Research Directions for Distributed Databases

Hector Garcia-Molina
Department of Computer Science
Princeton University
Princeton, NJ 08544

Bruce Lindsay
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

1. Introduction

Communication networks make it feasible to access remote data or databases, allowing the sharing of data among a potentially large community of users. There is also a potential for increased reliability: when one computer fails, data at other sites is still accessible. Critical data may be replicated at different sites, making it available with higher probability. Multiple processors also open the door to improved performance. For instance, a query can be executed in parallel at several sites.

We have so far avoided the term *distributed database*. For some people, this term implies a particular type of distributed data management system where users are given transparent, integrated access to a collection of databases. In other words, a user is given the illusion of a single database with one global schema. Queries on this database are automatically translated into queries on the underlying databases. In the early days (before 1980) this was thought to be the ultimate goal for all distributed data management systems, and hence the term distributed database became associated with transparency and integration. Nowadays most researchers agree that transparency and integration may be incompatible with requirements for autonomy and diversity of implementations. They are using the term "distributed database" in a more general sense to mean a collection of possibly independent or federated database systems. Each system has some set of facilities for exchanging data and services with other members. In this paper we take the broader meaning of distributed databases in order to cover a wider spectrum of the challenging problems facing researchers.

This paper forms part of a collection of articles on current and future research issues in the database area. Since it is impossible to cleanly partition research areas, it is natural to expect overlap between the articles. In our case the overlap is more significant because two of the most important distributed database issues are being discussed in separate articles: heterogeneous and parallel databases. Many of the topics covered by other articles also have strong connections to distributed databases: security is especially critical in a distributed environment, scientific databases are often distributed, future DBMS architectures must have distribution in mind, etc.

In an attempt to reduce overlap, in this paper we will focus on distributed database issues that are not central to the other papers in this collection. Thus, we stress that our coverage here will be incomplete. Of course, even for the remaining issues, our discussion must be viewed as illustrative, not comprehensive. We are simply trying to point out some research areas that in the author's opinion have potential. For this, we have grouped our ideas into four broad areas and covered each in one of the following sections.

Before starting we would also like to clarify that due to space limitations this will not be a survey of relevant papers and work. We will actually avoid making references, for as soon as one reference is made, for fairness others must follow. Interested readers may refer to a Special Issue of the IEEE Proceedings (May 1987) on Distributed Databases. It contains many valuable references, as well as a discussion of state of the art distributed database processing. Current database textbooks are also a good source of references.

2. Distributed Data Architecture

Consider a user local to a database management system. Consider also a second remote database that the user wishes to access. How should the local system present the remote data? As discussed in the introduction, under a transparent, fully integrated architecture, the remote database is made to appear as part of the local one. Operations on the remote data, e.g., joining a remote table with a local one, can be done (at least from the point of view of the user) as easily as fully local operations. At the other end of the spectrum, the remote site may simply offer a set of services that may be invoked by explicit calls. For example, if the remote computer handles an airline database, it may let a user request the schedule for a given flight or reserve a seat on a flight.

Transparency is not an all or nothing issue. It can be provided at various levels, and each level requires a particular type of agreement between the participants. In the fully transparent case, the sites must agree on the data model, the schema interpretation, the data representation, the available functionality, and where the data is located. In the service (non-transparent) model, there is only agreement on the data exchange format and on the functions that are provided by each site.

The tradeoffs involved with providing or not transparency revolve around simplicity of access and ability to integrate data from diverse sources versus issues of site autonomy and specialized functions. Clearly, from the point of view of a user desiring remote access, a transparent architecture is desirable. All the data at the remote site is accessible, just as if it were local. However, from the point of view of the administrator of the remote site, transparency provides access that is difficult to control. The remote site could only make visible certain views on its data, but view mechanisms in many systems are not powerful enough to provide the desired protection. For instance, at a bank site funds transfer may only be allowed if the account balance is positive and the customer has a good credit rating. A simple view mechanism cannot express this.

It is much easier to provide these checks within a procedure that is called remotely. Although the data may be freely accessible to local users, remote users see the data encapsulated by a set of procedures, much like in an object oriented

programming environment. This type of remote service or federated architecture is simpler to implement than full transparency. Less agreement is needed between the participants, and complex algorithms such as a multi-site join need not be implemented. Sites have greater autonomy to change the services they provide or how they provide them.

The research challenge in this area is to fully understand the spectrum of alternatives. While we have sketched the two extreme solutions (full transparency and a service model), the intermediate models are not well defined. The fundamental issue is the *level* at which remote requests and responses are exchanged. Great care is needed to avoid weakening remote access functionality to the lowest common denominator while, at the same time, avoiding a proliferation of service and implementation specific protocols. Fruitful research directions include extending data access and manipulation protocols to support database procedures (to encapsulate services and policies), authentication standards, and relaxed serializability levels (with special authorization required for full serializability). In addition, further research is needed on technologies for exporting type definitions and *behavior* to allow remote users to exploit the semantic content of retrieved information (i.e., object distribution).

3. Problems of Scale

Current trends indicate that the number of databases is rapidly growing, while at the same time their size is also increasing. Some database and distributed data algorithms do not scale up nicely as the number of components in the system grows. In a conventional database, for example, one may need to place data off-line to make a backup or for reorganization. Typically, this is done during the night. As the database grows in size, the backup or reorganization time grows, and a night is no longer long enough.

In a distributed database, one is faced with such problems of scale as individual databases grow, and also as the number of databases and the scope of the system grows. For instance, in a world-wide distributed system, there is no night time to do reorganizations or backups. Key system algorithms may break down in larger systems. For example, in a small system, it may be feasible to search for a particular file of interest by broadcasting a request to all nodes. In a very large system, this becomes impractical. Having a central directory of all resources is also a bad idea, not just because of its large size, but because it is prone to failures and because not all sites may want to advertise their resources to everyone. Thus, the problem of resource finding in a very large distributed data system is quite challenging. When one starts a search, one not only does not know where the resource is, but one does not know what directories are available for this type of resource.

As an illustrative example, consider a scientist searching for a database of ozone readings over antarctica for the year 1980. (For one thing, "antarctica" and "ozone" are denoted differently in Russian databases.) Different organizations have directories of their own databases, but there is no reliable directory of organizations. Heterogeneity is an added complication here: it is not clear how to make our query understandable to different organizations, and if a relevant database is found, how to know what it really contains, expressed in our terms. While some progress has been made with yellow and white pages servers, the mechanisms for describing data resources in human

or machine readable form are quite crude. One only needs to try to use today's bibliographic search systems to realize that capturing and encoding the semantic or technical essence of data collections is not well advanced.

In addition to the resource finding protocols, there are other algorithms that may not scale up. The challenge is to identify them and to find alternatives. For instance, what deadlock detection, query processing, or transaction processing algorithm will work best in a very large system, or in a system with very complex queries, or with many participants in a transaction?

Administration of large distributed databases is another problematic area. As the number of components, users, and transactions rapidly grows, it becomes harder and harder to manage the system effectively. The volume of accounting, billing, and user authentication information grows. The number of choices for improving or speeding up a system grows. The size and number of database schemas grows. It becomes harder to evaluate the performance of the system and to predict its behavior under changes. Upgrading or installing new software also is harder, as there are more sites that are affected and it is impractical to halt the entire system to do an operating or database system upgrade. A related problem is the management of the underlying computer communication network. The problems are analogous: handling growing information on links, protocols, and performance. Also, key network algorithms do not scale up, e.g., those for initializing a network after its failure.

4. Information Management

Distributed systems are increasingly used for retrieving information on a wide variety of topics, ranging from stock prices, to cars for sale; from news items to jokes. In some cases the information is stored in commercial database services and a fee is paid. In other cases, computers are interconnected in ad-hoc networks and information exchanged between end-users, as in "net-news" or in bulletin boards. In still other cases, the communications companies themselves are providing information services. This is the case of MiniTel in France. Traditionally, these information networks have not been considered a distributed database. However, there is no reason why the mechanisms developed for formal distributed databases could not be extended to informal and/or loosely coupled information repositories.

In simple terms, the problem of information management is one of matchmaking. On one hand we have a user desiring some type of information (e.g., who has found this bug in this program?). On the other hand we have suppliers of information that may fully or partially match those needs. The goal is to make a connection between the supplier and the consumer. The problem is related to that of resource finding described in Section 3, but there are added complications.

Sometimes information requests are not for a single item, but are "standing orders", e.g., send me all future news items on this topic. This means the system must not only find the appropriate sources, but it must also set up the data paths for ongoing communication. Batching data transmissions is also important. For example, if two users at neighboring computers request the same information, it may be more effective to route the information to one computer, and to then forward it to the other. Existing systems such as net-news provide batching, but they make many restrictions as to what users may read and when they can read.

A provider of information often wishes not only to track who has received it, but also may need to be able to control how it is to be used. This will require facilities for access tracking and for "contracting" with the recipient. Important social, legal, political, and scientific issues must be addressed before open information distribution systems can be used for anything other than the exchange of "trivial" information.

Existing information systems usually do not provide reliability. In particular, a user may miss information if his machine is down. Thus, one challenge is to incorporate exiting distributed database crash recovery technology into information networks.

Distributed data and information can also be used in non-traditional ways, i.e., more than simply submitting "queries" and getting replies. For instance, electronic newsletters with user contributions are one such interaction. Users submit articles or news items to an editor or a group of editors. The editors check the articles, trimming them or eliminating uninteresting ones. The accepted articles are then distributed on the network to "subscribers" or are made available for queries. The National Science Foundation has recently suggested a "colaboratory," a distributed electronic laboratory where researchers can share information and conduct their research. Clearly, shared, distributed data must play a critical role here. The challenge is to expand the models and mechanisms of distributed database to encompass these new applications.

5. Transaction Models

In a federated database architecture, computers provide services to other sites. The glue that ties together the system is the transaction management system. It coordinates a sequence of interactions with different nodes, providing data consistency and atomicity.

Conventional transaction models based on locking and two phase commit may be inadequate. One reason is that they force participants to use the same model, and to possibly give up their autonomy. For instance, a participant in a two phase commit must become a subordinate of the coordinator(s), not releasing resources held by a transaction (locks) until instructed to do so. Another problem is that a large transaction may last for a long time, holding up critical resources. In a sense, this is a problem of scale: as the number of participants in the protocol grows, or that amount of work each participant must do grows, the time that resources are tied up also grows. This is unacceptable if these are critical, often accessed resources.

The need for relaxed concurrency control protocols in the local case has been recognized in some products and standards proposals. For distributed systems, there have already been numerous proposals for weaker transaction models that give participants more autonomy and improve performance, while still guaranteeing some basic correctness properties. For example, a sequence of steps at various sites can be considered a saga and not a full fledged transaction. After each step completes, a local transaction commits. If the saga needs to be aborted later on, a compensating step is run at nodes where transactions committed. This eliminates the need for two phase commit. However, sagas can now see intermediate results of other sagas, so programming such applications may be trickier. Also, the need for compensating steps creates more work for the application programmers. Tools for developing applications in a saga environment would be a very valuable contribution.

Without global transactions (as is the case with sagas and similar approaches), only consistency constraints local to a single site are maintained. Global constraints, e.g., object X at site A must be a copy of object Y at site B, are not necessarily guaranteed. If the inter-site constraints are known, it is possible to maintain them in an "approximate" way, e.g., making X approximately equal to Y or X equal to a relatively recent value of Y. Such approximate constraints may be adequate for some applications, e.g., an ATM machine may not need to know precisely how much money a customer has in his account; a rough estimate may be enough to make the decision if funds can be withdrawn. Approximate constraints may make it possible to operate without two phase commit for many steps or sub-transactions, improving autonomy and performance.

In general terms, there is a need for more options for transaction management in a distributed database. For each option it is necessary to precisely define the type of correctness it provides, and to evaluate the performance and autonomy it yields.

6. Conclusions

In order to extend data distribution to large scale environments, the requirements and characteristics of different DBMS implementations and information collecting organizations must be addressed. The challenge is to support integration of information from diverse sources without retreating to a low-function common protocol (e.g. NFS) while, at the same time, avoiding a proliferation of high-level, service-specific interfaces. Future research should consider carefully the role of remotely invoked procedures (which might have interfaces similar to other, general data accessing interfaces) as a mechanism to respond to organizational autonomy and control issues. Such interfaces could encapsulate local control procedures but execute in the context of a larger information interchange environment (e.g., authenticated user, transactions, accounting, data exchange formats, etc.).

We also believe that any success in providing distributed information processing facilities in a heterogeneous environment will probably rely, ultimately, on standard protocols for authentication, accounting / billing / contracting, data access specifications, schema exchange, typed object transport, and information resource characterizations. Accommodating multiple data models will increase the difficulty of developing mechanisms for information exchange and integration. The alternative is to provide the user with an array of information accessing tools, each of which can extract or manipulate data in a single type of DBMS using a unique user interface and communication protocol. Information integration from different sources in such an environment would be the user's problem.

Finally, we believe that one fruitful direction for data distribution technology is the extension of data semantics. This means that mechanisms for defining type specific behavior (procedures, functions, or methods) should, somehow, be extended to allow data objects to retain their semantic qualities as they are transported from one location to another. In some sense, this challenge echoes the earlier efforts to develop seamless functional distribution for the data model and its operations. Without techniques for transmitting objects without loss of their semantic qualities, information is pinned to the environment in which it is created and cannot interact (be integrated) with information (objects) from other cultures (environments).