A STATE OF THE ART REVIEW of DISTRIBUTED DATABASE TECHNOLOGY

Contract Number F30602–89–C–0082
Data & Analysis Center for Software

October 5, 1992

Prepared for:

Rome Laboratory
RL/C3CB
Griffiss AFB, NY 13441–5700

Prepared by:

Kaman Sciences Corporation
258 Genesee Street
Utica, New York 13502–4627

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*

2. REPORT DATE
October 5, 1992

3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE
A State of the Art Review of
Distributed Database Technology

5. FUNDING NUMBERS
F30602-89-C-0082

6. AUTHOR(S)
Carol Wawrzusin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Kaman Sciences Corporation
258 Genesee Street
Utica, NY 13502

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Sponsoring Org.	Monitoring Org.
Defense Technical Info. Ctr.	Rome Laboratory
DTIC/AI, Cameron Station	RL/C3C
Alexandria, VA 22304	Griffiss AFB, NY 13441

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
N/A

11. SUPPLEMENTARY NOTES
Available from: Data & Analysis Center for Software
P. O. Box 120
Utica, NY 13503

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release.
Distribution Unlimited

12b. DISTRIBUTION CODE
UL

13. ABSTRACT *(Maximum 200 words)*

A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network. A Distributed Database Management System (DDBMS) is a software system that permits the management of distributed data making the distribution transparent to the user. This report reviews the issues that arise with such systems, surveys current commercially available DDBMSes, and summarizes the state of the art. Although no standards yet exist within this new technology, some guidelines have been provided by C. J. Date and E. F. Codd. True implementations of general purpose DDBMSes are only now beginning to emerge in the marketplace. Their implementations with respect to issues of distributed database technology differ markedly.

14. SUBJECT TERMS
Distributed Databases, Database Management Systems, Relational Databases, Object-Oriented Databases

15. NUMBER OF PAGES
40

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

A STATE OF THE ART REVIEW of DISTRIBUTED DATABASE TECHNOLOGY

Contract Number F30602-89-C-0082
Data & Analysis Center for Software

October 5, 1992

Prepared for:

Rome Laboratory
RL/C3CB
Griffiss AFB, NY 13441-5700

Prepared by:

Kaman Sciences Corporation
258 Genesee Street
Utica, New York 13502-4627

1. INTRODUCTION 3
1.1 Evolution of Distributed Databases 3
1.2 Reasons for Distributed Databases 4

2. THE STATE OF THE ART IN DISTRIBUTED DATABASE TECHNOLOGY 6
2.1 The Architectural Issues of Distributed Databases 6
2.1.1 The Server/Host 6
2.1.2 The Network 6
2.1.3 The Data Model 8
2.1.4 The Schema 10
2.2 Functional Issues of Distributed Databases 11
2.2.1 Data Location and Function Distribution 11
2.2.1.1 Fragmentation 12
2.2.1.2 Replication 17
2.2.2 Distributed Query Processing 18
2.2.3 Transaction Management 19
2.2.3.1 Concurrency Control 19
2.2.3.1.1 Locking 19
2.2.3.1.2 Timestamping 19
2.2.3.1.3 Multiple Protocol Methodology 20
2.2.3.1.4 Deadlock Management 20
2.2.3.2 Reliability 20
2.3 Current Technology Implementing Distributed Databases 21
2.3.1 Data Models and Schema Integration 22
2.3.2 Data Distribution 22
2.3.2.1 Degree of Distribution 22
2.3.2.2 Location Transparency 22
2.3.3 Distributed Query Processing 23
2.3.4 Distributed Transaction Management 23
2.3.4.1 Concurrency Control Protocols 24
2.3.4.2 Recovery Protocols 24
2.3.4.3 Termination Protocols 24
2.3.4.4 Reliability 25
2.4 Implementation Strategies and Considerations 25
2.4.1 Degree of Site Autonomy 25
2.4.2 Lack of Standards 25
2.4.2.1 ANSI Standard SQL−2 26
2.4.2.2 Remote Data Access 27
2.4.2.3 ISO/OSI 27
2.4.3 Distributed Database Tools 27
2.4.4 Planning for the Future 27
2.5 Summary of the State of the Art 28
2.5.1 No Full Implementation 28
2.5.2 Market Pull versus Technology Push 28
2.6 Related Research Issues 28
2.6.1 Distributed Database Operating Systems 28
2.6.2 Distributed Multidatabase Operating Systems 29
2.6.3 Object Oriented Distributed Databases 29
2.6.4 Distributed Knowledge Bases 30

3. SUMMARY 31

BIBLIOGRAPHY 33

APPENDIX A: GLOSSARY 36

APPENDIX B: VENDORS IN DISTRIBUTED DATABASE
TECHNOLOGY 39

A STATE OF THE ART REVIEW
of
DISTRIBUTED DATABASE TECHNOLOGY

1. INTRODUCTION

1.1 Evolution of Distributed Databases

During the past twenty years, the practice of organizing repositories of data under a central point of control became commonplace. In an effort to overcome the unmanageable situation of applications generating and maintaining autonomous files of data, corporations organized their data into "centralized" databases, free of duplications and inconsistencies, managed by a database management system (DBMS) under the control of a central database administrator. Guarded by the MIS minions, this practice gave management a secure hold over corporate data, and at the same time gave the company's diverse internal organizations the illusion of being in control of their own data. For the corporate decision makers, gathering data was a "simple" process of querying the databases residing on the corporate mainframe.

In recent years the physical makeup of corporations, private entities and government activities, has changed from a centralized to a distributed structure. The rise of giant conglomerates is an example of this change. Within one of these composite organizations, even the manufacturing, engineering, and business units of one component can be geographically dispersed. The distributed physical architecture of these organizations demands that the information architecture also be distributed, embracing the concepts of open systems, distributed computing, and hardware and software independence. As a natural consequence of this distribution lies the requirement to distribute and manage the corporate data. Evolving from this need, the confluence of database technology and network technology has now produced one of the newest members of software engineering technology called distributed database technology.

A *distributed database* is a collection of multiple, logically interrelated databases distributed over a computer network. A distributed database management system (DDBMS) is a software system that permits the management of distributed data making the distribution transparent to the user. A distributed database is more reliable and more responsive than a centrally located and controlled database; data can be entered where it is generated, data at different sites can be shared, and data can be replicated giving users the option of accessing copies of the data in the event of a site or network failure. Outgrowing data storage resources or computing power doesn't necessitate moving up to the next expensive mainframe; distributed database technology allows affordable, incremental hardware growth.

As with all new technology, the definition of a distributed database was unclear until time and use brought clarification. During the early eighties, vendors selling "distributed" databases, users who felt a need to implement such a beast, and theorists who wrote articles dealing with the topic for technical publications all had their own ideas about what constituted a "distributed" database.

In the Summer 1987 publication of InfoDB, C.J. Date proposed twelve rules that apply to a distributed database. Like E.F. Codd's famous rules for the relational model, Date's have become a bible for distributed database technology. Summarized, the rules are as follows:

o *Site Autonomy*—Each site maintains local privacy and control of its own data; users that commonly share data can have it located at the site where they work

o *No Central Site*—The operation of the database does not depend on any single site; each site in the network runs local applications independently of the other sites, or globally on data at remote sites; no single DBMS is more necessary than any other

o *Continuous Operation*—The distributed database should never require downtime; planned activity should not require a shutdown

o *Transparency*—The location of the data does not need to be known to applications or users

o *Fragmentation Independence*—The division of a table into fragments should be transparent to the applications/users

o *Replication Independence*—Replication of data should be unknown to the user and updates to replicated data are performed transparently to the user

o *Distributed Query Processing*—The performance of a query should be independent of the site at which it is submitted; interleaved transactions updating multiple sites should be capable of serialization and, in the event of failure, should leave the database in a consistent state

o *Distributed Transaction Management*—The distributed system should be able to support atomic transactions

o *Hardware Independence*—The database should be able to integrate data from a wide variety of systems

o *Operating System Independence*—The database should be able to run on different operating systems

o *Network Independence*—The database must be able to operate using any communications protocol

o *DBMS Independence*—Databases must be able to communicate with those of other vendors

More recently, in 1990, Codd specified four minimal conditions to be satisfied by a distributed database. [38] These conditions are the following:

o *The database consists of data dispersed at two or more sites*

o *The sites are linked by a communications network*

o *At any site X, the users and programs can treat the totality of the data as if it were a single global database residing at X*

o *All of the data residing at any site X and participating in the global database can be treated by the users at site X in exactly the same way as if it were a local database isolated from the rest of the network*

Codd uses these four conditions to distinguish products that support true distributed database management from those supporting only distributed processing.

1.2 Reasons for Distributed Databases

Correctly implemented, distributed databases are more reliable, provide faster data access, reduce communications load, and allow for the incremental upward scaling of hardware. Among the many motivations for developing a distributed database, these are the most frequently encountered:

o a distributed organizational structure demands distributed data

o a need to generate global applications based on pre-existing databases

o a requirement to reduce communications costs

o increased performance or reliability demands

A DDBMS is homogeneous if the same DBMS occurs at each site regardless of the hardware and operating system.[6] Generally, when the motivation is to integrate pre-existing databases, the "bottom-up" design solution involves heterogeneous databases – those belonging to several vendors probably not based on the same data model. In other situations, a "top-down" design can be used which takes best advantage of the functionality of a distributed database. In either case, the database designer will need to know what technology is available to implement a distributed database system. Most DBMS vendors currently offer a "distributed" version of their product, but because of the lack of standards, these offerings vary in the level of support given to the various aspects of distributed database technology. Also, depending on the particular market served by a vendor, some aspects of distributed database technology are emphasized while others are minimized or non-existent.

In order to select the right DDBMS or to develop an optimum distributed design, the database system designer must understand the relative merits of each feature and be able to make tradeoffs to effectively match implemented features to the specific data needs to be supported.

The objective of this state of the art review is to review those unique features of distributed databases that distinguish them from centralized databases and to examine currently available implementations of these features.

2. THE STATE OF THE ART IN DISTRIBUTED DATABASE TECHNOLOGY

As in centralized databases, regardless of the underlying data model, the fundamental issue of distributed databases is *transparency*. In a centralized database, transparency refers only to data independence; in a distributed database, transparency refers to the data and to the network. According to Date's first rule, the distributed database should appear to the user as one, unified database. To accomplish this, not only the location of the data, but the very existence of the network must be transparent to the user.

The flipside of the transparency issue is the issue of local autonomy. Each site participating in a distributed database must be wholly independent; its operating system, administration, resident databases and associated catalogs must be totally autonomous.

Factors that come into play when considering transparency and local autonomy include architectural issues, such as the underlying data model, the schema, and the site and network hardware, and the functional issues, such as how and where data is located and how the system synchronizes updates among the participating sites.

2.1 The Architectural Issues of Distributed Databases

The architecture of a distributed database includes the physical components of hosts/servers and the network connecting them, the data models used to implement the component databases, and the schema used to integrate the various independent databases. The structure of a distributed database system is shown in Figure 1.

2.1.1 The Server/Host

The architecture of a distributed database permits a very large database to be supported on a collection of host equipment of varying capacities and performance levels. Each participating site in a network is a general-purpose computer that executes both local application programs and distributed database management functions. These computers range in size from personal computers to powerful workstations and parallel computers.

One of the strong points of distributed database technology is the ability to incrementally add host power to a database structure. However, the design of the allocation of data to the hosts must consider the performance characteristics of each host in order to ensure the most efficient operation of the distributed database system. A frequently used portion of the database should not be allocated to a host with inferior performance characteristics, to a host connected to an unreliable power supply, or to a host on a poorly performing, or overloaded Local Area Network (LAN). Taking advantage of the performance improvements offered by parallel architectures, recent trends in distributed database technology are toward assigning database functions to dedicated data servers where the servers are parallel processors. These machines are capable of hosting very large (many gigabyte) databases, enhancing performance by concurrent execution of parallel, complex queries, and significantly reducing the I/O bottleneck via parallel disk accessing.

2.1.2 The Network

The communication network connecting the sites cooperating in a distributed database are most frequently one of these three basic types:

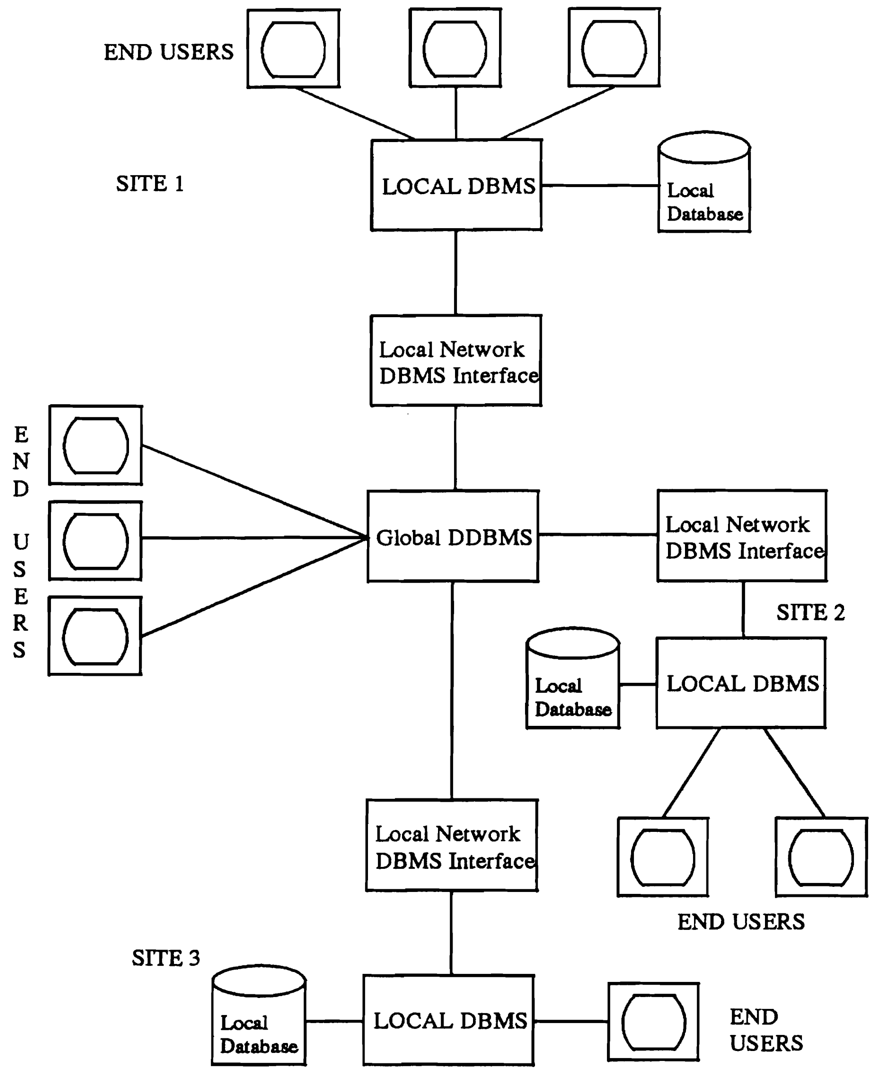

FIGURE 1: The Structure of a Distributed Database [6]

o high bandwidth, low delay "Ethernet"–like local area networks

o lower bandwidth, higher delay, longer range packet–switch networks, like Arpanet

o lower bandwidth, lower delay point–to–point leased circuit

Radio and/or satellite broadcast networks are also being employed as distributed database technology gains popularity.

Not part of the technology of distributed databases per se, but certainly within the purview of the database implementor, are the compression/decompression algorithms employed throughout the configuration of the database. This includes an analysis of any bridge, router, and gateway hardware and software that may be part of the total system. As the number and types of hosts (and workstations) on a network, and the number of local area networks (LANs) comprising a system increase, the ability of the bridges, routers, and gateways to handle the traffic efficiency is seriously affected.

2.1.3 The Data Model

Treated as an extension of centralized database technology, considerable discussion and disagreement has continued concerning the appropriate data model to be used for distributed databases. A collection of conceptual tools for describing data, data relationships, data semantics, and consistency restraints, a data model can be one of three types: object–based logical model, record–based logical model, and physical data model.[10]

Object–based logical models provide flexible structuring capabilities and allow the explicit specification of data constraints. The entity–relationship (E–R) model is an example of an object based logical model. It models the real world as a collection of objects, called entities, and the relations between them. Shown in Figure 2 is an E–R model of customers, accounts, and the relationship of "customer account".

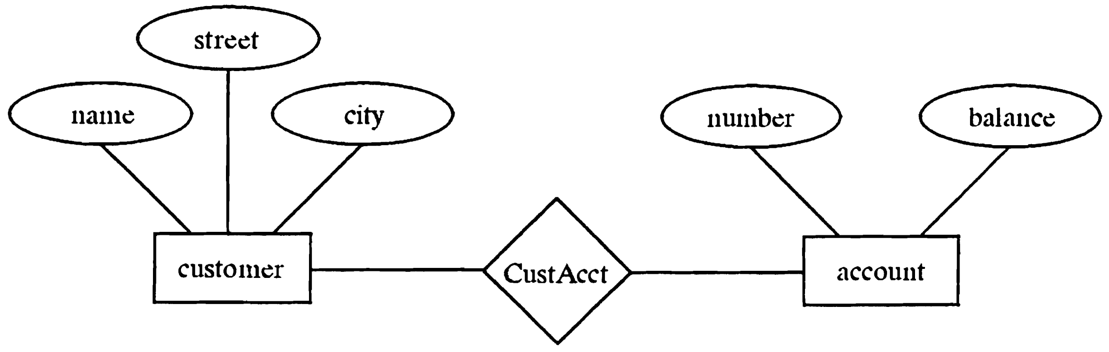

FIGURE 2: E–R Model of Customer Account

Record–based logical models describe data at the conceptual and view levels, specifying both the overall logical structure and the implementation, but not the data constraints. The relational model, a record–based logical model, represents data and the relationships between data as a collection of tables. Figure 3 shows the customer account as a relational model.

name	street	city	number
Lowery	Maple	Queens	900
Shiver	North	Bronx	556
Shiver	North	Bronx	647
Hodges	Sidehill	Brooklyn	801
Hodges	Sidehill	Brooklyn	647

number	balance
900	55
556	100000
647	105366
801	10533

FIGURE 3: Relational Model of Customer Account

The network model, another record–based logical model, represents data by a collection of records and represents relationships among data by links. Figure 4 is a network model of the customer account.

Physical data models, which describe data at the lowest level, are not very popular and are not considered appropriate in the context of distributed databases.

There has been research done into the "universal" model which takes all the relations in a regular relational database and glues them together by means of one operator (natural join) to form a single relation of very high degree that contains all the information in the database.[40] The universal relational model aims at achieving complete access–path independence in relational databases by relieving the user of the need for logical navigation among relations. Access paths are embedded in attribute names, hiding all information about the logical structure of the database from the user. Although relational databases removed the need for physical navigation, access paths among relations must still be specified. The motivation behind the universal relational model is to fully realize Codd's goal to free users from the need to specify access paths.

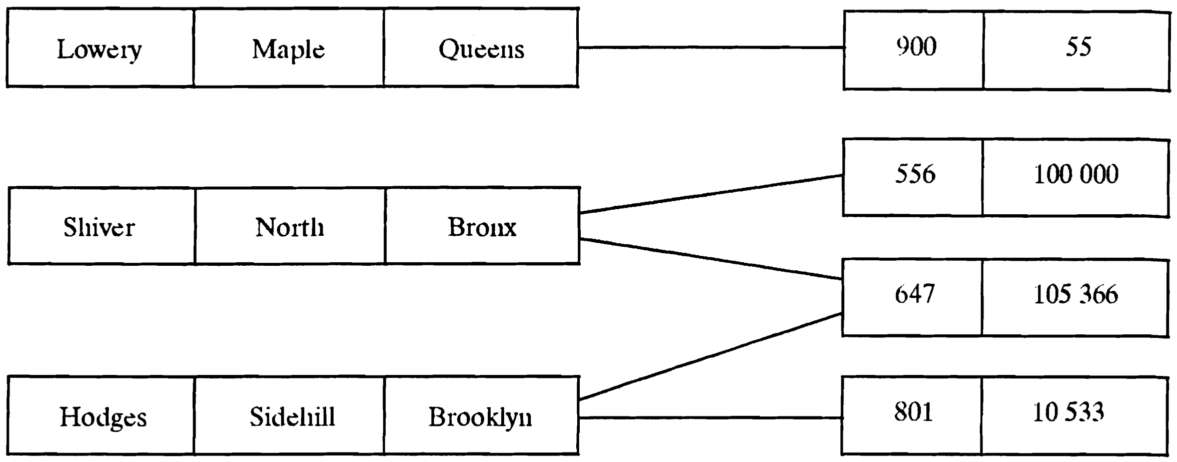

FIGURE 4: Network Model of Customer Account

Among the current leading contenders for use with distributed databases are the record–based relational and network models and the object–oriented model. However, as it has with centralized databases, the relational data model has become the de facto standard for DDBMS. E.F. Codd, founder of the relational model, holds that distributed database technology is only feasible when based on a relational model. As characterized by Codd, the relational model contains simple data structures, provides a solid foundation for data consistency, and allows set–oriented manipulations of relations. These three powerful features have propelled the relational model to the forefront of the technology.

The superiority of the relational model for use in distributed databases is refuted by recent work done at the German National Research Center for Computer Science which espouses the use of an object–oriented database approach to distributed database management.[4] A discussion of this effort is found in section 2.6.3.

It should be noted that it is possible to build a distributed database system without a single "global" data model. Providing a high degree of site autonomy by not enforcing a global data model or schema, the Sybase DDBMS product supports distributed operations via application programming or database–oriented remote procedure calls (RPCs) between Structured Query Language (SQL) Servers.[23] When multiple data models exist within a distributed database system, the system must provide for mapping from structures of one DBMS to another and for translating the commands of one DBMS's data manipulation language to their equivalents in the data manipulation language of the other DBMS(s).[6] For example, Ingres' distributed product, Ingres/STAR, provides these functions via gateway products (restrictions apply to the location of the global data dictionary). Also providing transparent join and view of multiple databases, the Informix–STAR product includes an extended synonyms feature permitting users to employ synonyms as pointers when tables are moved between sites thus freeing them from the need to specify which computer to access.

2.1.4 The Schema

The schema describes a database as it is stored; it describes physical characteristics such as format, storage location, and access paths, and defines the logical structure of the database. In a centralized database, the schema is the global view of the database in terms of which all user views, called subschemas, are defined. In a distributed database, schema integration refers to the way users logically view the distributed data. Whether the

DDBMS is homogeneous or heterogeneous, there are two kinds of schemas – the global schema and the local schema. The global schema defines all of the data in the system; the local schema defines the data at the local sites.

When the distributed database involves heterogeneous databases, two general approaches exist to mapping the component distributed database schemas:

o integrate all the local schemas into one global schema and derive all user views from the global schema

o integrate various portions of the local schemas into multiple federated schemas

In the heterogeneous case, the designer may also be faced with a schema translation problem when different data models are involved or different naming conventions are used. Data mapping may also be required if data types or data values need to be converted for conformity (for instance, temperatures stored as Fahrenheit and Celsius).

Whatever schema approach is chosen, users should be able to refer to and create tables by name without needing to know where in the system the table is physically located or having to be concerned about naming conflicts. The ability of the database to ensure unique system names is provided through a catalog called the data dictionary. Information about sites and storage structures, database sizes and other statistics, access privileges, fragmentation and replication of tables, and system naming conventions are kept in a global data dictionary which is itself a distributed database.

The conceptual problems associated with the schema are embodied in the data dictionary. If it is kept at a single site there then exists a single point of failure. If replicated at every node, then every change in its information requires a change at every site. Some DDBMSs employ an approach in which each site maintains its own local catalog which the system searches for each reference to a table. This method saves in the maintenance effort but generates overhead network traffic.

2.2 Functional Issues of Distributed Databases

Transparency refers to the separation of the higher–level semantics of a system from its lower level implementation issues. It is the fundamental characteristic of a distributed database, with the degree of transparency being directly related to the degree of distribution. In order to achieve a high degree of transparency, the system must automatically record and maintain information about the location of the data in the database, the status of transactions, failure of any site or communication link in the network, and must support commit and recovery protocols for ensuring transaction atomicity, isolation, and durability. These concerns can be divided into four major issues pertaining to functionality: data location and function distribution, transaction management, and query processing.

2.2.1 Data Location and Function Distribution

General distributed processing allocates parts of an application to different machines based on where the parts are required. The user is very much aware of the distribution; the user must physically move between machines to perform different application functions. Within the context of a distributed database, the functional distribution implies that the data is distributed across database servers based on where the data is required. The fact that the data is distributed is transparent to the user.

Location transparency, also termed distribution independence, hides the physical distribution of the data from the user. Supporting location transparency is the single most important function of a distributed database system. Programs must continue to operate regardless of the distribution configuration of the data. Codd points out that only prototypes and products based on the relational model have been able to demonstrate the capability of supporting distribution independence.[38]

Data distribution is the single function under the control of the applications system designer or database administrator (the remaining issues are generally an integral part of the DDBMS). The two key issues of designing a distributed database are it's data fragmentation and allocation. The purpose of fragmenting, or breaking up, the tables of a database and allocating them to one or more sites in the network is to increase the performance and/or reliability of applications using the database. The problems associated with the allocation of the fragments are similar to those encountered in allocating files to nodes on a computer network. Although much of the research into file allocation can been applied to fragment allocation, there currently are no automated tools or allocation algorithms available to aid in evaluating alternative allocation designs.

2.2.1.1 Fragmentation

The ability to *fragment* a relation, dividing it into subrelations and allocating the subrelations to a subset of participating sites, is the distinguishing feature of distributed database technology. Since applications generally view subsets of a relation, subsets are natural units of distribution. Fragmentation permits this finer granularity in the unit of data distribution.

Fragment groups are collections that include the primary fragment and those fragments resulting from derived fragmenting. Derived fragmentation refers to relations that become partitioned due to a primary fragmentation performed elsewhere. For example, if a relation dealing with department numbers in an organization is fragmented by department location, then another relation dealing with employee data may become fragmented since the employee's department number would most likely appear in such a relationship.

Fragmenting a relation can be performed horizontally, vertically, or in combination. Horizontal fragmentation partitions a relation along its rows; for example, if a column in a relation contains site identification data, it makes sense to store records associated with a site at that site, giving local users local access to the data.

Table 1 shows the global relationship Projects containing software project data. Within Projects are the project number, the title of software project, its approximate dollar value, and the performing location.

Table Projects

Number	Title	$Value	Location
1	Maintenance	500,000	New Jersey
2	Database Devel	200,000	Alabama
3	Training	150,000	New Jersey
4	Requirements	400,000	Illinois
5	Training	175,000	Alabama

TABLE 1: Global relationship Projects

Two examples of possible horizontal partitionings of the global relationship Projects are shown below. In Tables 2(a), (b), and (c) the relationship is partitioned along the location column. The three resulting fragments would ideally be allocated to hosts at their respective site locations. This scheme places geographically related data closest to the location most likely to be accessing the data, and reduces the processing required when joins are executed against the Projects data at each site. Of course, additional communications costs would be incurred if, for instance, the New Jersey site required access to the Alabama data.

Table Projects (New Jersey)

Number	Title	$Value	Location
1	Maintenance	500,000	New Jersey
3	Training	150,000	New Jersey

TABLE 2(a): Horizontal fragmenting of Projects by Location, New Jersey fragment

Table Projects (Alabama)

Number	Title	$Value	Location
2	Database Devel	200,000	Alabama
5	Training	175,000	Alabama

TABLE 2(b): Horizontal fragmenting of Projects by Location, Alabama fragment

Table Projects (Illinois)

Number	Title	$Value	Location
4	Requirements	400,000	Illinois

TABLE 2(c): Horizontal fragmenting of Projects by Location, Illinois fragment

The fragmentation by dollar value shown in Tables 2(d) and (e) might be of value in situations where different organizations, each with its own computing resources, must deal with data based on a dollar threshold. For example, if the purchasing function were divided such that one organization dealt with orders exceeding $200,000 while another dealt with orders of lesser value, this fragmentation might be suitable. Locality of reference is the relevant criterion for the design of fragments.[16]

Table Projects ($Value > 200,000)

Number	Title	$Value	Location
1	Maintenance	500,000	New Jersey
4	Requirements	400,000	Illinois

TABLE 2(d): Horizontal fragmenting of Projects by $Value, > $200,000

Table Projects ($Value <= 200,000)

Number	Title	$Value	Location
2	Database Devel	200,000	Alabama
3	Training	150,000	New Jersey
5	Training	175,000	Alabama

TABLE 2(e): Horizontal fragmenting of Projects by $Value, <= $200,000

Using SQL, the horizontal fragmentation of Projects based on location would be defined as follows:

```
New_Jersey_Projects =
        select * from Projects where Location = 'New Jersey'

Alabama_Projects =
        select * from Projects where Location = 'Alabama'

Illinois_Projects =
        select * from Projects where Location = 'Illinois'
```

The horizontal fragmentation of Projects based on dollar value would be defined as follows:

```
Projects_Over =
        select * from Projects where $Value > 200,000

Projects_Not_Over =
        select * from Projects where $Value <= 200,000
```

Vertical fragmentation partitions a relation into smaller relations with the goal of minimizing the execution time of user applications on the fragments. Joins performed on the smaller relations will require much less processing time. The concept of vertical partitioning, developed within the context of centralized databases for the same reason, is useful in distributed databases where each fragment may contain data with common geographical properties. Table 3 shows a vertical partitioning of Projects. Note that the primary key of Projects, project_ number, appears as the primary key of each vertical fragment.

Table Projects (Title and Location)

Number	Title	Location
1	Maintenance	New Jersey
2	Database Devel	Alabama
3	Training	New Jersey
4	Requirements	Illinois
5	Training	Alabama

Table Projects ($Value)

Number	$Value
1	500,000
2	200,000
3	150,000
4	400,000
5	175,000

TABLE 3: Vertical fragmentation of Projects

Using SQL, the definition for the vertical fragmentation of global relationship Projects as shown in Table 3 is as follows:

```
Projects_Title_Location =
            select Number, Title, Location from Projects
Projects_$Value =
            select Number, $Value from Projects
```

Keeping in mind that the performance of query execution will be affected by the extent to which a database is fragmented, the database designer must determine the correct level of fragmentation while maintaining the following properties:

1. Completeness – If a global relation is decomposed into fragments, each data item that can be found in the global relationship can also be found in one or more of the fragments. This property ensures against loss of data.

2. Reconstruction – It must always be possible to reconstruct a global relation by joining the fragments together. Horizontal fragments can be recombined by using the SQL UNION operator. In vertical partitioning this is generally accomplished by including the key of the global relationship in each fragment guaranteeing the reconstruction through a join relationship. This property ensures that constraints defined on the data in the form of dependencies are preserved.

3. Disjointedness – If a global relation is horizontally decomposed into fragments, any individual data item can be found in only one of the fragments. Since the primary key attributes of a relation are typically repeated in each of its vertical fragments, disjointedness is defined only on the nonprimary key attributes of a vertical fragmentation.

In a distributed database system, a query written against a fragmented database would look exactly like a query written against a centralized database. However, since no current DDBMS product supports fragmentation, the user must know how the database is fragmented to be able to construct correct queries.

Table 4 shows the relationship Staff occurring in the database of software engineering projects. Staff contains the employee identification number, name, position, and number of current project assignment.

Table Staff

Id#	Name	Position	Number
30248	Jones, J.	Programmer	2
19846	Smith, M.	Manager	5
10002	Wilson, R.	Analyst	4
48051	Williams, R.	Programmer	1
34143	Larsen, T.	DB Specialist	2

TABLE 4: Global relationship Staff

To find out where programmers are correctly working the SQL query against either a distributed or centralized database would be:

```
select Title, Location from Staff, Projects
            where Projects.Number = Staff.Number
            and Position = 'Programmer'
```

A true distributed database system would automatically expand this query to some equivalent of the following:

```
select Title, Location from Staff, New_Jersey_Projects
            where New_Jersey_Projects.Number = Staff.Number
            and Position = 'Programmer'
if not found
select Title, Location from Staff, Alabama_Projects
            where Alabama_Projects.Number = Staff.Number
            and Position = 'Programmer'
if not found
select Title, Location from Staff, Illinois_Projects
            where Illinois_Projects.Number = Staff.Number
            and Position = 'Programmer'
else print "No programmers assigned"
```

With the current state–of–the–art in distributed databases, the user must be aware of the fragmentation and must provide for the expansion.

2.2.1.2 Replication

Once fragmentation has been completed, the individual fragments must be allocated to various sites on the network. A big decision for the distributed database designer is whether any or all of the fragments should be maintained at more than one site. If no data is replicated, the system is referred to as *partitioned*; if all the data exists at every site, the system is termed *fully replicated*; if only some fragments exist at multiple sites the system is called *partially replicated*.

An optimal allocation of fragments must address both the costs associated with storing multiple copies and the performance of the resultant system. Storing costs must include the physical storage costs, the querying costs, and the updating costs, where updating involves concurrency control mechanisms and integrity enforcements across multiple copies of data.

Transaction oriented database applications demand a high level of reliability and availability. With multiple copies, the probability that some copy will be available even when system failures occur is high. For read only queries accessing the same data, multiple copies provide an opportunity for parallel execution. While a system with no data replication eliminates the complexities related to update synchronization, reliability and performance requirements may dictate either full replication of data at each site or varying degrees of partial replication. Although partial replication introduces the additional cost of remote accesses, the cost is low when compared with the costs associated with write operations in a fully replicated situation.[37] Data placement is essentially a trade–off between update costs and the benefits of increased reliability and performance.

The complexity of synchronization procedures and the level of communications required are dependent on the number of copies of the data maintained in the system. Using their robust and adaptable distributed database system, RAID, Purdue University is currently carrying out experiments to obtain measurements that provide empirical evaluation of algorithms used in distributed database systems. [37] The Raid experiments have examined replicated copy control during site failure and recovery to determine how fast database consistency can be restored and what are the associated costs. At system configuration time, Raid provides a threshold value which specifies the minimum number of copies of each database object to be maintained in the system. Availability was measured by the number of aborted transactions due to site failures. The experiments indicate that thresholds up to three improve availability, while those above three yield substantially smaller improvements, and full replication produces the poorest performance results.

2.2.2 Distributed Query Processing

The query processor in a centralized DBMS transforms high–level queries into equivalent lower–level queries which implement the execution strategy, focusing on optimization of performance primarily by reducing disk accesses.

Distributed query processing must deal with the analysis, optimization, and execution of queries referencing distributed data. Query optimization and execution in a distributed database environment involves global and local optimization plans and the selection of access paths. Choices concerning the best site to process data and how data should be moved between sites make the task of distributed query processing significantly more complex than the centralized version.

A distributed query optimizer decomposes a query into a sequence of serial and parallel operations, groups the operations that can be performed at the same site, and stages the transmission of results between sites to eventually yield the desired result. The dynamic nature of a DDBMS adds to the complexity of the optimizer, since each site must also carry on its own local execution load, while the network is subjected to varying traffic patterns and bottlenecks. Optimizing distributed queries involves consideration of the following:

o speed differences in communication links

o speeds and loads of local processors

o nature of operations at sites

o possible parallelism in query execution

o replicated/fragmented data possibilities

Significant research has occurred in the area of distributed query processing. The results of this research can be observed in the variety of implementations currently found in commercial and research systems. The research emphasis has been on finding methods that minimize the costs associated with intersite communication. In most cases, optimization is broken into two separate problems: selection of a global execution strategy, based on intersite communication, and selection of each local execution strategy, based on centralized query processing algorithms. [31]

Just as the ordering of joins is important in centralized databases, it is more important in distributed databases because of the existence of fragments where their joining may significantly increase communication costs. Some optimizing algorithms exploit the existence of replicated fragments at run time in order to minimize communication costs by using the semijoin operation to reduce the amount of data that must be moved between sites. However, because the semijoin also increases the overhead associated with control messages, some recent systems no longer rely on it except in cases where it significantly reduces the amount of data that must be moved.[31]

Successful distributed query processing often depends on the availability of database statistics. Where limited bandwidth is a determining factor, the selection of the ordering of operations is a critical operation. In order to make this selection, the optimizer must have at it's disposal statistics on the database fragments with which it will dynamically estimate the cardinalities of results of relational operations. These statistics are maintained in the data dictionary which, depending on the particular DDBMS implementation, may be centralized or distributed. The designer of a distributed database who anticipates heavy distributed querying activity should be especially concerned about the distributed capabilities of the DDBMS's data dictionary.

2.2.3 Transaction Management

Managing transactions in a distributed database environment requires dealing with concurrency control, system reliability, and the efficiency of the system as a whole. The execution of transactions must be done in a way that preserves the characteristics of transactions, minimizes the cost, and maximizes system availability. The transaction manager must provide the system with resiliency. Despite component failures, the system must be able to continue operations and ensure that database consistency is not violated.

2.2.3.1 Concurrency Control

Concurrency control is the most difficult of the problems faced by distributed databases when data redundancy is permitted. Generally, the techniques in use today to maintain data consistency while minimizing the overhead of propagating control information to all nodes in the network are extensions of one or both of the same techniques used in centralized databases – locks and timestamps. Likewise, when locking is used as the method of synchronization, deadlock of the DDBMS can result. Well established methods of deadlock prevention, avoidance, and detection can be applied to distributed database systems.

2.2.3.1.1 Locking

Locking, the simplest form of concurrency control to implement, is the method most used in centralized DBMS products. Those portions of a database involved in a read or write operation are "locked", made unavailable for any other operations. Differences in DBMS products can be found in the granularity of the locks; products may "lock" at the data item, the record, the page, the table or file, etc.

When used in a distributed database environment, the locking method results in long delays while the locking protocol is propagated to all the affected nodes, the transaction is accomplished, and the acknowledgements are again propagated. For an "n" node network, straightforward locking involves 5n internode messages to accomplish one transaction as follows: n lock messages, n lock grant messages, n update messages, n update acknowledgments, and n release lock messages. Several variations of locking, including the popular "two phase commit", reduce the number of messages to 4n, 3n, and even 1.5n by using concepts such as majority locking, where only a majority of the nodes are required for a commit rather than unanimous approval, and piggybacking update messages on top of lock requests, but all of these techniques prove to be unsatisfactory in situations involving large numbers of sites and high transaction volumes.[40]

Another variation of locking, the primary–site concept, involves funnelling all updates for given partitions of the database through a primary site. When requests for data conform to well–defined patterns, for instance, by geographical location, the primary site technique is effective; however, when requests can span multiple primary sites this technique can result in global database locking.[40]

2.2.3.1.2 Timestamping

Timestamp–based concurrency control algorithms establish a serialization ordering of transactions by assigning to each a unique identifier, usually a composite stamp containing a site identifier and a monotonically increasing counter value. The transactions are executed according to the assigned order.

To compensate for the real–life situation of operations arriving at nodes out of sequence, each *data item* is assigned two timestamps. The read timestamp indicates the largest timestamp of transactions to have read the data item; the write timestamp indicates the largest timestamp to have updated the data item. The transaction manager compares the value of a data item's timestamps to those of the incoming transactions to determine if it should apply the transaction.

A hybrid class of locking–based algorithms also use timestamping to improve efficiency and the level of concurrency. These algorithms are not currently implemented in any commercial or research distributed DBMS. [31]

2.2.3.1.3 Multiple Protocol Methodology

Another concurrency control method under development today employs several different synchronization techniques depending on the transaction being executed. At system design time, after an analysis of the ways in which transactions can interfere with each other, several synchronization protocols are established which vary in cost according to the level of control provided. Transactions are identified as belonging to a class depending on the level of concurrency control required to maintain consistency. At run–time, the system does a table look–up to determine which protocol to employ; if the transaction belongs to several classes, the system chooses the most efficient, if it doesn't belong to any, the system imposes the strongest protocol defined. This technique, implemented in IBM's experimental R* distributed database system, is reported as providing the fastest, lowest cost method of concurrency control at this time.[40]

2.2.3.1.4 Deadlock Management

When locking based algorithms are used to provide distributed concurrency control in a system containing redundant data, system deadlock – a circular waiting situation – can occur. Most DDBMS products strive to prevent deadlocks by using timeouts as a detection mechanism. The timeout method causes a transaction to abort after waiting for a resource for a given time interval. Determining an appropriate value for the interval is difficult in a distributed environment because of the unpredictable load on the network and site hosts. A longer timeout value introduces unnecessary delay, while shorter intervals cause unnecessary aborts. A phenomena associated with short values is the cascading effect caused when an overloaded system causes aborts which generate more aborts, increasing the load.

2.2.3.2 Reliability

Within a distributed database environment, the database recovery manager must deal with four types of failures: transaction failures, media failures, site failures, and communication failures.

Transaction, media, and site failures are common to both centralized and distributed DBMSs. Transaction failures, usually caused by an error in the data or by the existence or potential for deadlock, are handled by aborting the transaction and restoring the database to its state prior to the transaction. Media failures, which result in levels of data loss ranging from complete loss of the stable database and/or the database log to loss of recent transactions, are most often repaired by either a full restore from an archive copy or a restore accomplished by redoing and undoing transactions stored in the database log.

Unique to distributed databases, communication failures generally are related to messages that either contain errors, are delivered out of sequence, or are lost. The lower three layers of the International Standards Organization's Open Systems Interconnect (ISO/OSI) architecture are expected to handle the first two types of message related errors. Lost messages, typically the result of communication line or site failures, must be handled by the DDBMS. In the event of communication line failures the network may become divided, known as *partitioned*, and each partition may continue operation. Maintaining the consistency of a distributed database across a partitioned network, especially if replication of data exists across the partitions, is a monumental task for the distributed transaction manager.

Protocols employed in reliability techniques include the commit, terminate, and recover protocols. Commit and recover protocols exist in centralized DBMSs, but their implementation differs in DDBMSs. Maintaining the atomicity of transactions across multiple sites implies that if a transaction fails at one site, it must be aborted all all other sites. Termination protocols, unique to distributed databases, complement recovery protocols; while recovery deals with re–establishing a consistent database across multiple sites, termination deals with terminating active transactions when a failure has occurred at one or more sites. The commit protocol must ensure that the effects of a transaction across the entire database is an all or nothing situation.

2.3 Current Technology Implementing Distributed Databases

When contemplating entry into a new technology domain, most systems designers survey the current *implemented* state–of–the–art. The majority of today's *implemented* distributed databases are of the heterogeneous variety, having been developed as a response to the problem of integrating databases scattered throughout organizations. Among the most documented cases, all having been underway for several years, are General Motors' DATAPLEX, Amoco's *Amoco Distributed Database System* (ADDS), Xerox's MULTIBASE, and the *Integrated Manufacturing Data Administration System* (IMDAS) developed to support the National Bureau of Standards Automated Manufacturing Research Facility. Each of these systems is a special purpose, one of a kind system, customized to include those features of distributed database technology which meet the needs of the organization.

By and large, "bottom–up" distributed database implementations have been accomplished through sizable in–house projects involving years of effort. For example, IMDAS, developed to support a prototype computer integrated manufacturing environment at the National Institute of Standards and Technology, represents *15–20 staff years* of effort. In addition to the substantial problems related to handling distributed data, these programs must deal with a multitude of heterogeneity issues in areas such as the following:

o computer hardware

o operating systems

o communications protocols

o communications links

o database management systems

o data models

o data representations

o data manipulation languages

o transaction management protocols

Within these custom systems can be found sophisticated solutions to the problems of distributed databases. However, these solutions are tailored to the needs of the particular implementation.

The major focus of research and development today is to develop general purpose distributed database management systems that will solve a wide range of data management problems. True implementations of general purpose distributed database technology are only now beginning to emerge in the marketplace. These products are generally homogeneous solutions with limited support for the heterogeneous environment via proprietary gateway products. Their implementations with respect to the issues of distributed database technology differ markedly.

2.3.1 Data Models and Schema Integration

The most desirable distributed database design would be based on a single data model, preferably one in which each site implemented the same database management system. In this situation, all the problems of disparate schema integration and query language translation disappear. Because we hardly ever get to deal with the "perfect" situation, several commercial distributed database products now support multiple data models and network protocols via gateway products. Although all these products use the relational data model for the native system, gateway products provide them with the ability to incorporate older hierarchical and network databases as nodes. Factors to be considered when beginning the design of a distributed database include the following:

o whether or not the system enforces a global data model

o the methodology employed to generate the global schema

o naming conventions imposed

o the location(s) of the global dictionary

2.3.2 Data Distribution

The advantages to be offered by data distribution must be fulfilled by the DDBMS. Offsetting the promises of improved performance, reliability, and availability are the complexities related to update synchronization, distribution of control, security and the general lack of experience dealing with distributed databases. Therefore, when the development of or migration to distributed databases is contemplated, the degree of distribution and level of location transparency supported by a DDBMS product are factors for serious consideration.

2.3.2.1 Degree of Distribution

In the design of a distributed database, it may be decided that the organization's structure, geographical dispersion, or other data requirements may necessitate or lend itself to the use of fragmentation and/or replication of relations. Although there are plans for it in every vendor's future, no distributed database product currently supports transparent horizontal and/or vertical fragmentation. If the use of fragmentation is a requirement, custom software must be written to support the level of transparency required.

On the other hand, most distributed database products currently support replicated data for query purposes; with the exception of two products (see section 2.3.4), however, these same products only support single site update within a single transaction.

2.3.2.2 Location Transparency

Within the context of a DDBMS, location transparency boils down to naming transparency – providing unique names for each object in the database. Implementations of this function range from requiring the user to provide unique names to having the system embed site location names within the name of each database object. Embedding locations in the object names can make it unwieldy when the user is required to specify the full name, as in IBM's experimental R* system. The embedding practice causes other problems when objects are moved across machines for performance optimization. Some systems elect to embed the "birth" site name in an object's name, providing referencing functions within the system's data dictionary that resolve the current location of the object. Other systems provide an aliasing capability for long names. However implemented, the best solution is for the system to provide unique internal names for database objects and to translate the user names to these transparently.

2.3.3 Distributed Query Processing

Distributed query capability can be found in just about every distributed database product, with significant differences occurring in the query manager. Some current distributed DBMS products send queries to each database, and then compile the results into one response rather than handoff the query to a distributed query manager. Some products contain sophisticated cost optimizers.

The location of the data dictionary is a significant factor in query optimization. Some systems fully replicate the dictionary at each site to expedite query processing; others maintain a centralized version of the dictionary with the emphasis on expediting updates. Depending on the specific application to be implemented, the designer must consider the dictionary's location(s). Using a product that supports a centralized dictionary has serious limitations for an application with hefty distributed query requirements.

Most query optimizers are tied to data transmission costs; a cost–based optimizer reviews all possible semijoins, determines the time and communications burden for each, and chooses the least cost alternative. The query method that minimizes network traffic is generally considered the most cost effective. One commercial product, Informix–STAR, has a verbose feature that informs the user of the costs involved for each SQL statement; however, it only reveals the costs for the chosen alternative, not for all the possibilities. The Ingres/STAR product boasts the industry's only "intelligent" optimizer; it relies on database sampling statistics and heuristics to arrive at an optimal query processing strategy.

Some databases take advantage of the parallelism offered in distributed databases by concurrently executing sub–queries at remote sites, and then bringing the data together in some optimal manner for final processing. Other products require that the processing be performed at the dataserver nearest the user; if the designer's network contains dataservers with significant performance differences, this is a serious concern.

2.3.4 Distributed Transaction Management

Distributed transaction management deals with the problem of concurrency – synchronizing transactions that update redundantly stored data. Transaction management protocols handle the commit/abort decision at each site in the distributed database. Fully implemented, these protocols require transaction logging, recovery, commit, and deadlock detection/prevention features.

The capability to read and update data located at multiple sites within a single transaction, preserving the properties of atomicity, isolation, and durability [18], may or may not be provided by a distributed database management system. How the database handles distributed concurrency control and commit protocols (without incurring excessive overhead costs while propagating control information), and the ability of the system to continue operation despite a component failure (ensuring that database consistency is not violated), determines the extent to which distributed transaction management is supported.

With the exception of Sybase and Ingres/STAR, most current distributed database management systems, even the customized implementations, restrict distributed update to a single site within a single transaction. The Sybase and Ingres products both support distributed updates that span multiple locations with a two–phase commit protocol; however, only Sybase supports multisite updates within one transaction with guaranteed recoverability. Both the research models and the commercial products list multisite−update in their future plans. Until then, if you require update at multiple locations you must either use the Sybase product, or develop custom software to fill the gap.

These two options are, in reality, closer than the reader might suspect. Sybase accomplishes multisite update by providing its users with a library of database functions to be used in developing distributed database applications. By incorporating the update, prepare, and commit function, the applications developer directs his own distributed transaction management.

2.3.4.1 Concurrency Control Protocols

The majority of commercially available general purpose distributed database management systems do not currently implement multisite update; updates are limited to either the local host or a another single host in the network.

The most well−known and widely used method for implementing concurrency control is two−phased locking in which transactions wanting to read data obtain a shared lock on the data item, and transactions wanting to write the data item obtain an exclusive lock. The granularity of the lock has been the subject of discussion and dispute; generally, locking occurs at the record, or tuple, level, with a few systems locking at the file, or "relation" level. Database systems that handle disk accesses themselves, rather than using I/O provided by the operating system, may lock at the "page" level.

The deadlock situation, in which two transactions each have a locked data item and are waiting for the other to release the lock, is generally handled via deadlock detection mechanisms.

2.3.4.2 Recovery Protocols

The most well−known and widely used method for implementing recovery protocol is the two−phase commit. During the first phase, the participating sites indicate the ability and willingness to commit; during the second phase, if all participants have answered affirmatively, the transaction is globally committed. If even one participant responds negatively, or fails to respond, the transaction is aborted at every site.

The successful implementation of the two−phased commit depends on a transaction logging function at each site during which log records containing information for undoing and redoing transactions is written to redundant, non−volatile storage. The two−phased commit is tolerant of failures as long as there is no loss of log information. Protocols exist that deal with those situations where sites fail during the ready−commit sequencing. One of the problems associated with the two−phased commit occurs when a communications failure or a failure of the site initiating a transaction occurs resulting in a partitioned network (see section 2.3.4.3). Some sites may be blocked while waiting for the commit/abort command. During this time system availability is affected by the held resources related to the blocked transaction. The practice of eliminating the "ready" phase by having sites transmit a "ready" immediately after executing the transaction exacerbates the blocking problem when a network or initiating site failure occurs.

Variations in the two−phased commit have been designed in an attempt to solve the blocking problem. The "presumed commit/abort" variation assumes a transaction is committed/aborted if no information about it is contained in the log. The "spooling" variation stores messages for a downed site at a predefined "spooling site". When the site recovers, it applies the spooled messages. Another variation directs recovering sites to look for lost information at other sites in the network.

2.3.4.3 Termination Protocols

Sites participating in a distributed database must have a consistent view of the network. If, because of a communications failure, the network becomes partitioned, sites in each partition will have a different "view", since all the sites in the other partition will appear to be down.

Addressing *how* sites deal with this type of communication failure, termination protocols handle the abortion of executing transactions. These protocols, which use the timeout mechanism, vary depending on the stage of the transaction, the kinds of communication permitted within the DDBMS, and whether it is the initiator of a transaction or a participant that has failed.

Considering the initiator, if a failure (timeout) occurs while waiting for the participants to respond with a commit/abort decision, then the transaction can be globally aborted. If a failure occurs while waiting for a commit or abort acknowledgement, the initiator can only continue to wait. For a participating site, if it has received an initial update message but never receives the prepare to commit or abort, it can abort the transaction. However, if a participant has voted to commit a transaction, but never receives a commit message from the initiator, it will be blocked from any further activity unless the system allows it to communicate with another participating site.

Blocking and non–blocking termination algorithms have been developed that deal with variations that may arise when sites are allowed to "discuss" their transaction states.

2.3.4.4 Reliability

Two related aspects of the reliability of distributed database systems are correctness and availability. These two factors are inversely related; imposing more of one results in less of the other. The trade–off needs to be evaluated by the designer of the system at hand.

For non–redundant data, availability depends strictly on the occurrence of site or network failures; there is no way to increase the reliability of the system. Increasing the availability of the system is a major goal when introducing redundant data into a distributed database system.

2.4 Implementation Strategies and Considerations

2.4.1 Degree of Site Autonomy

Despite Date's rule concerning site autonomy, reality may dictate varying degrees of autonomy. Given that each site maintains control of its own data, there may be compelling reasons for the existence, at some central site, for any of the following:

1. A *global catalog* responsible for maintaining information about fragmentation and allocation of data; an alternative where there is high probability of frequent catalog updates coupled with infrequent distributed queries

2. A central scheduler, or coordinating process, responsible for synchronizing access to the global database

3. A central deadlock detector to which local sites periodically report information relating to transactions waiting for resources; a simple detection mechanism, this may be a viable choice if the network has the capacity to carry the extra communications load and if the issue of failures related to the time it takes to transmit deadlock data to the central site has been considered

2.4.2 Lack of Standards

As in any field of engineering, a system's architecture defines its structure. Within the field of computer systems we try to establish some reference architecture that we term a "standard". Software developers may deviate from this reference, and in the past they have, but deviating in today's market is risky business.

Standards rely on proven and mature technology. The rapid innovation rate in this field makes standards obsolete before they can be established. Since the relational data model and some variant of SQL have been adopted by today's commercial DBMS products, these have been "standardized". Especially for heterogeneous distributed databases, standards for both language and remote access are essential. If the two-phase commit and two-phase locking protocols were standardized, implementations would be straightforward.

2.4.2.1 ANSI Standard SQL-2

All of the available distributed database products support some version of IBM's Structured Query Language (SQL). Although the American National Standards Institute (ANSI) established an SQL standard, SQL-86, in 1986, each product's version of SQL is different.

The SQL-86 standard has been vigorously attacked by E.F. Codd in at least three publications. The first two occurred in a two part article, "Fatal Flaws in SQL", appearing in the August and September, 1988 editions of *Datamation*. Codd reiterated and elaborated his complaints in his recent publication The Relational Model for Database Management/Version 2.[38] Three flaws, described by Codd as having "grave consequences" are these:

o SQL permits duplicate rows in relations

o It supports an inadequately defined kind of nesting of a query within a query

o It does not adequately support three-valued (or four-valued) logic

Since increasing numbers of businesses and government institutions are becoming dependent on relational DBMSs for the success of their operations, Codd believes these flaws must be repaired. He recommends that database users avoid duplicate rows within relations at all times, avoid nested versions of SQL statements whenever a non-nested version is possible, and take extra care when manipulating relations that have columns that may contain missing values.

The ANSI X3H2 Database Standards Committee is currently battling over the newly emerging standard called SQL-2. Embroiled in the battle, but not on the committee, are Codd and Date. Leading problems to be solved and/or negotiated are the following:

o Should duplicate rows be supported

o Should NULLs be supported

o Should primary keys be supported

o Datetime functions complicated by institution of Universal Time Coordinate (UTC) replacing Greenwich mean time

o Complexity added by updateable views

o Security issues associated with GRANT and REVOKE; REVOKE has been added and GRANT now permits circular references

The latest information indicates that the committee may be close to agreement with International Standards Organization's (ISO) working draft.[8] There is still a long way to go before the next SQL standard is available.

2.4.2.2 Remote Data Access

Remote access protocols support common communication mechanisms between local and remote processes. Two alternative approaches have been proposed: message passing and remote procedure calls.[31] Message passing consists of two primitives, send and receive, which, depending on their implementation, may provide reliable or unreliable communications. Remote procedure calls are a restricted form of message passing equivalent to blocking send and receive.

DBMS vendors have established proprietary interprocess communications protocols, but if heterogeneous distributed databases are to flourish standards must be established and followed. In 1985 an ISO working group was formed to work on Remote Access Standards.

2.4.2.3 ISO/OSI

The standardization necessary to interconnect heterogeneous hardware and support the transfer of data between them is provided by the Open Systems Interconnect (OSI) protocol family of the ISO. Can all the system functions of a distributed database management system be performed adequately at the applications layer? There have been suggestions in the research community that this may be the correct approach.

2.4.3 Distributed Database Tools

There is an acute need for automated tools to support distributed databases. Tools are required for each of the following:

- o Creation and maintenance of schemata
- o Design and maintenance of table & fragment locations
- o Measuring and monitoring system performance
- o Administering the dictionary
- o Global security administration

Other than the performance tool associated with the Ingres/STAR query optimizer (discussed earlier), no automated tools are provided with currently available distributed DBMS products. Developers of the one–of–a–kind systems also express a desire for automated design and measurement tools.

2.4.4 Planning for the Future

Implementing a homogeneous distributed database is a sizable effort; implementing a heterogeneous distributed database is a monumental task. Even with the newest commercially available database products, not all features of distributed database technology are available, with most products emphasizing those features that complement their particular market niche.

Toward building long–lasting database applications, and planning for upward migration, the database designer is urged to consider design strategies that insulate applications from changes that would otherwise be required by future releases of your underlying DBMS product that increase its distributed functionality. If your DDBMS does not support multisite update within a single transaction, provide custom software that makes it *appear to the application* that it is supported. If your DDBMS does not support table replication, supplement it with custom software that copies a remote table to the user's site transparently to the application. Follow Codd's advice regarding avoiding capabilities existing in SQL today that may not be there in the future. When the time comes that the missing feature is provided by the DDBMS, or standardization eliminates one that is there now, the custom software is removed and all applications may take advantage of the increased functionality without modification.

2.5 Summary of the State of the Art

Distributed database technology's chief advantage is the ability to access data faster and cheaper than the alternative centralized database approach. In order to make full use of this advantage, data must be able to be located transparently throughout the system, updates to the data must be synchronized, and queries of the data must be optimized to reduce not only the local disk accesses, but also the communications costs. Where the system involves heterogeneous databases, the system must be able to cope with various SQL dialects and remote procedure call protocols.

2.5.1 No Full Implementation

No current, commercially available distributed database product fully implements the concepts of distributed database technology. None address the problem of table fragmentation. However, potential users who carefully analyze their particular data requirements probably can find general purpose DDBMS software that will meet those requirements, although it may need to be augmented with custom software or additional hardware to compensate for deficiencies.

2.5.2 Market Pull versus Technology Push

The situation within the distributed database research and development community is currently one directed by market pull rather than technology push. The technology is going to advance based primarily on the needs of the users, rather than on any radical breakthroughs accomplished in the research labs.

Date's twelve rules spell out the requirements for implementing true distributed databases, and until those rules can be satisfied by a general purpose distributed database product, applications will not be able to take advantage of the full functionality offered by this technology.

2.6 Related Research Issues

Researchers in some technology domains that have traditionally studied and developed in isolation now find their technologies overlapping. Today, they are either joining ranks or are being forced into cooperation in order to produce solutions that meet the growing demand of user communities. The following topics are all technology areas that are being impacted by developments being made in distributed database technology.

2.6.1 Distributed Database Operating Systems

Distributed database systems run as user applications on top of a host operating system. Although the topic of distributed database operating systems has not been fully researched, there has been some discussion to the effect that the performance and functionality of DBMSs can be improved by modifying and enhancing the operating system to satisfy the additional requirements of DBMSs, particularly their transaction support, buffer management, and concurrency control requirements.

Enhancements and improvements have been implemented in special purpose "database operating systems" found in database machines, but not within the context of general purpose operating systems, although some research operating systems designs now include some of the required functionality. Areas where operating system change is contemplated are in the provision of the following:

o Fragmentation and replication transparency

o Network transparency

o User authentication and authorization control

o Full transaction management

o Special buffer and memory management

2.6.2 Distributed Multidatabase Operating Systems

Coordinating existing autonomous heterogeneous databases without attempting to integrate them with a unified schema is the subject of *multidatabase* or *federated* systems. These systems exhibit a high degree of autonomy and do not lend themselves to integration. The global schema in a multidatabase represents each local database separately. The user is presented a common data manipulation language with which he/she identifies the database to be used. Queries against a multidatabase generally are directed at only one of its components.

Some examples of autonomous databases that are federated for the purpose of information retrieval are dial–up information services such as CompuServe (TM) and The Source (TM). Dial–up services frequently guide the user through a sequence of queries to arrive at the required information.

The techniques developed for distributed databases will not suffice for multidatabases. Currently, the emphasis in this technology area is focusing on a common language to be used in managing information retrieval. The major commercial systems are involved and, through ISO and SQL Access standards, they will be able to cooperate in processing multidatabase queries.[25]

2.6.3 Object Oriented Distributed Databases

Object–oriented (OO) database systems are fast gaining support from the design communities. Where relational databases meet the demands of business applications typified by very large amounts of well–structured information, limited types and structures, and transactions that last for short lengths of time, the OO data model supports entities that are objects with functional characteristics and supports the requirement for dealing with long–lived transactions. The mathematical simplicity of relational database cannot support complex data types or programming language control structures. The OO data model provides a natural way to map real–world objects and their relationships directly to computer representations, meeting the data modeling requirements of applications such as computer aided design (CAD), computer aided manufacturing (CAM), computer aided software engineering (CASE), hypermedia and expert systems.

M.P. Papazoglou and L. Marinos[4] refute the position that the relational model is the model best suited for supporting distributed database applications. Concentrating on distributed heterogeneous information systems, they point out that the relational model does not adequately support the complex structures required and has limited semantic expression capabilities. The data model must "facilitate the communication between the users of diverse and incompatible information systems and assist ... with the uniform representation and integration of heterogeneous data from one site to another." [4] Defined in terms of an object–oriented data model which encapsulates the behavioral properties of the database objects, a distributed object–oriented database management system (doodms) maps the *data and the processing components* of the entire system into a unique system wide object space.

The doodms, as envisaged by Papazoglou and Marinos, consists of a layered umbrella over each autonomous DBMS. The umbrella is comprised of (1) a system language component, providing a system–wide query language in addition to each site's own query language, (2) metadata data modules, mapping local conceptual schemas into the distributed conceptual schema, and (3) the global transaction module which provides for distributed query decomposition and execution, concurrency control, and recovery.

There are no implementations, commercial or research, of a doodms, but its flexible, modular approach and its conformance to modern software engineering principles indicate that it will be forthcoming.

2.6.4 Distributed Knowledge Bases

Knowledge bases are relational databases extended with logic – the capability of deducing new information from existing information. Most of the technology required to implement distributed databases can be used to implement distributed knowledge bases; the consistency of the knowledge base and its query processing capabilities (especially recursive querying) being two of the the major issues.

Current trends toward the development of knowledge bases, removing the "intelligence" of artificial intelligence and expert systems from applications and placing it where it can be shared by many applications, is spurring database researchers to expand and extend their efforts in distributed database technology to meet the growing needs of knowledge bases.

3. SUMMARY

The data processing requirements of today's decentralized corporations together with advances in both database and network technologies has led to the emergence of distributed database technologies. Although no standards yet exist within this new technology, some guidelines have been provided by C.J. Date and E.F. Codd, both developers of relational database technologies.

A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network; a distributed database management system is a software system that permits the management of distributed data making the distribution transparent to the user. Distributed databases are more reliable and more responsive than centrally located and controlled databases; data can be entered where it is generated, data at different sites can be shared, and data can be replicated giving users the option of accessing copies of the data in the event of a site or network failure.

The fundamental issue of distributed databases is transparency, which, in a distributed database, refers to both the data and the network. However, achieving transparency must not infringe on the autonomous nature of each participating site.

Although several data models have been proposed for use with distributed databases, only the relational model has been implemented in current commercial products. The relational model contains simple data structures, provides a solid foundation for data consistency, and allows set–oriented manipulations of relations.

Distributed databases can be either homogeneous, where all participating local databases are based on the same data model (and are from the same vendor), or heterogeneous, involving databases belonging to several vendors probably not based on the same data model. Homogeneous distributed databases generally use a single, global schema, while heterogeneous distributed databases may opt for either a single, integrated schema or a federation of the local schemas.

To obtain the high degree of reliability and availability offered by distributed database technology, the relational tables must be fragmented and/or replicated across multiple sites. Fragmentation involves partitioning a relation either horizontally or vertically and allocating the partitioned relations to sites where the data is most often required. This practice is most useful in situations where fragments contain data with common geographical properties. Replication is a trade–off exercise between update costs and the benefits of increased reliability and performance. Since many factors contribute to an optimal fragmentation/replication design, no tools or algorithms have been developed to assist the designer in this task.

Distributed query capability can be found in just about every distributed database product, with significant differences occurring in the the performance capabilities of the query manager. Distributed query processing must deal with the analysis, optimization, and execution of queries referencing distributed data. The dynamic nature of a DDBMS increases the complexity of optimizing, since each site must also carry on its own local execution load, while the network is subjected to varying traffic patterns and bottlenecks. While past research has led to the development of query optimizers for centralized databases, these optimizers are designed with the goal of minimizing response time. Now they are being extended for distributed databases, with the objective of optimizing both response time and communication cost.

Managing transactions in a distributed database environment requires dealing with concurrency control, system reliability, and the efficiency of the system as a whole. Today's commercial DDBMS products use extensions of one or both of the same techniques used in centralized databases – locks and timestamps. Differences in DBMS products can be found in the both the granularity of the locks and in the particular implementation of the popular two–phase commit protocol. Most current implementations are unsatisfactory in situations involving large numbers of sites and high transaction volumes, and most restrict distributed update to a single site within a single transaction.

Despite the requirements for transparency and site autonomy, the lack of universally accepted standards and differences in the implementation of the data dictionary in various versions of distributed database systems have produced significant variations in the degree to which the requirements have been met. In order to select the right DDBMS or to develop an optimum distributed design, the database system designer must understand the relative merits of each feature and be able to make trade–offs to effectively match implemented features to the specific data needs to be supported.

The development of distributed database technology is stimulating the development of new applications that require support for distributed data. Advanced office automation systems, computer aided design systems, and knowledge based systems are three that profit from the ability to share data across a network of computers.

BIBLIOGRAPHY

1. "A Benchmark for Performance Evaluation of a Distributed File System", Anna Hac, **The Journal of Systems and Software**, Vol. 9 No.4, May 1989, pp. 273–285.

2. "A Homogeneous Relational Model and Query Languages for Temporal Databases", Shashi K. Gadia, **ACM Transactions on Database Systems**, Vol. 13 No. 4, December 1988, pp. 418–448.

3. "A Parallel Pipelined Relational Query Processor", Won Kim, Daniel Gajski and David J. Kuck, **ACM Transactions on Database Systems**, Vol. 9 No. 2, June 1984, pp. 214–242.

4. "An Object–Oriented Approach to Distributed Data Management", M.P. Papazoglou and L. Marinos, **The Journal of Systems and Software**, Vol. 11 No. 2, February 1990, pp. 95– 109.

5. "Application Design for Distributed DB2", Rob Goldring, **Database Programming & Design**, Vol. 33 No. 9, September 1990, pp 31–36.

6. "Architecture of Distributed Data Base Systems", Sudha Ram and Clark L. Chastain, **The Journal of Systems and Software**, Vol. 10 No. 2, September 1989, pp. 77–95.

7. "Cracks in the ANSI Wall", Joe Celko, **Database Programming & Design**, Vol. 2 No. 6, June 1989, pp. 66–69.

8. **Data Management on Distributed Databases**, Benjamin W. Wah, UMI Research Press, Ann Arbor, Michigan, 1981.

9. **Database System Concepts**, Henry F. Korth and Abraham Silberschatz, McGraw–Hill Book Company, New York, N.Y.,1986.

10. "DATAPLEX: An Access to Heterogeneous Distributed Databases", Chin–Wan Chung, **Communications of the ACM**, Vol. 33 No. 1, January 1990, pp. 70–80.

11. "DB2 v. 2.2: A Few More Bells & Whistles", Craig S. Mullins, **Database Programming & Design**, Vol. 3 No. 6, June 1990, pp. 59–61.

12. **Distributed Databases**, Edited by I.W. Draffan and F. Poole, Cambridge University Press, London, England, 1980.

13. "Distributed Databases", Herb Edelstein, **DBMS**, Vol. 3 No. 10, September 1990, pp. 36–48.

14. "Distributed Database for SAA", R. Reinsh, **IBM Systems Journal**, Vol. 27, No. 3, 1988, pp. 362–369.

15. **Distributed Database Management Systems**, Olin H. Bray, D.C. Heath and Company, Lexington, Massachusetts, 1982.

16. **Distributed Databases Principles and Systems**, Stefano Ceri and Giuseppe Pelagatti, McGraw Hill Book Company, New York, N.Y. 1984.

17. "Distributed File Systems: Concepts and Examples", Eliezer Levy and Abraham Silberschatz, **ACM Computing Surveys**, Vol. 22 No. 4, December 1990, pp. 321–374.

18. "Divide and Conquer Your Database", Dennis Livingston, **Systems Integration,** Vol. 24 No. 5, May 1991, pp. 43–45.

19. "Does Client–Server Equal Distributed Database?", Beth Gold–Bernstein, **Database Programming & Design**, September 1990, pp.52–62

20. "Dynamic File Migration in Distributed Computer Systems", Bezalel Gavish and Olivia R. Liu Sheng, **Communications of the ACM**, Vol 33. No.2, February 1990, pp. 177–189.

21. "Emerald Bay's Quiet Return", Lan Barnes, **DBMS**, Vol. 3 No. 11, October 1990, pp. 50–57.

22. "Experiences with the Amoeba Distributed Operating System", Andrew S. Tanenbaum, Robbert van Renesse, Hans van Staveren, Gregory J. Sharp, Sape J. Mullender, Jack Jansen and Guido van Rossum, **Communications of the ACM**, Vol. 33 No. 12, December 1990, pp. 46–63.

23. "Heterogeneous Distributed Database Systems for Production Use",Gomer Thomas,et.al., **ACM Computing Surveys**, Vol. 22 No. 3, September 1990, pp. 237–266.

24. "Heterogenous Processing: A 4GL Case Study", Michael M. David, **Database Programming & Design**, Vol. 4 No. 3, March 1991, pp.27–34.

25. **Intelligent Databases**, Kamran Parsaye, Mark Chignell, Setrag Khoshafian, and Harry Wong, John Wiley & Sons, Inc, New York, N.Y. 1989.

26. "Maintaining Availability in Partitioned Replicated Databases", Amr El Abbadi and Sam Toueg, **ACM Transactions on Database Systems**, Vol. 14 No. 2, June 1989, pp. 265–290.

27. "Multiple–Query Optimization", Timos K. Sellis, **ACM Transactions on Database Systems**, Vol. 13 No. 1, March 1988, pp. 23–52.

28. "Object–Oriented Databases: Design and Implementation", John V. Joseph, et al, **Proceedings of the IEEE**, January 1991, pp. 42–64.

29. "On the Foundations of the Universal Relation Model", David Maier, Jeffrey D. Ullman and Moshe Y. Vardi, **ACM Transactions on Database Systems**, Vol. 9 No. 2, June 1984, pp. 283–308.

30. "On the Management of Long−Living Transactions", K. Brahmadathan and K. V. S. Ramarao, **The Journal of Systems and Software**, Vol. 11 No. 1, January 1990, pp. 45−52.

31. **Principles of Distributed Database Systems**, M. Tamer Ozsu and Patrick Valduriez, Prentice Hall, Inc, Englewood Cliffs, New Jersey 1991.

32. "Relational Database Design Using an Object−Oriented Methodology", Michael R. Blaha, William J. Premerlani and James E. Rumbaugh, **Communications of the ACM**, Vol. 31 No. 4, April 1988, pp. 414−427.

33. **Relational Database Technology**, Suad Alagic, Springer−Verlag, New York, N.Y., 1986.

34. "SQL2 An Emerging Standard", Jim Melton, **Database Programming & Design**, Vol. 3 No. 11, November 1990, pp. 25− 32.

35. "Strategic Database Planning", G. Lawrence Sanders, **Database Programming & Design**, Vol. 3 No. 11, November 1990, pp. 52− 56.

36. "The Case for Object−Oriented Databases", Thomas Atwoord, **IEEE Spectrum**, February 1991, pp. 44−47.

37. "The Raid Distributed Database System", Bharat Bhargava and John Riedl, **IEEE Transactions on Software Engineering**, Vol. 15 No. 6, June 1989, pp. 726−736.

38. **The Relational Model for Database Management/Version 2**, E.F. Codd, Addison−Wesley Publishing Company, Reading, Massachusetts, 1990.

39. "The Trouble with Two−Phase Commit", George D. Tillmann, **Database Programming & Design**, Vol. 3 No. 9, September 1990, pp. 64−70.

40. "Transaction Processing Monitors", Philip A. Bernstein, **Communications of the ACM**, Vol. 33 No. 11, November 1990, pp. 75−86.

41. **Tutorial: Distributed Database Management**, Philip A. Bernstein, James B. Rothnie, David W. Shipman, IEEE Publishing Services, New York, N.Y., 1978.

42. "Update and Retrieval in a Relational Database Through a Universal Schema Interface", Volkert Brosda and Gottfried Vossen, **ACM Transactions on Database Systems**, Vol. 13 No. 4, December 1988, pp. 449− 485.

43. "Why Choose Distributed Database?", Michael Krasowski, **Database Programming & Design**, Vol. 4 No. 3, March 1991, pp. 46−53.

APPENDIX A: GLOSSARY

atomicity Either all or none of a transaction's operations are performed.

bridge Network hardware that serves to restrict packets to a local segment of a network.

broadcast network A network in which all sites receive all the messages sent by another site; a mechanism (typically a prefix containing an identification of the destination site) allows each site to recognize those messages directed to it.

catalog A repository of information about a database including, in distributed databases, the description of the fragmentation and allocation of data and the mapping to local names.

concurrency control Ensures transaction atomicity in the presence of concurrent execution of transactions.

data distribution Refers to the partitioning, fragmentation, replication, and allocation of data among the sites participating in a distributed database.

data dictionary The major database module that contains database *metadata*; it includes, at a minimum, schema and mapping definitions.

data manipulation language A language that enables users to access or manipulate data organized by a data model. A *procedural language* requires the user to specify what data is needed and how to get it; a *nonprocedural* language requires the user to specify what data is needed without specifying how to get it.

data mapping Data types or data values are converted for conformity with each other.

data model A collection of conceptual tools for describing data, data relationships, data semantics, and consistency restraints.

deadlock A circular waiting situation which arises when two or more transactions obtain exclusive locks on one or more data resources and are waiting for a resource held by another waiting transaction.

deadlock avoidance Methods that employ either concurrency control techniques that never result in deadlocks or require schedulers to detect potential deadlock situations in advance and ensure that they will not occur.

deadlock detection The detection of a state of deadlock and the preemption and abortion of one (or more) transaction(s) until processing may continue.

deadlock prevention Method that guarantees deadlock cannot occur; all data items required for a transaction are predeclared and must be accessible before the transaction is initiated.

distributed database A database system that provides access to data located at multiple sites in a network.

distributed processing Based on a collection of programs that are distributed among sites in a network, permits a program at any site to invoke a program at another site in the network as if it were a locally resident subprogram.

distributed query processor A distributed database system module that, given a query, determines an execution strategy that minimizes a system cost function which includes I/O, CPU, and communication costs.

dump An image of a previous state of a database, usually stored on offline storage.

durability Once a transaction is committed, the results of its operations will never be lost, independent of subsequent failures.

Ethernet An example of a packet–switched network in which packets may vary in size from 64 bytes to 1,518 bytes and operates at 10 megabits per second.

fragment groups Collections that include the primary fragment and those fragments resulting from derived fragmentation.

fragmentation Dividing a global relation into subrelations and allocating the subrelations to sites participating in the global database.

fragmentation, horizontal Data items in different local databases may be identified as logically belonging to the same table in the global database; a relation partitioned along its rows.

fragmentation, vertical Data items in different local databases may be identified as logically representing the same row in the global database but containing different attributes for the row; a relation partitioned into smaller relations.

gateway Network software that permits the movement of information between networks of differing communications protocols.

homogeneous database Refers to a distributed database in which each physical component runs under either the same database management system or, at least, the same data model.

heterogeneous database Refers to a distributed database in which not all physical components run under the same database management system; some literature refers to a distributed database as being *heterogeneous* if the local nodes have different types of computers and operating systems, even if all local databases are based on the same data model and even the same database management system.

ISO International Standards Organization.

isolation An incomplete transaction cannot reveal its results to other transactions before its commitment.

local autonomy Refers to the amount of control exercised by local database administrators within a distributed database environment; local administrators with total control over that part of a distributed database at their sites are said to be autonomous.

local recovery manager Module of a distributed database management system (one exists at a local site) responsible for implementing local procedures by which the local database can be recovered to a consistent state following a failure.

long–lived transaction Data that lives after the processes that created them terminate.

object–based logical model A data model used in describing data at the conceptual and view levels. The conceptual level describes what data is stored in the database and what relationships exist among the data and the view level restricts the conceptual level to part of the database. Object–based logical models allow the explicit specification of data constraints.

OSI Open Systems Interconnect.

packet–switched network A network in which messages are broken up into packets and each packet is transmitted individually. The packets may travel independently and may take different routes.

point–to–point network A network in which sites are connected by communications channels, typically telephone lines. Leased circuit utilizes point–to–point technology along leased communications lines.

record–based logical model A data model used in describing data at the conceptual and view levels where the conceptual level describes what data is stored in the database and what relationships exist among the data and the view level restricts the conceptual level to part of the database. Record–based logical models can be used to specify the overall logical structure of the database, but do not provide for specifying data constraints.

replication Data items in different local databases may be identified as copies of each other.

router Network hardware that picks the optimal route to send traffic over a network.

scheduler Module of a distributed database management system responsible for the implementation of a specific concurrency control algorithm for synchronizing access to the database.

schema Describes the database as it is stored; describes the physical format, storage locations, and access paths and defines the logical structure.

schema, global Describes all the data in a distributed database

schema, local Describes the data at the local sites in a distributed database

serializability If several transactions are executed concurrently, the result must be the same as if they were executed serially in some order.

timestamp Uniquely identifies a transaction; for two transactions A and B, if A occurred before B then the timestamp of A is less than the timestamp of B.

transaction A sequence of operations which either are performed in entirety or are not performed at all; an atomic unit of execution

transaction manager Module of a distributed database management system responsible for coordinating the execution of the database operations on behalf of an application.

transparency Also called *data independence*, refers to the independence of application programs from the physical or logical organization of the data.

transparency, distribution Refers to the independence of application programs from the physical location of the data in a distributed database.

transparency, fragmentation Refers to the lack of awareness by application programs of the existence of fragmented relations in a distributed database.

transparency, replication Refers to the lack of awareness by application programs of the existence of replicated data in a distributed database.

two–phase commit A protocol that requires for each transaction a first phase during which an abort/commit decision is made by each participant and a second phase during which the decision is implemented.

two–phase locking A locking protocol that requires for each transaction a first phase during which new locks are acquired and a second phase during which locks are only released.

APPENDIX B: VENDORS IN DISTRIBUTED DATABASE TECHNOLOGY

ASK Computer Systems Inc., Ingres Products Division, 1080 Marina Village Parkway, Alameda, California 94501, (415) 769–1400

Computer Associates International Inc., 711 Stewart Avenue, Garden City, New York, 11530, (516) 227–3300

Digital Equipment Corporation, Database Systems Group, 55 Northeastern Blvd., Nashua, Hew Hampshire 03062, (603) 884– 2423

Gupta Technologies Inc., 1040 Marsh Road, Menlo Park, California 94025, (415) 321–9500

IBM Corporation, Old Orchard Road, Armonk, New York 10504, (914) 765–1900

Informix Software Inc., 4100 Bohannon Drive, Menlo Park, California 94025, (415) 926–6300

Interbase Software Corporation, 209 Burlington Road, Bedford, Massachusetts 01730, (800) 245–7367

Microsoft Corporation, 1 Microsoft Way, Redmond, Washington 98052, (206) 882–8080

Ontologic Inc., Billerica, Massachusetts

Oracle Corporation, 500 Oracle Parkway, Redwood Shores, California 94065, (415) 506–7000

PeerLogic Inc., 333 DeHaro Street, San Francisco, California, 94107, (415) 626–4545

Progress Software Corporation, 5 Oak Park, Bedford, Massachusetts, 01730, (617) 275–4500

Ratliff Software Production Inc., 2155 Verdugo Blvd., Suite 20, Montrose, California 91020, (818) 546–3850

Revelation Technologies, 2 Park Avenue, New York, New York, 10016, (617) 275–4500

Saros Corporation, 10900 N.E. 8th Street, Bellevue, Washington 98004, (206) 646–1066

Sybase Inc., 6475 Christie Avenue, Emeryville, California 94608, (415) 596–3500

Tandem Computers Inc., 19333 Valico Parkway, Cupertino, California 95014, (408) 965–7542

WordTech Systems Inc., 21 Altorinda Road, Orinda, California 94563, (415) 254–0900

XDB Systems, 14700 Sweitzer Lane, Laurel, Maryland 20707, (301) 317–6800