An Overview of Security in Distributed Database Management System

D.Arockia Sathis Kumar1 M.Natarajan2
1Research Scholar (M.Phil.) 2Assistant Professor
1,2Department of Computer Science

Thanthai Hans Roever College Perambalur-621212, India

Abstract— Databases are exceptionally familiar in businesses. The ordinary database is naturally held on a central server and people log in to the system to query or update the database. Still, there is an additional type of database known as a distributed database that bids advantages for some kinds of organization. This paper will examine the underlying features of the distributed database system and its security. Learning the task of distributed database management system will lead us to a successful design. Developing a successful distributed database system requires to address the importance of security issues that may arise and possibly compromise the access control and the integrity of the system. This paper propose some solutions for some security aspects such as multilevel access control, confidentiality, reliability, integrity and recovery that pertain to a distributed database system.

Key words: Distributed Database Management System, Security in Distributed Database, Centralized or Decentralized Authorization, Integrity, Elements of Distributed Database Security

I. INTRODUCTION

Databases have been common in government departments and commercial enterprises for many years. Today, databases in any organization are increasingly opened up to a multiplicity of suppliers, customers, partners—and employees - an idea that would have been unheard of a few years ago. Numerous applications and their associated data are now accessed by a variety of users requiring different levels of access via manifolds devices and channels - often simultaneously [1]. For example:

- On-line banks allow customers to perform a variety of banking operations - via the Internet and over the telephone – whilst maintaining the privacy of account data.
- E-Commerce merchants and their Service Providers must store customer, order and payment data on their merchant server -and keep it secure.
- HR departments allow employees to update their personal information – whilst protecting certain management information from unauthorized access.
- The medical profession must protect the confidentiality of patient data – whilst allowing essential access for treatment.
- On-line brokerages need to be able to provide large numbers of simultaneous users with up-to-date and accurate financial information.

This complex landscape leads to many new demands upon system security. The global provide mechanisms to segregate environments; perform integrity checking and maintenance; enable strong authentication and non-repudiation; and provide for confidentiality. In turn, this necessitates comprehensive business and technical risk assessment to identify the threats, vulnerabilities and impacts, and from this define a security policy. This leads to security definitions throughout the infrastructure - operating system, database management system, middleware and network [1].

As distributed networks become more popular, the need for improvement in distributed database management systems becomes even more important. A distributed system varies from a centralized system in one key respect: The data and often the control of the data are spread out over two or more geographically separate sites. Distributed database management systems are subject to many security threats additional to those present in a centralized database management system (DBMS). Furthermore, the development of adequate distributed database security has been complicated by the relatively recent introduction of the object-oriented database model. This new model cannot be ignored. It has been created to address the growing complexity of the data stored in present database systems [2].

II. DISTRIBUTED DATABASE

A distributed database is a single logical database whose parts are physically located in more than one place. Each location is accessible across a network. A standard database on a network looks like this:

Fig. 1: Standard database

This shows a central database being accessed from a workstation over a network. Note that all traffic has to flow over the network, and this may be a problem. A typical distributed database looks like this:

Fig. 2: Distributed database

In this case the database is split across different sites and each site is connected over a network. Each site has a database management system (DBMS) that allows queries and updates to be made locally. But each site also has a distributed database component (DDBMS) that allows each site to make global queries of the whole system.

A. Use of Distributed Database:

Many organizations have branches and offices located across the country or even the world. For example
- A travel company with many branches across the country
-- An international car maker with dealers across the world
-- A supermarket stock system with many branches

It may be that each branch only needs to access their own data 90% of the time and only occasionally have a need to access remote information. On the other hand the central office needs to have an up to date version of all the data as well. With a single central database, all queries and updates have to travel across a network. This may cause a heavy load on the system. It also means if the central database breaks or the network goes down, the entire organization is affected.

A distributed database solves some of these issues. With a distributed database, the central database is updated over night as a batch operation, but day-to-day work in the branches is carried out using the local portion of the database. If the network fails, then they can carry on as normal until it is fixed

B. Types of Distributed Database:
There are three main options when putting together a distributed database.
1) Duplicated Database
Duplicated database at each location with this system, a local copy of the entire database is kept at each location. As shown below. This is often not practical for a number of reasons:

Fig. 3: Duplicated database
a) Advantages:
If the database is fairly small, then this may be a practical solution. Considering the issues with the duplication method, the configuration below is another option
b) Disadvantages:
  -- Very heavy use of the network.
  -- Whole database may be huge requiring large storage requirement at each location
  -- Difficult to keep all copies in synchronization
  -- It does not scale well as the database grows.
  -- Wasteful if only a small area of the database is needed by each branch

2) Partitioned database
The distributed database is split or ‘partitioned’ into convenient data sets depending on the specific needs of the organization. For example, consider a supermarket company that has dozens of large stores dotted around the country and it needs an efficient stock control database. Solution: Each store has its own section of the distributed stock control database. They keep their part of the system up to date. Each night, a central database is synchronized with records from each store.

Fig. 4: Partitioned database

a) Advantages:
  -- Each store is independent. Problems in one store does not affect any other store
  -- Network load is much less, as only the central database needs to be kept in sync
  -- High performance as there is no network bottleneck.
  -- Very good solution if each store is largely independent
b) Disadvantages:
  -- Not so useful if each branch constantly needs information from other branches. For example a national holiday booking company where each branch needs to check the status of the holiday packages
  -- The database must be carefully partitioned to keep each section as local as possible

3) Partitioned + Index
A modification of the partitioned database is to include an index of all remote database records as well. As shown below

Fig. 5: Partitioned + index
This still partitions the database to keep all records as local as possible, but each store has an index pointing to records in all other databases. A central database keeps the indexes updated on a nightly batch run.

a) Advantage
  -- Still high performance as most queries and updates remain local
  -- Can efficiently access remote records by using the index rather than a network query to a central database
b) Disadvantage
  -- The system must now keep all indexes up to date
  -- More complicated

III. SECURITY OF A DISTRIBUTED DATABASE
Security is more difficult to maintain compared to a central database. Database management systems normally run on top of an operating system and provide the security associated with a database. Typical operating system security features include memory and file protection, resource access control and user authentication. Memory protection prevents the memory of one program interfering with that of another and limits access and use of the objects employing techniques such as memory segmentation.

The operating system also protects access to other objects (such as instructions, input and output devices, files and passwords) by checking access with reference to access control lists. Security mechanisms in common operating systems vary tremendously and, for those that are lacking,
there exists special-purpose security software that can be integrated with the existing environment. However, this can be an expensive, time-consuming task and integration difficulties may also adversely impact application behaviors. Most database management systems consist of a number of modules including database querying and database and file management along with authorization, concurrent access and database description tables.

These management systems also use a variety of languages: a data definition language supports the logical definition of the database; developers use a data manipulation language; and a query language is used by non-specialist end-users [3].

IV. ELEMENTS OF DISTRIBUTED DATABASE SYSTEM SECURITY

A. General Database Security Concerns:
The distributed database has all of the security concerns of a single-site database plus several additional problem areas. We begin our investigation with a review of the security elements common to all database systems and those issues specific to distributed systems [4]. A secure database must satisfy the following requirements (subject to the specific priorities of the intended application):

1) It must have physical integrity (protection from data loss caused by power failures or natural disaster),
2) It must have logical integrity (protection of the logical structure of the database),
3) It must be available when needed,
4) The system must have an audit system,
5) It must have elemental integrity (accurate data),
6) Access must be controlled to some degree depending on the sensitivity of the data,
7) A system must be in place to authenticate the users of the system, and
8) Sensitive data must be protected from inference

The following discussion focuses on requirements 5-8 above, since these security areas are directly affected by the choice of DBMS model. The key goal of these requirements is to ensure that data stored in the DBMS is protected from unauthorized observation or inference, unauthorized modification, and from inaccurate updates.

This can be accomplished by using access controls, concurrency controls, and updates using the two-phase commit procedure.

The level of access restriction depends on the sensitivity of the data and the degree to which the developer adheres to the principal of least privilege (access limited to only those items required to carry out assigned tasks). Typically, a lattice is maintained in the DBMS that stores the access privileges of individual users. When a user logs on, the interface obtains the specific privileges for the user. According to Pfleeger [5], access permission may be predicated on the satisfaction of one or more of the following criteria:

1) **Availability of Data:**
   Unavailability of data is commonly caused by the locking of a particular data element by another subject, which forces the requesting subject to wait in a queue.
2) **Acceptability of Access:**
   Only authorized users may view and or modify the data. In a single level system, this is relatively easy to implement. If the user is unauthorized, the operating system does not allow system access. On a multilevel system, access control is considerably more difficult to implement, because the DBMS must enforce the discretionary access privileges of the user.
3) **Assurance of Authenticity:**
   This includes the restriction of access to normal working hours to help ensure that the registered user is genuine. It also includes a usage analysis which is used to determine if the current use is consistent with the needs of the registered user, thereby reducing the probability of a fishing expedition or an inference attack. Concurrency controls help to ensure the integrity of the data. These controls regulate the manner in which the data is used when more than one user is using the same data element. These are particularly important in the effective management of a distributed system, because, in many cases, no single DBMS controls data access. If effective concurrency controls are not integrated into the distributed system, several problems can arise. Bell and Grisoni [6] identify three possible sources of concurrency problems: (1) Lost update: A successful update was inadvertently erased by another user. (2) Unsynchronized transactions that violate integrity constraints. (3) Unrepeatable read: Data retrieved is inaccurate because it was obtained during an update. Each of these problems can be reduced or eliminated by implementing a suitable locking scheme (only one subject has access to a given entity for the duration of the lock) or a timestamp method (the subject with the earlier timestamp receives priority) [6]. Special problems exist for a DBMS that has multilevel access. In a multilevel access system, users are restricted from having complete data access. Policies restricting user access to certain data elements may result from secrecy requirements, or they may result from adherence to the principal of least privilege (a user only has access to relevant information). Access policies for multilevel systems are typically referred to as either open or closed. In an open system, all the data is considered unclassified unless access to a particular data element is expressly forbidden. A closed system is just the opposite. In this case, access to all data is prohibited unless the user has specific access privileges. Classification of data elements is not a simple task. This is due, in part, to conflicting goals. The first goal is to provide the database user with access to all non-sensitive data. The second goal is to protect sensitive data from unauthorized observation or inference. For example, the salaries for all of a given firm's employees may be considered non-sensitive as long as the employee's names are not associated with the salaries. Legitimate use can be made of this data. Summary statistics could be developed such as mean executive salary and mean salary by gender. Yet an inference could be made from this data. For example, it would be fairly easy to identify the salaries of the top executives. Another problem is data security classification. There is no clear-cut way to classify data. Millen and Lunt [7] demonstrate the complexity of the problem: They state that when classifying a data element, there are three dimensions:
- The data may be classified.
- The existence of the data may be classified.
- The reason for classifying the data may be classified [7].

The first dimension is the easiest to handle. Access to a classified data item is simply denied. The other two dimensions require more thought and more creative strategies. For example, if an unauthorized user requests a data item whose existence is classified, how does the system respond? A poorly planned response would allow the user to make inferences about the data that would potentially compromise it. Protection from inference is one of the unresolved problems in secure multilevel database design. Pfleeger [5] lists several inference protection strategies. These include data suppression, logging every move users make (in order to detect behavior that suggests an inference attack), and perturbation of data. As we will discuss later, the only practical strategy for the distributed environment that maintains data accuracy is suppression.

B. Security Problems Unique to Distributed Database Management Systems

1) Centralized or Decentralized Authorization
In developing a distributed database, one of the first questions to answer is where to grant system access. Bell and Grisom [6] outline two strategies: (1) Users are granted system access at their home site. (2) Users are granted system access at the remote site. The first case is easier to handle. It is no more difficult to implement than a centralized access strategy. Bell and Grisom point out that the success of this strategy depends on reliable communication between the different sites (the remote site must receive all of the necessary clearance information). Since many different sites can grant access, the probability of unauthorized access increases. Once one site has been compromised, the entire system is compromised. If each site maintains access control for all users, the impact of the compromise of a single site is reduced (provided that the intrusion is not the result of a stolen password). The second strategy, while perhaps more secure, has several disadvantages. Probably the most glaring is the additional processing overhead required, particularly if the given operation requires the participation of several sites. Furthermore, the maintenance of replicated clearance tables is computationally expensive and more prone to error. Finally, the replication of passwords, even though they're encrypted, increases the risk of theft. A third possibility offered by Woo and Lam [8] centralizes the granting of access privileges at nodes called policy servers. These servers are arranged in a network. When a policy server receives a request for access, all members of the network determine whether to authorize the access of the user. Woo and Lam believes that separating the approval system from the application interface reduces the probability of compromise.

2) Integrity
According to Bell and Grisom [BellGris92], preservation of integrity is much more difficult in a heterogeneous distributed database than in a homogeneous one. The degree of central control dictates the level of difficulty with integrity constraints (integrity constraints enforce the rules of the individual organization). The homogeneous distributed database has strong central control and has identical DBMS schema. If the nodes in the distributed network are heterogeneous (the DBMS schema and the associated organizations are dissimilar), several problems can arise that will threaten the integrity of the distributed data. They list three problem areas:
- Inconsistencies between local integrity constraints,
- Difficulties in specifying global integrity constraints,
- Inconsistencies between local and global constraints [6].

Bell and Grisom explain that local integrity constraints are bound to differ in a heterogeneous distributed database. The differences stem from differences in the individual organizations. These inconsistencies can cause problems, particularly with complex queries that rely on more than one database. Development of global integrity constraints can eliminate conflicts between individual databases. Yet these are not always easy to implement. Global integrity constraints on the other hand are separated from the individual organizations. It may not always be practical to change the organizational structure in order to make the distributed database consistent. Ultimately, this will lead to inconsistencies between local and global constraints. Conflict resolution depends on the level of central control. If there is strong global control, the global integrity constraints will take precedence. If central control is weak, local integrity constraints will [8].

V. CONCLUSION

This paper illustrates database security concerns in general and how the database model involves database system security in particular. The latest emergences of hybrid models that combine the features of the two models discussed raise many new security questions. Distributed database management systems are veracity. Many organizations are now deploying distributed database management systems. Accordingly we have no choice but to make sure that these structures operate in a sheltered environment. We deem that as lot of technologies emerge, the impact of secure distributed database management systems on these technologies will be significant.

REFERENCES

[2] Steven P. Coy "Security Implications of the Choice of Distributed Database Management System Model: Relational vs. Object-Oriented” University of Maryland