Distributed and Parallel Database Systems

M. TAMER ÖZSU

Department of Computing Science, University of Alberta, Edmonton, Canada ⟨*ozsu@cs.ualberta.can*⟩

PATRICK VALDURIEZ

INRIA, Rocquencourt, Le Chesnay Cedex, France

The maturation of database management system (DBMS) technology has coincided with significant developments in distributed computing and parallel processing technologies. The end result is the emergence of *distributed database management systems* and *parallel database management systems*. These systems have started to become the dominant data-management tools for highly data-intensive applications.

A distributed database (DDB) is 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 database and makes the distribution transparent to the users [Özsu and Valduriez 1991].

There are many possible distributed DBMS implementation alternatives. *Client/server architectures* [Orfali et al. 1994], where multiple clients access a single database server, is the most straightforward. *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. Most current database management systems 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, called *peer-to-peer*, require sophisticated protocols to manage the data distributed across multiple sites. The complexity of required software has delayed the offering of peer-to-peer distributed DBMS products.

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 (by a projection operation) partitions. Fragmentation is desirable because it makes possible the placement of data in close proximity 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.

A parallel DBMS [Valduriez 1993] can be defined as a DBMS implemented on a tightly coupled multiprocessor. The differences between a parallel DBMS and a distributed DBMS are somewhat unclear. In particular, shared-nothing parallel DBMS architectures, which we discuss in the following, are quite similar to the loosely interconnected distributed systems. Perhaps 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 unit(s). 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 interconnect (e.g., a high-speed bus or a cross-bar switch). In the shared-disk approach, any processor has access to any disk unit through the interconnect, but exclusive (nonshared) 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 coherency are needed.

Distributed and parallel DBMSs provide the same functionality as centralized DBMSs except in an environment where data are distributed across the sites on a computer network or across the nodes of a multiprocessor system. This functionality is provided *transparently*; thus the users are unaware of data distribution, fragmentation, and replication. Maintaining this view places significant challenges on system functions, which we summarize in the following.

Query Processing and Optimization. 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 (because fragments are stored at different sites) and unnecessary work is eliminated (because 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).

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 a dataflow or independently. We designate as *dataflow* the form of parallelism induced by *pipelining*. *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.

Concurrency control. In distributed DBMSs, the challenge in synchronizing concurrent user transactions is to extend both the *serializability* argument and the 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

(a) the execution of the set of transactions at each site be serializable, and
(b) the serialization orders of these transactions at all these sites be identical.

If locking-based algorithms 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 sites is difficult.

Reliability Protocols. In addition to transaction, system, and media failures that can occur in a 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* (all-or-nothing property) of transactions by implementing *atomic commitment protocols* such as the *two-phase commit* (2PC) [Gray 1979]. 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 of 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 the 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 data 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 that all copies of all logical data items that 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.

Distributed and parallel DBMS technologies have matured to the point that fairly sophisticated and reliable commercial systems are now available. As expected, a number of issues have yet to be satisfactorily resolved. These deal with skewed data placement in parallel DBMSs, network scaling problems (in particular, calibrating distributed DBMSs for the specific characteristics of emerging communication technologies such as broadband networks and mobile and cellular networks), advanced transaction models (now commonly called *workflow models*) [Elmagarmid 1992], multidatabase systems and interoperability [Sheth and Larson 1990], and distributed object management [Dogac et al. 1994; Özsu et al. 1994].

REFERENCES

DOGAC, A., ÖZSU, M. T., BILIRIS, A., AND SELLIS, T. Eds. 1994. *Advances in Object-Oriented Database Systems.* Springer-Verlag, Berlin.

ELMAGARMID, A. K., ED. 1992. *Transaction Models for Advanced Database Applications.* Morgan-Kaufmann, San Mateo, CA.

GRAY, J. N. 1979. Notes on data base operating systems. In *Operating Systems: An Advanced Course.* R. Bayer, R. M. Graham, and G. Seegmüller, Eds., Springer-Verlag, New York, 393–481.

ORFALI, R., HARKEY, D., AND EDWARDS, J. 1994. *Essential Client/Server Survival Guide.* Wiley, New York.

ÖZSU, M. T. AND VALDURIEZ, P. 1991. *Principles of Distributed Database Systems.* Prentice-Hall, Englewood Cliffs, NJ.

ÖZSU, M. T., DAYAL, U., AND VALDURIEZ, P., EDS. 1994. *Distributed Object Management.* Morgan-Kaufmann, San Mateo, CA.

SHETH, A. AND LARSON, J. 1990. Federated databases: Architectures and integration. *ACM Comput. Surv. 22,* 3 (Sept.), 183–236.

VALDURIEZ, P. 1993. Parallel database systems: Open problems and new issues. *Distrib. Parallel Databases 1,* 2 (April), 137–165.