Fundamental Research of Distributed Database

Swati Gupta[1], Kuntal Saroha[2], Bhawna [3]

[1]**Lecturer, RIMT, Chidana**
Swati.mangla.555@gmail.com

[2]**Research Scholar , IIIT, Gwaliar,**
sarohakuntal@gmail.com

[3]**M.Tech. Scholar , PDMCE,Bahadurgarh**
bhawna.kochhar9@gmail.com

ABSTRACT

The purpose of this paper is to present an introduction to Distributed Databases which are becoming very popular now a days. Today's business environment has an increasing need for distributed database and Client/server applications as the desire for reliable, scalable and accessible information is Steadily rising. Distributed database systems provide an improvement on communication and data processing due to its data distribution throughout different network sites. Not Only is data access faster, but a single-point of failure is less likely to occur, and it provides local control of data for users.

Keywords: *Distributed databases fundamentals, current research: query optimization, distribution optimization, fragmentation optimization.*

I INTRODUCTION

In today's world of universal dependence on information systems, all sorts of people need access to companies' databases. In addition to a company's own employees, these include the company's customers, potential customers, suppliers, and vendors of all types. It is possible for a company to have all of its databases concentrated at one mainframe computer site with worldwide access to this site provided by telecommunications networks, including the Internet. Although the management of such a centralized system and its databases can be controlled in a well-contained manner and this can be advantageous, it poses some problems as well. For example, if the single site goes down, then everyone is blocked from accessing the databases until the site comes back up again. Also the communications costs from the many far PCs and terminals to the central site can be expensive. One solution to such problems, and an alternative design to the centralized database concept, is known as distributed database.

In short a *distributed database* is a collection of databases that can be stored at Different computer network sites. Each database may involve different database management systems and different architectures that distribute the execution of transactions. The objective of a distributed database management system (DDBMS) is to control the management of a distributed database (DDB) in such a way that it appears to the user as a centralized database.

II DISTRIBUTED DATABASES

A distributed database management system (DDBMS) is the software that manages the DDB, and provides an access mechanism that makes this distribution transparent to the user. Distributed database system (DDBS) is the integration of DDB and DDBMS. This integration is achieved through the merging the database and networking technologies together. Or it can be described as, a system that runs on a collection of machines that do not have shared memory, yet looks to the user like a single machine.

A distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network. A distributed database management system (distributed DBMS) is the software system that permits the management of the distributed database and makes the distribution transparent to the users [1]. The term distributed database system (DDBS) is typically used to refer to the combination of DDB and the distributed DBMS. Distributed DBMSs are similar to distributed file systems (*see* Distributed File Systems) in that both facilitate access to distributed data. However, there are important differences in structure and functionality, and these characterize a distributed database system:

1. Distributed file systems simply allow users to access files that are located on machines other than their own. These files have no explicit structure (i.e., they are flat) and the relationships among data in different files (if there are any) are not managed by the system and are the users responsibility. A DDB, on the other hand, is organized according to a *schema* that defines both the structure of the distributed data, and the relationships among the data. The schema is defined according to some data model, which is usually relational or object-oriented (*s e e* Distributed Database Schemas).

2. A distributed file system provides a simple interface to users which allows them to open, read/write (records or bytes), and close files. A distributed DBMS system has the full functionality of a DBMS. It provides high-level, declarative query capability, transaction management (both concurrency control and recovery), and integrity enforcement. In this regard, distributed DBMSs are different from transaction processing systems as well, since the latter provide only some of these functions.
3. A distributed DBMS provides *transparent* access to data, while in a distributed file system the user has to know (to some extent) the location of the data. A DDB may be partitioned (called *fragmentation*) and replicated in addition to being distributed across multiple sites. All of this is not visible to the users. In this sense, the distributed database technology extends the concept of *data independence*, which is a central notion of database management, to environments where data are distributed and replicated over a number of machines connected by a network. Thus, from a user s perspective, a DDB is logically a single database even if physically it is distributed.

III ARCHITECTURE CONCERN

A. The Hardware

Due to the extended functionality the DDBS must be capable of, the DDBS design becomes more complex and more sophisticated. At the physical level the differences between centralized and distributed systems are:

- Multiple computers called sites.
- These sites are connected via a communication network, to enable the data/query communications. Figure 1. illustrates this architecture

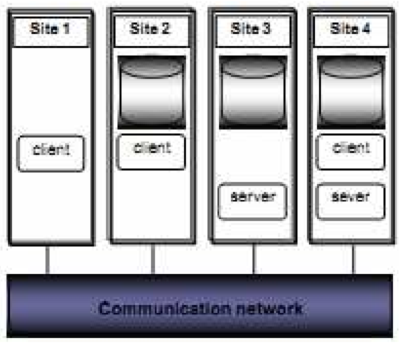

Figure 1 Client Server Architecture

Networks can have several types of topologies that define how nodes are physically and logically connected. One of the popular topologies used in DDBS, the client-server architecture is described as follows: the principle idea of this architecture is to define specialized servers with specific functionalities such as: printer server, mail server, file server, etc. these serves then are connected to a network of clients that can access the services of these servers. Stations (servers or clients) can have different design complexities starting from Diskless client to combined server-client machine. This is illustrated in Figure 1.

The server-client architecture requires some kind of function definition for servers and clients. The DBMS functions are divided between servers and clients using different approaches. We present a common approach that is used with relational DDBS, called centralized DMBS at the server level.
The client refers to a data distribution dictionary to know how to decompose the global query in to multiple local queries. The interaction is done as follows:

1. Client parses the user's query and decomposes it into independent site queries.

2. Client forwards each independent query to the corresponding server by consulting with the data distribution dictionary.

3. Each server process the local query, and sends back the resulting relation to the client.

4. Client combines (manually by the user, or automatically by client abstract) the received sub queries, and do more processing if needed to get to the final target result.

We would like to discuss the different architectures of DDBS for the two main types, the client/server, and the distributed databases:

The client/server:
the simplest tactic is known as the file server approach. When a client computer on the LAN needs to query, update, or otherwise use a file on the server, the entire file must be sent from the server to that client. All of the querying, updating, or other processing is then performed in the client computer. If changes were made to the file, the entire file is then shipped back to the server. Clearly, for files of even moderate size, shipping entire files back and forth across the LAN with any frequency will be very costly. In terms of concurrency control, obviously the entire file must be locked while one of the clients is updating even one record in it. Other than providing a basic file-sharing capability, this arrangement's drawbacks render it not very practical or useful.

DBMS server approach:
A much better arrangement is variously known as the database server or DBMS server approach. Again, the database is located at the server, but this time, the processing is split between the client and the server, and there is much less data traffic on the network. Say that someone at a client computer wants to query the database at the server. The query is entered at the client, and the client computer performs the initial keyboard and screen interaction processing, as well as initial syntax checking of the query. The system then ships the query over the LAN to the server where the query is actually run against the database. Only the results are shipped back to the client. Certainly, this is a much better arrangement than the file server approach! The network data traffic is reduced to a tolerable level, even for frequently queried databases. Also, security and concurrency control can be handled at the server in a much more contained way. The only real drawback to this approach is that the company must invest in a sufficiently powerful server to keep up with all of the activity concentrated there.

Two-tier client/server:
Another issue involving the data on a LAN is the fact that some databases can be stored on a client PC's own hard drive while other databases that the client might access are stored on the LAN's server. This is also known as a two-tier approach, (Figure 2). Software has been developed that makes the location of the data transparent to the user at the client. In this mode of operation, the user issues a query at the client, and the software first checks to see if the required data is on the PC's own hard drive. If it is, the data is retrieved from it, and that is the end of the story. If it is not there, then the software automatically looks for it on the server.

In an even more sophisticated three-tier approach (Figure 3), if the software doesn't find the data on the client PC's hard drive or on the LAN server, it can leave the LAN through a gateway computer and look for the data on, for example, a large, mainframe computer that may be reachable from many LANs.

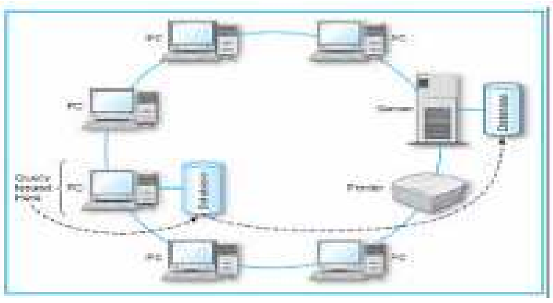

Figure 2: Two-tier Client/Server

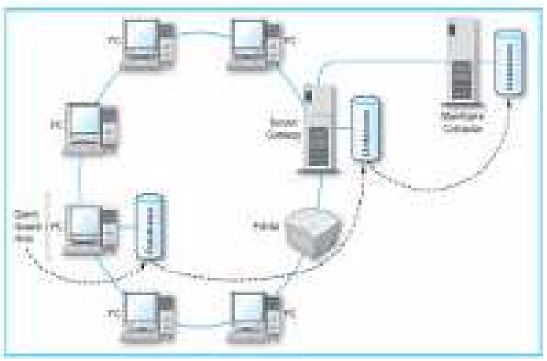

Figure 3: Three Tier Client/Server

Three-tier approach: In another use of the term three-tier approach, the three tiers are the client PCs, servers known as application servers, and other servers known as database servers, (Figure 4). In this arrangement, local screen and keyboard interaction is still handled by the clients, but they can now request a variety of applications to be performed at and by the application servers. The application servers, in turn, rely on the database servers and their databases to supply the data needed by the applications. Though certainly well beyond the scope of LANs, an example of this kind of arrangement is the World Wide Web on the Internet. The local processing on the clients is limited to the data input and data display capabilities of browsers such as Netscape's Communicator and Microsoft's Internet Explorer. The application servers are the computers at company Web sites that conduct the companies' business with the "visitors" working through their browsers. The company application servers in turn rely on the companies' database servers to provide the necessary data to complete the transactions. For example, when a bank's customer visits his bank's Web site, he can initiate lots of different transactions, ranging from checking his account balances to transferring money between accounts to paying his credit card bills. The bank's Web application server handles all of these transactions. It, in turn, sends requests to the bank's database server and databases to retrieve the current account balances, add money to one account while deducting money from another in a funds transfer, and so forth.

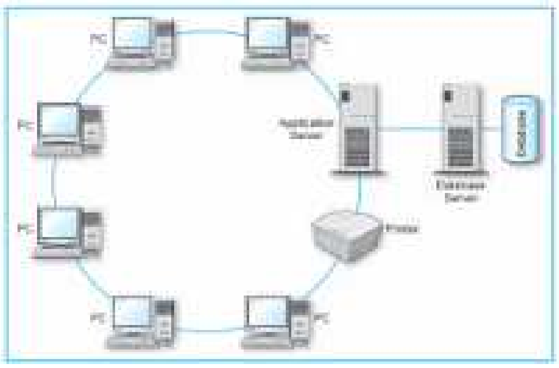

Figure 4 : Another version of Three Tier

Distributed Database

1. *No replication*:
The first and simplest idea in distributing the data would be to disperse the six tables among the five sites. If particular tables are used at some sites more frequently than at other sites, it would make sense to locate the tables at the sites at which they are most frequently used. Benefits include: local autonomy (security, concurrency, backup, recovery), efficient local transaction. Problems include: if one site goes down, then it is not accessible by the rest of the system. Expensive joins. The security can be argued, one single place, one database is more secure than DDBS

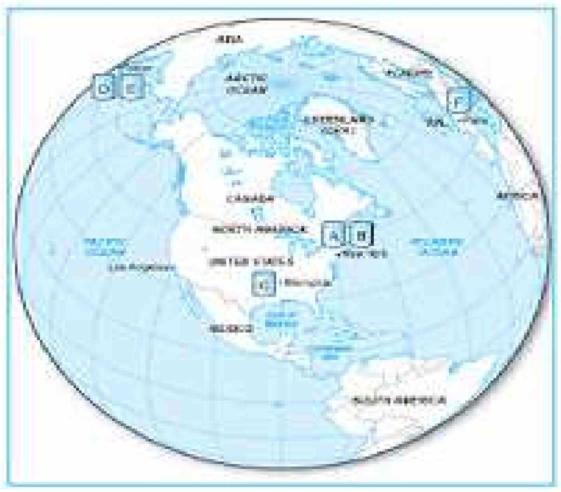
Figure 5 : No replication Approach

2 Replication the entire DB at each site: Benefits include, better availability. If more than one site requires frequent access to a particular table, the table can be replicated at each of those sites, again minimizing telecommunications. And copies of a table can be located at sites that have tables with which it may have to be joined. Problems include, less security, concurrency and consistency. At the extreme: all tables are replicated, very efficient for availability and join, whereas it is the worst alternative for concurrency, consistency, and disk space Figure 6.

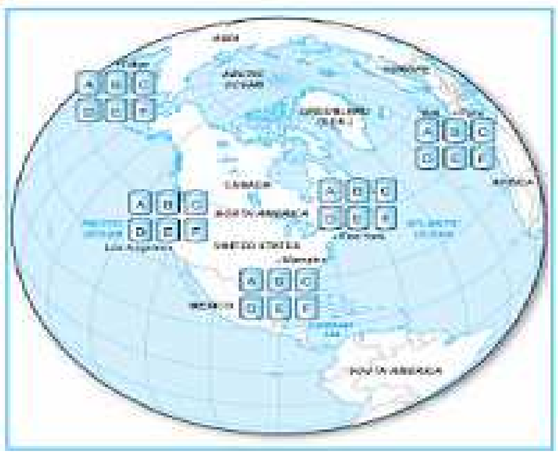
Figure 6 : Replication of all Tables

Selective replication: replicate all at the headquarters (improves join, all joins at the headquarters, and replicate each table only once in the network, so you have 2 copies of each on the entire network. Figure 7.

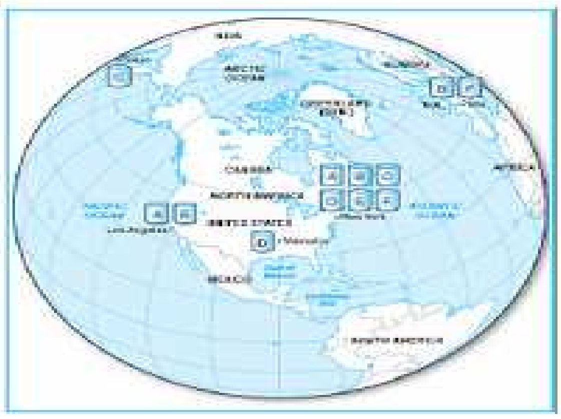
Figure 7 : Selective Replication

This last approach has some down sides, more than two sites could use a table frequently (need more replicas), bottleneck at the headquarter for the join operations. To avoid these, we use the heuristics:

- Place copies of tables at the sites that use them most heavily in order to minimize telecommunications costs.
- Ensure that there are at least two copies of important or frequently used tables to realize the gains in availability.
- Limit the number of copies of any one table to control the security and concurrency issues.
- Avoid any one site becoming a bottleneck.

Figure 8. illustrates a DDBS using these Heuristics

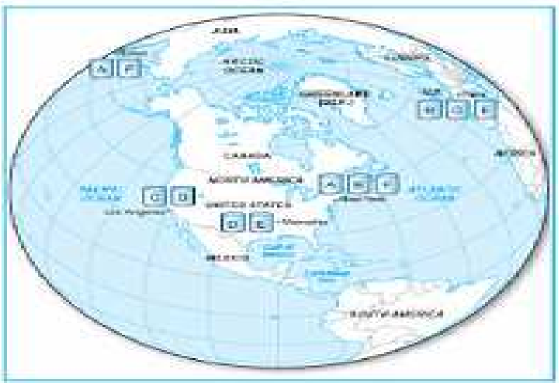

Figure 8: Replication by heuristics

IV SOFTWARE ASPECT

In a typical DDBS, three levels of software modules are defined:

- ➢ The server software: responsible for b local data management at site.
- ➢ The client software: responsible for most of the distribution functions; DDBMS catalog, processes all requests that require more than one site. Other functions for the client include: consistency of replicated data, atomicity of global transactions.
- ➢ The communications software: provides the communication primitives, used by the client/server to exchange data and commands Figure 9.

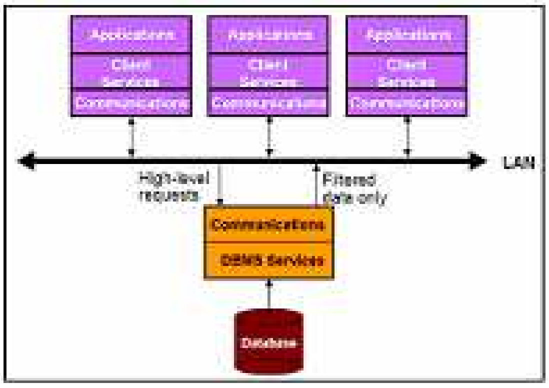

Figure 9 : Client/ Server Software

Advantages of Client/Server architecture include: More efficient division of labor, horizontal and vertical scaling of resources, better price/performance on client machines, ability to use familiar tools on client machines, client access to remote data (via standards), full DBMS functionality provided to client workstations, and overall better system price/performance

Disadvantages of Client/Server architecture include: server forms bottleneck, server forms single point of failure, and database scaling is difficult .

It is preferable for a DDMBS to have the property of distribution transparency (Figure 10), where the user's can issue a global queries without knowing or worrying about the global distribution in the DDBS.

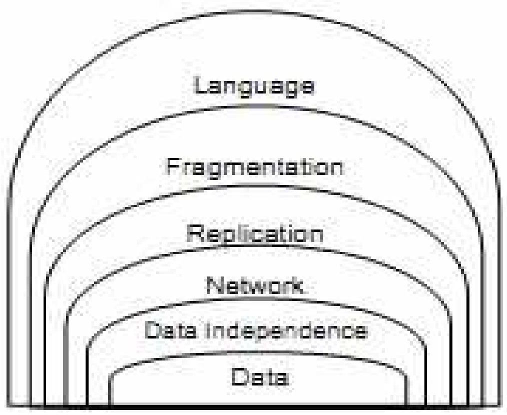

Figure 10 : Layers of Transparency

V FRAGMENTATION & REPLICATION

In distributing and allocating the database in the previous section, we assumed that the entire relations are kept intact. However, in DDBS we need to define the logical unit of DB distribution and allocation. In some cases it might be more efficient to split the tables into smaller units (fragments) and allocate them in different sites.

Fragmentation has three different types:

A. Horizontal Fragmentation

As appears in Figure 11. the table G has been added to demonstrate the fragmentation operation. An example on horizontal fragmentation is the employee's table (G). It makes since for the company to split G into different partitions based on the employees who work on that site. This makes the management, queries, and transactions convenient and efficient. The Down side of this choice is that, whenever a query involving all G records, it has to request all partitions from all sites and do a union on them. .

Figure 11 : Fragmentation among Tables

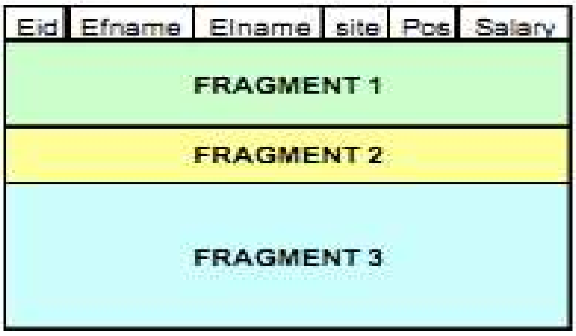

Figure 12 : Horizontal Fragmentation

B. Vertical Fragmentation

In vertical partitioning, the columns of a table are divided up among several cities on the network. Each such partition must include the primary key attribute(s) of the table. This arrangement can make sense when different sites are responsible for processing different functions involving an entity. For example, the salary attributes of a personnel table might be stored in one city while the skills attributes of the table might be stored in another city. Both partitions would include the employee number, the primary key of the full table. A down side of this option is that, a query involving the entire table G (Figure 13) would have to request all portions from all sites and do a join on them.

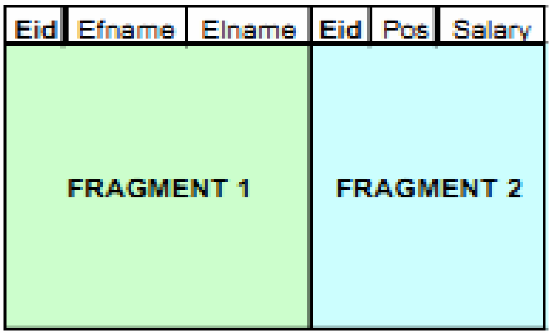

Figure 13 : Vertical Fragmentation

C. Hybrid Fragmentation

In this type of fragmentation scheme, the table is divided into arbitrary blocks, based on the needed requirements. Each fragment hen can be allocated on to a specific site. This type of fragmentation is the most complex one, which needs more management. This is illustrated in Figure 14

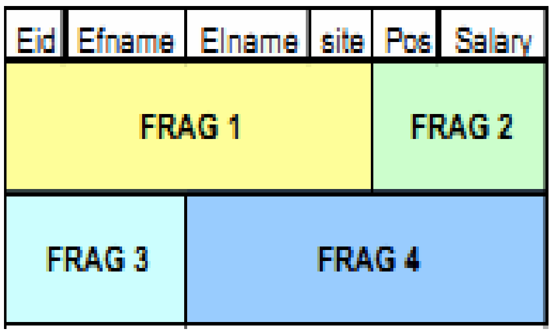

Figure 14 : Hybrid Fragmentation

.

VI QUERY PROCESSING

DDBS adds to the conventional centralized DDBS some other types of processing expenses, because of the additional design (hardware & software) to handle the distribution. These expenses present as the cost of data transfer over the network. Data transferred could be, intermediate files resulting from local sites, or final results need to be sent back to the original site that issued the query. Therefore, database designers are concerned about query optimization, which target minimizing the cost of transferring data across the network.

One method to optimize query on DDBS is, the simijoin, where a relation R1 can send the entire join-column CR1 to the target relation R2, then the site containing R2 would perform the join on CR1, and project on the passed attributes. The resulting tuples are then shipped back to R! for further processing. This can significantly enhance the query efficiency, since the data transferred on the network is minimized.

VII CONCURRENCY & RECOVERY

DDBS design of concurrency and recovery, has to consider different aspects other than of those of centralized DBS. These aspects include:

- Multiple copies of data: concurrency has to maintain the data copies consistent. Recovery on the other hand has to make a copy consistent with others whenever a site recovers from a failure.
- Failure of communication links
- Failure of individual sites
- Distributed commit: during transaction commit some sites may fail, so the two-phase commit is used to solve this problem.
- Deadlocks on multiple sites.

The following two sections describe two suggestions to manage concurrency control .

A. Distinguished Copy of a Data Item

There are three variations to this method: primary site technique, primary site with backup site, and primary copy technique. These techniques are described as follows:

a) Primary site
In this method, a single site is designated as the coordinator site. All locks and unlocks for all data units are controlled by this site. One advantage is, easy to implement. However two downsides of this method are: overloading of the coordinator site, and this site forms a single point failure for the entire DDBS.

b) Primary site with backup site
This technique addresses the second disadvantage in the 1st technique (primary site) by designating a backup site, that can take over as the new coordinator in case of failure, in which case, another backup site has to be selected.

c) Primary copy technique
This method distribute the load to the sites that have a designated primary copy of a data unit as opposed to centralizing the entire data units in one coordinator site. This way if a site goes down, only transactions involving the primary copies residing on that site will be effected.

B. Voting

This method does not designate any distinguished copy or site to be the coordinator as suggested in the 1st two methods described above. When a site attempts to lock a data unit, requests to all sites having the desired copy, must be sent asking to lock this copy. If the requesting transaction did was not granted the lock by the majority voting from the sites, then the transaction fails and sends cancellation to all. Otherwise it keeps the lock and informs all sites that it has been granted the lock.

C. Recovery

The first step of dealing with the recovery problem is to identify that there was a failure, what type was it, and at which site did that happen. Dealing with distributed recovery requires aspects include: database logs, and update protocols, transaction failure recovery protocol, etc .

VIII CONCLUSION

Through this paper, we want to attract readers towards the advantageous side of distributed databases. We also mentioned the software architecture being used for the distributed database .We also described Fragmentation, replication and recovery aspect also in order to make readers completely aware about the topic being described here. Besides having a fruitful side of DDBs ,It also attracts researchers for finding the new scope in it.

IX REFERENCES

[1] Patrick O'Neil, and Goetz Graefe. 1995. Multi-Table Joins Through Bitmapped Join Indices. *SIGMOD Record, Vol. 24, No. 3, September 1995*

[2] Ambrose Goicoechea. 2000. Requirements Blueprint and Multiple Criteria For Distributed Database Design. *International Council on Systems Engineering (INCOSE) 2000.*

[3] Yin-Fu Huang, and Jyh-Her Chen. 2001. Fragment Allocation in Distributed Database Design. *Journal of Information Science and Engineering 17, 491-506 (2001).*

[4] Cyrus Shahabi, Latifur Khan, and Dennis McLeod. 2000. A Probe-Based Technique to Optimize Join Queries in Distributed Internet Databases. *Knowledge and Information Systems (2000) 2: 373-385*

[5] Charles P. Pfleeger and Shari Lawrence Pfleeger, Security in Computing, Prentice Hall Professional Technical Reference, Upper Saddle River, New Jersey, 2003.

[6] James F. Kurose and Keith W. Ross, Computer Networking: A Top-Down Approach Featuring the Internet, Pearson Education, Inc, New York, 2003.