query like the above, an NFS solution would transfer both relations over the network and join them in the processing location. On the other side, by using a distributed DBMS, a heuristic optimizer may choose an intelligent accessing strategy and probably choose to move the first-floor departments to London, perform the join there and then move the result to New York. This strategy is said to be generally be orders of magnitude faster than a NFS strategy. This presents us that one should send the queries to the data instead of bringing the data to the query [1].

A distributed database is defined as a collection of multiple, *logically interrelated* databases distributed over a *computer network*. A distributed database management system (distributed DBMS) is then defined as the software system that permits the management of the distributed databases and makes this distribution transparent to the users. Distributed database system is to referred as a combination of the distributed databases and the distributed DBMS (Figure 1).

The implicit assumptions in a distributed database system are; data is physically stored across several sites, and each site is typically managed by DBMS that is capable of running independently of the other sites. The processors at these sites are connected by a computer network rather than a multiprocessor configuration. Distributed database is a database, not a collection of files that can be individually stored at each node of the network. To form a distributed database, distributed data should be *logically related* - where the relationship is defined according to some structural formalism - and access to data should be via a common interface. Distributed DBMS is neither a remote file system nor a transaction processing system. Transaction processing is not only one type of distributed application, but it is also among the functions provided by a distributed DBMS. However, a distributed DBMS provide other functions including integration of heterogeneous data, query optimization and processing, concurrency control and recovery [2].

Possible Implementation Alternatives

There are many possible distributed DBMS implementation alternatives. Client/server architectures, where multiple clients access a single server, is the most straightforward. What tasks will be done in the client's side and what kind of services that server should provide, whether message-passing or RPC features are to be used, caching will be done or not are the design considerations.

In a *multiple client/single server* system, server satisfies the queries and commands from clients, does not initiate conversation with clients. The user interface on the client side, provides user to form their queries in a predefined language and send to the server. The DBMS on the server side will provide the usual parts as Communications Manager, Query Optimizer, Lock Manager, Storage Manager and Cache Manager and it will take the query, process it and give back the resulting tables. In this way, resource sharing and availability will be better and full DBMS functionality will be provided to all clients. More efficient division of labor and better price/performance will be achieved. However, this implementation will become difficult to scale as more clients are added to the system and server will become a performance bottleneck. Server will also be a single point of failure, when the server fails, none of the clients will get service.

Multiple client/multiple server architectures are more flexible because the database is distributed across multiple servers. Each client machine has a '*home*' server to which it directs user requests. The communication of the servers among themselves to execute user queries and transactions is transparent to the users. Each server has its own DBMS services. Most current DBMSs implement one or the other type of client/server architectures.

A truly distributed DBMS does not distinguish between client and server machines. Ideally, each site can perform the functionality of a client and a server. Such architectures, *peer-to-peer*, require sophisticated protocols to manage data distributed across multiple sites. The complexity of required software has delayed the offering of peer-to-peer distributed DBMS products [3].

In a typical distributed DBMS component architecture, the user (or client of the database) software will contain User Interface Handler, Semantic Data Controller, Global Query Optimizer and Decomposer, Global Execution Monitor, External Schema and Global Conceptual Schema. The data (server) software will include Local Query Optimizer, Local Recovery Manager, Run-time Support Processor, System Log, Local Conceptual Schema and Local Internal Schema. The actual database file will be accessed by the Run-time Support Processor (Figure 2).

Figure 2 Components of a Distributed DBMS

2. ADVANTAGES AND DISADVANTAGES

The distribution of data and applications has promising advantages. Although they may not be fully satisfied by the time, these advantages are to be considered as objectives to be achieved.

Local Autonomy : Since data is distributed, a group of users that commonly share such data can have it placed at the site where they work, and thus have local control. By this way, users have some degree of freedom as accesses can be made independently from the global users.

Improved Performance : Data retrieved by a transaction may be stored at a number of sites, making it possible to execute the transaction in parallel. Besides, using several resources in parallel can significantly improve performance.

Improved Reliability/Availability : If data is replicated so that it exists at more than one site, a crash of one of the sites, or the failure of a communication line making some of these sites inaccessible, does not necessarily make the data impossible to reach. Furthermore, system crashes or communication failures do not cause total system not operable and distributed DBMS can still provide limited service.

Economics : If the data is geographically distributed and the application are related to these data, it may be much more economical, in terms of communication costs, to partition the application and do the processing at each site. On the other hand, the cost of having smaller computing powers at each site is much more less than the cost of having an equivalent power of a single mainframe.

Expandibility : Expansion can be easily achieved by adding processing and storage power to the existing network. It may not be possible to have a linear improvement in power but significant changes are still possible.

Shareability : If the information is not distributed, it is usually impossible to share data and resources. A distributed database makes this sharing feasible.

On the other hand, distribution of the database can cause several problems :

Lack of Experience : Some special solutions or prototype systems have not been tested in actual operating environments. More theoretical work is done compared to actual implementations.

Complexity : Distributed DBMS problems are more complex than centralized DBMS problems.

Cost : Additional hardware for communication mechanisms are needed as well as additional and more complex software may be necessary to solve the technical problems. The trade-off between increased profitability due to more efficient and timely use of information and due to new data processing sites, increased personnel costs has to be analyzed carefully.

Distribution of Control : The distribution creates problems of synchronization and coordination as the degree to which individual DBMSs can operate independently.

Security : Security can be easily controlled in a central location with the DBMS enforcing the rules. However, in distributed database system, network is involved which it has its own security requirements and security control becomes very complicated.

Difficulty of Change : All users have to use their legacy data implemented in previous generation systems and it is impossible to rewrite all applications at once. A distributed DBMS should support a graceful transition into a future architecture by allowing old applications for obsolete databases to survive with new applications written in current generation DBMSs.

In spite of the problems and its complexity, the users that will mostly benefit from the distributed DBMSs will be decentralized organizations, and applications such as, manufacturing, military command and control, EFT, corporate MIS, airlines, hotel chains.

3. TRANSPARENCY

The most commonly referred promises as advantages of the distributed DBMSs are;

- transparent management of distributed, fragmented, and replicated data,
- improved reliability/availability through distributed transactions,
- improved performance,
- easier and more economical expansion.

Transparency is the separation of the higher level semantics of a system from the lower level implementation issues. In a transparent system, system hides the implementation details from users. The advantage of a fully transparent DBMS is the high level of support that it provides for the development of complex applications. The fundamental issue to provide in a DBMS is the *data independence*, where the application programs are immune to changes in the logical or physical organization of data.

The distributed database technology intends to extend the concept of data independence to environments where data are distributed and replicated over a number of machines connected by a network. This is provided by several forms of transparency such that *network* (or *distribution*) transparency, *replication* transparency, and *fragmentation* transparency. Thus the database users would see a logically integrated, single-image database, even if it is physically distributed, enabling them to access the distributed database as if it were a centralized one. In its ideal form, full transparency would imply a query language interface to the distributed DBMS that is no different from that of a centralized DBMS.

Most of the commercial distributed DBMSs do not provide a sufficient level of transparency. Part of this is due to the lack of support for the management of replicated data but the operating system can assist in replication transparency. On the other hand, some systems require remote login to the DBMS for replication. Some distributed DBMSs attempt to establish transparent naming scheme, requiring users to specify the full path to data or to build aliases to avoid long names. Also for the network transparency, operating system support is needed. Generally no fragmentation transparency is supported but horizontal fragmentation techniques may come in distributed DBMSs. Most of the distributed DBMSs are supporting multiple client/single server architecture.

In commercial systems, some of them do not provide any of the transparencies, such as Sybase provides the primitives available, but applications have to implement distributed transactions themselves. In Oracle 6.x, one can open one database at a time, whereas Oracle 7, Ingres, NonStop SQL support distributed transactions.

Potentially improved performance depends generally on the data localization, enabling data to be stored in closed proximity to its points of use and the inherent parallelism of distributed systems. Data localization requires some support for fragmentation and replication instead of benefiting from the reduced contention and reduced communication overhead. The inherent parallelism may be exploited for interquery and intraquery parallelism. Interquery parallelism results from the ability to execute multiple queries at the same time, while intraquery parallelism is achieved by breaking up a single query into a number of subqueries, each of which is executed at a different site, accessing a different part of the distributed database. It must be noted that, parallelism would require full replication of data and updates to data must be implemented using distributed concurrency control and commit protocols. Some of commercial systems would multiplex the database as one is for query, one is for updates, whereas some others would do database updates during off-hours with batch jobs.

In a distributed environment, it is more easier to accommodate increasing database sizes. Expansion handled by adding processing and storage power to the system is referred to as database size scaling. In contrast, network scaling is much more difficult as it has many restrictions but if general and sufficiently powerful performance models, measurement tools, and methodologies are developed and detailed and comprehensive simulation studies are performed it would give better scalability results.

4. DISTRIBUTED DBMS ISSUES

When a distributed DBMS is to be designed the most important subjects that need to be considered are database design, query processing, concurrency control and reliability.

Distribution Design

In distributed database design, it is desirable to decide on how to distribute the database. The database is physically distributed across the data sites by fragmenting and replicating the data. Given a relational database schema, fragmentation subdivides each relation into horizontal (by a selection operation) or vertical fragmentation (by a projection operation) partitions. Fragmentation is desirable because it makes possible the placement of data in close nearness to its place of use, thus potentially reducing transmission cost, and it reduces the size of relations involved in user queries.

Based on the user access patterns, each of the fragments may also be replicated. This is preferable when the same data are accessed from applications that run at a number of sites. In this case, it may be more cost-effective to duplicate the data at a number of sites rather than continuously move it between them.

Directory management in a distributed DBMS has to be done in a similar way that the data placement. A directory may be global to the entire DBMS or local to each site; it can be centralized at one site or distributed over several sites; there can be a single copy or multiple copies.

Query Processing and Optimization

Query processing deals with designing algorithms that analyze queries and convert them into a series of data manipulation instructions. It is an optimization problem as it requires strategies executing the queries in a cost-effective way. Cost involves local processing time and the data transmission time and the general formulation of this problem is NP-hard in nature and the approaches are usually heuristic.

Query processing in distributed environment provides, on the controlling site, Query Decomposition using global schema, Data Localization using the fragment schema, and Global Optimization using the state of fragments information. On the local sites Local Optimization is done by using local schemes.

In a distributed DBMS, query processing and optimization techniques have to address difficulties arising from the fragmentation and distribution of data. To deal with fragmentation, data localization techniques are used where an algebraic query, which is specified on global relations, is transformed into one that operates on fragments rather than global relations. In the process, opportunities for parallel execution are identified (since fragments are stored at different sites) and unnecessary work is eliminated since some of the fragments are not involved in the query. Localization requires the optimization of global operations, which is undertaken as part of global query optimization. Global query optimization involves permuting the order of operations in a query, determining the execution sites for various distributed operations, and identifying the best distributed execution algorithm for distributed operations (especially the joins).

Concurrency Control

Concurrency control involves the synchronization of accesses to the distributed database, such that the integrity of the database is preserved. In distributed databases, one have to deal not only one database but with the consistency of multiple copies of the database. Besides, the deadlock problem in DDBSs is similar in nature to that is met in operating systems, including alternatives to prevention, avoidance, detection/recovery.

In distributed DBMSs, the challenge is synchronizing concurrent user transactions is to extend both serializability argument and concurrency control algorithms to the distributed execution environment. In these systems, the operations of a given transaction may execute at multiple sites where they access data. Thus global serializability requires that, the execution of the set of transactions at each site be serializable, and the serialization orders of these transactions at all these sites be identical.

If locking-based algorithm are used, lock tables and lock management responsibility may be centralized or distributed. A well-known side effect of all locking-based concurrency control algorithms is that they cause deadlocks. The detection and management of distributed deadlocks involving a number of site is difficult.

Reliability Protocols

The reliability and availability of a distributed DBMS involves issues that will make the system resilient to failures. Atomicity (do all the work or nothing) and durability (committed updates have to be permanent and up-to-date in each site) properties of the transactions must be satisfied.

In addition to transaction, system and media failures that can occur in centralized DBMS, a distributed DBMS must also deal with communication failures. In particular, the existence of both system and communication failures poses complications because it is not always possible to differentiate between the two. Distributed DBMS protocols have to deal with this uncertainty.

Distributed reliability protocols enforce atomicity of transactions by implementing atomic commitment protocols such as the two-phase commit (2PC). 2PC extends the effects of local atomic commit actions to distributed transactions by insisting that all sites involved in the execution of a distributed transaction agree to commit the transaction before its effects are made permanent (i.e. all sites terminate the transaction in the same manner).

The inverse termination is recovery. Distributed recovery protocols deal with the problem of recovering the database at a failed site to a consistent state when that site recovers from failure.

Replication Protocols

In replicated distributed databases, each logical data item has a number of physical instances. The issue in this type of a database system is to maintain some notion of consistency among the physical instance copies when users transparently update logical items. A straightforward consistency criterion is one copy equivalence, which asserts that the values of all physical copies of a logical data item should be identical when the transaction that updates it terminates.

A typical replica-control protocol that enforces one-copy serializability is known as Read-One / Write-All (ROWA) protocol. ROWA protocol is simple and straightforward, but it requires all copies of all logical data items are updated by a transaction be accessible for the transaction to terminate. Failure of one site may block a transaction, reducing database availability.

A number of alternative algorithms have been proposed that reduce the requirement that all copies of a logical data item be updated before the transaction can terminate. They relax ROWA by mapping each write to only a subset of the physical copies. One well-known approach is quorum-based voting, where copies are assigned votes and read and write operations have to collect votes and achieve a quorum to read/write data.

5. PARALLEL DATABASE SYSTEMS

A parallel database system can be defined as a DBMS implemented on a tightly coupled multiprocessor. The differences between a parallel DBMS and a distributed DBMS are not so clear. In particular, *shared-nothing* parallel DBMS architectures, which will be discussed in the following paragraphs, are said to be quite similar to the loosely interconnected distributed

systems. An important distinction is that distributed DBMSs assume *loose interconnection* between processors that have their own operating systems and operate independently. Parallel DBMSs exploit recent multiprocessor computer architectures in order to build high-performance and high availability database servers at a much lower price than equivalent mainframe computers.

Parallel system architectures range between two extremes, the *shared-nothing* and the *shared memory* architectures. A useful intermediate point is the *shared-disk* architecture. In the shared-nothing approach, each processor has exclusive access to its main memory and disk units. Thus each node can be viewed as a local site (with its own database and software) in a distributed DBMS. In the shared-memory approach, any processor has access to any memory module or disk unit through a fast interconnection network (high-speed bus or a cross-bar switch). In the shared-disk approach, any processor has access to any disk unit through the interconnection network, but exclusive (non-shared) access to its main memory. Each processor can then access database pages on the shared disk and copy them into its own cache. To avoid conflicting accesses to the same pages, global locking and protocols for the maintenance of cache coherence are needed.

Parallel query optimization takes advantage of both intra-operation parallelism and inter-operation parallelism. Intra-operation parallelism is achieved by executing an operation on several nodes of a multiprocessor machine. Parallel optimization to exploit intra-operation parallelism can make use of some of the techniques devised for distributed databases. Inter-operation parallelism occurs when two or more operations are executed in parallel, either as pipelined or independently. Independent parallelism occurs when operations are executed at the same time or in arbitrary order. Independent parallelism is possible only when the operations do not involve the same data.

As the developments in parallel DBMS technologies have fairly come to point of maturity, a number of issues are remaining for solution. Parallel data management techniques intend to overcome the startup delays in parallel operations, hot spot problems when accessing shared resources, and skewed data placement problems.

CONCLUSION

When we look to the history of databases, the technology is developed from graph-based systems, to relational systems. Because of its simplicity and clean concepts, many studies are accomplished in relational database design. While using relational model, to provide declaration of application specific types, a new data model is introduced based, on object-oriented programming principles. More recently, a hybrid model, object-relational model has emerged which embeds object-oriented features in a relational context. The use of objects has also been demonstrated as a way to achieve both interoperability of heterogeneous databases and modularity of the DBMS itself.

Sophisticated and reliable commercial distributed DBMSs are now available in the market, but there is also a number of issues need to be solved satisfactorily. These deal with skewed data placement in parallel DBMSs, network scaling problems, i.e calibrating distributed DBMSs for the specific characteristics of communication technologies such as broadband networks and mobile and cellular networks.

Advanced transaction models such as workflow models in distributed environments or models for mobile computing and distributed object management are among the research issues. Additionally, in a highly distributed environment, the cost of moving data can be extremely high. So the optimal usage of communication lines and caches on intermediate nodes becomes an important performance issue to be considered.

By the significant developments in Internet and usage of WWW, DBMS vendors are making their products web-enabled, to provide better web servers. By this way, a path to the direction of manipulation of huge volume of nonstandard data that exists on web is opened.

REFERENCES

1. M. Stonebraker, "Future Trends in Database Systems", in *Multidatabase Systems*, IEEE Computer Society Press, pp 339-350, 1994.
2. T.Ozsu, P.Valduriez, *Principles of Distributed Database Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1991..
3. T,Ozsu, P.Valduriez, "Distributed and Parallel Database Systems", *ACM Computing Surveys*, vol.28, no.1, pp 125-128, March 1996.
4. T,Ozsu, P.Valduriez, "Distributed Data Management: Unsolved Problems and New Issues", *Readings in Distributed Computing Systems*, IEEE Computer Society Press, pp 512-544, 1994.
5. A.Silberschatz, S.Zdonik, et.al., "Strategic Directions in Database Systems - Breaking Out of the Box", *ACM Computing Surveys*, vol.28, no.4, pp.764-778, Dec. 1996.
6. V.Kumar, M.Hsu, *Recovery Mechanisms in Database Systems*, Prentice Hall PTR, Prentice-Hall, Inc. 1998.