Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 7.056

IJCSMC, Vol. 10, Issue. 1, January 2021, pg.23 – 48

DISTRIBUTED DATABASE MANAGEMENT SYSTEM (DBMS) ARCHITECTURES AND DISTRIBUTED DATA INDEPENDENCE

[1]Orlunwo Placida O.; [2]Prince Oghenekaro ASAGBA
[1]Computer Science Department, Ignatius Ajuru University of Education
[2]Computer Science Department, University of Port Harcourt
[1] orlunwoplacida@gmail.com; [2] asagba.prince@uniport.edu.ng
DOI: 10.47760/ijcsmc.2021.v10i01.004

Abstract: ***Distributed database (DDB) is one of the emerging fields of technology and market research. This article addresses different architectures of distributed databases, distributed database management systems (DDBMS), data dependency techniques, the importance and drawbacks of the DDB. The problem areas mentioned in the paper are extremely useful when implementing distributed databases to ensure easy management of competence, impasse, protection and privacy. In this paper we will also research distributed database design for integrating the business environment in a distributed database.***
Keywords: Distributed database architecture, Concurrency control, Distributed database management system, Deadlock, Query optimization, Query processing, Transaction, Transparency

1. INTRODUCTION

A database administration system enables a person to organize, store and get the data from a computer. In the very early years of computers, punch cards were used for input, output and data storage, but this is a means of communicating the stored memory with a computer. Punch cards offered a fast way to enter data, and to retrieve it. Herman Hollerith is given credit for adapting the punch cards used for weaving looms to act as the memory for a mechanical tabulating machine, in 1890. And later, there were databases. In the recent development of computers, databases (or DBs) played a very significant role. In the early 1950s the first computer programs were designed to concentrate almost exclusively on coding languages and algorithms. Computers

were essentially large computers and data (names, telephone numbers) were considered to be the legacy of information processing. Computer had just begun to become commercially available, which immediately became relevant as people from business began to use for real-life purposes. By the mid-1960's several types of general-use database systems were provided as computers gained speed and versatility and began to become popular. In turn, customers needed the creation of a specification that would lead to the formation of the Bachman Database Task Group. This community took care of the creation and standardization of a popular business-oriented language (COBOL). This standard was proposed in 1971 by the Database Task Group, also known as the 'CODASYL method' Thomas (2016). Before the advent of database, everything had to be registered on paper until there were databases. It included lists, papers, books and endless libraries containing hundreds of millions of documents in cabinets. It was a slow and painstaking task to locate and retrieve the record when it was required to access one of those documents. There have also been issues from incorrect documents to incidents that have wiped out whole libraries and ruined the history of societies. Security issues often existed, since physical access was always easy. Database has been developed to overcome these limitations of conventional paper storage. In databases, files are called documents and fields are referred to as the individual record data elements (e.g. name, telephone number, date of birth).

Since the beginning of databases, the way the elements are stored has changed. Design and implementation of a database is in fact a major challenge and requires consideration of the needs of an organization and careful design and execution. Different systems are likely to be better suited to your needs depending on the type, structure, data model, data store and planned use case of your data (Codd, 1993). Your decision may also be affected by your scheme or query method, accuracy or latency specifications, and even transaction speed (also in real-time). As such this has led to the emergence of various types of databases (Figure 1) used for storing different varieties of data to suit organization demand.

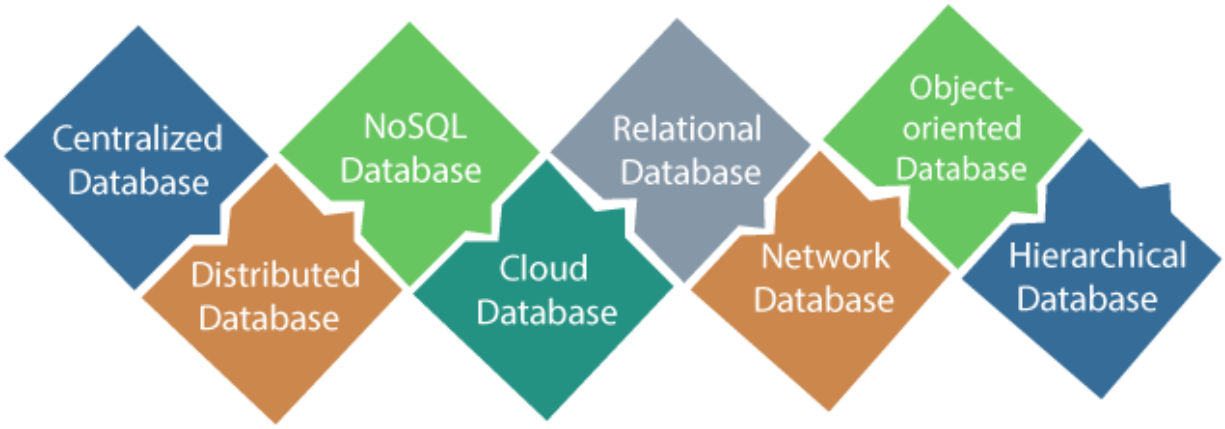

Figure 1: Types of Databases

Now if you ask, "What's a database? "Usually you'll be thinking of a set of linked tables. If an individual in the 1990s were asked the same question, he may have just thought of a single wide table from which they can get all information. But we have actually progressed from that simple concept of databases. So, where are you going to pick a database? Various database systems would probably be more suitable for your needs depending on the form, structure, data model, data storage and planned use case of your data. Your decisions can also be affected by your scheme or querying process, your accuracy or latency specifications, or even transaction speed (including real time). The effectiveness of an organization today, more than ever, depends on an organization's capable acquirement of reliable and timely operational data, efficient management and use for analysis and guidance on the operations of the organization.

Database Management System (DBMS) is the name of the software which provides the data for the other applications that enable us to interact today with all digital information systems (Patrick, 2001). A DBMS is also called a database. Data are exchanged with a range of standards, but they all serve the same purpose, namely to provide data to applications. The applications then process the data and make it usable to the users: information.

The main aim of this article is to describe and explain database in an understandable way. The thought of a single-size database is unlikely and this article shows that for various types of technology ventures, there are many types of databases which includes distributed database. This

article addresses the evolution of databases, explores the Distributed Database Management System (DDBMS) architecture and Distributed Data Independence and the new forms of data problems we have to face in business intelligence.

2. Distributed Database System

Distributed databases are classified as databases located at different Same machines or locations that appear as one centralized end user database (Joseph, 2009). Thus, the whole load is shared by a set of machines/computers, rather than providing a centralized database. In reality, it is a number of server machines working in sync to meet multiple users' requirements. These machines are linked to each other in a distributed system via wireless connection or via various communication media which transmit high-speed data.

Contrary to centralized database, all the data is stored at a single location and is believed to be handled sequentially in a single transaction. Databases distributed are used to process knowledge requests by client/server architectures. This distributed database system is commonly categorized as Homogenous and Heterogeneous distributed database system (Singole, 2016).

Homogenous distributed database system

in a homogenous distributed database system, all the physical locations have the same underlying hardware and run the same operating systems and database applications. i.e on all computers Oracle is used as DBMS system. Homogenous distributed database systems appear to the user as a single system, and they can be much easier to design and manage. For a distributed database system to be homogenous, the data structures at each location must be either identical or compatible. The database application used at each location must also be either identical or compatible.

There are two types of homogeneous distributed database:

Autonomous: Each database is independent that is it functions on its own. They are integrated by a controlling application and use message passing to share data updates.

Non-autonomous: Data is distributed across the homogeneous nodes and a central or master DBMS co-ordinates data updates across the sites.

Heterogeneous Distributed Database System

In a heterogeneous distributed database, the hardware, operating systems or database applications may be different at each location. Different sites may use different schemas and software, although a difference in schema can make query and transaction processing difficult. Different nodes may have different hardware, software and data structure, or they may be in locations that are not compatible. Users at one location may be able to read data at another location but not upload or alter it. Heterogeneous distributed databases are often difficult to use, making them economically infeasible for many businesses. It is quite often that distributed database is misunderstood to be alike replicated database. Up to some extent it may seems to be same but the distinction is based on its purity. In a pure (that is, not replicated) distributed database, the system manages a single copy of all data and supporting database objects.

Federated: The heterogeneous database systems are independent in nature and integrated together so that they function as a single database system.

Un-federated: The database systems employ a central coordinating module through which the databases are accessed.

2.1 Motivations behind Use of Distributed DBS

The increasing trends in developing and using distributed database are due to following motivational factors.

i. **Performance**: as multiple resources can be used performance can be increased.
ii. **Increased availability:** if one site is not in order then other site can be available.
iii. **Distributed Access to Data:** data can be accessed from remote site if the required data is not available on local site.
iv. **Analysis of Distributed Data:** for organization it is possible to access and analyse the data from multiple sites.

3. Distributed Database Management System Architecture

Distributed DBMS architectures are generally developed depending on three parameters

Distribution: It states the physical distribution of data across the different sites.

Autonomy: It indicates the distribution of control of the database system and the degree to which each constituent DBMS can operate independently.

Heterogeneity: It refers to the uniformity or dissimilarity of the data models, system components and databases.

The architecture of distributed DBMS involves;

i. Client-Server
ii. Collaborating Server (peer-to-peer)
iii. Middleware.

I. Client - Server Architecture for DDBMS

This is a two-level architecture where the functionality is divided into servers and clients. The server functions primarily encompass data management, query processing, optimization and transaction management. Client functions include mainly user interface. However, they have some functions like consistency checking and transaction management.

The main three reasons behind using this system are:

i. Easy separation of client and server.
ii. Neither the expensive server or client is underutilized
iii. Users can run GUI to which they are familiar

The two different clients - server architecture are:

i. Single Server Multiple Client
ii. Multiple Server Multiple Client

II. Collaborative (Peer-to-Peer) Architecture for DDBMS

In these systems, each peer acts both as a client and a server for imparting database services (Pudn, 2007). The peers share their resource with other peers and co-ordinate their activities.

This architecture generally has four levels of schemas:

i. Global Conceptual Schema: Depicts the global logical view of data.
ii. Local Conceptual Schema: Depicts logical data organization at each site.
iii. Local Internal Schema: Depicts physical data organization at each site.
iv. External Schema: Depicts user view of data.

III. Multi - DBMS Architectures

This is an integrated database system formed by a collection of two or more autonomous database systems.

There are two design alternatives for multi-DBMS:

i. Model with multi-database conceptual level.
ii. Model without multi-database conceptual level.

Multi-DBMS can be expressed through six levels of schemas:

i. **Multi-database View Level**: Depicts multiple user views comprising of subsets of the integrated distributed database.
ii. **Multi-database Conceptual Level**: Depicts integrated multi-database that comprises of global logical multi-database structure definitions.
iii. **Multi-database Internal Level**: Depicts the data distribution across different sites and multi-database to local data mapping.
iv. **Local database View Level**: Depicts public view of local data.
v. **Local database Conceptual Level**: Depicts local data organization at each site.
vi. **Local database Internal Level:** Depicts physical data organization at each site.

4. Distributed Data Dependence Strategy

DBMS technology has matured and has been significantly evolving in the computer network and distributed computing technologies. The end result is the formation of distributed systems. These systems have become the primary data management techniques for applications that are highly data intensive. Many DBMS vendors have included a certain amount of distribution in their products to enable for efficient data management, this includes design strategy, query.

4.1 Technology Design Strategy

A distributed database is a database managed by a DBMS system in which not all storage devices are associated with a common processing unit (CPU). It can be stored on many computers in the same physical location or distributed through an interconnected network of computers. Data sets may be spread across many physical locations (e.g. in a database). The most common distribution design strategy a are as follows:

i. Non-Replicated and Non-Fragmented
ii. Fully Replicated
iii. Partially Replicated

iv. Fragmentation

v. Mixed

i. Non-Replicated and Non-Fragmented

In this design alternative, different tables are placed at different sites. Data is placed so that it is at a close proximity to the site where it is used most. It is most suitable for database systems where the percentage of queries needed to join information in tables placed at different sites is low. If an appropriate distribution strategy is adopted, then this design alternative helps to reduce the communication cost during data processing.

ii. Fully Replicated

In this design alternative, at each site, one copy of all the database tables is stored. Since, each site has its own copy of the entire database; queries are very fast requiring negligible communication cost. On the contrary, the massive redundancy in data requires huge cost during update operations. Hence, this is suitable for systems where a large number of queries are required to be handled whereas the number of database updates is low.

iii. Partially Replicated

Copies of tables or portions of tables are stored at different sites. The distribution of the tables is done in accordance to the frequency of access. This takes into consideration the fact that the frequency of accessing the tables vary considerably from site to site. The number of copies of the tables (or portions) depends on how frequently the access queries execute and the site which generate the access queries.

Advantages of Replication

a. **Reliability**: In case of failure of any site, the database system continues to work since a copy is available at another site(s).

b. **Reduction in Network Load**: Since local copies of data are available, query processing can be done with reduced network usage, particularly during prime hours. Data updating can be done at non-prime hours.

c. **Quicker Response**: Availability of local copies of data ensures quick query processing and consequently quick response time.
d. **Simpler Transactions**: Transactions require a smaller number of joins of tables located at different sites and minimal coordination across the network. Thus, they become simpler in nature.

Disadvantages of Replication

a. **Increased Storage Requirements**: Maintaining multiple copies of data is associated with increased storage costs. The storage space required is in multiples of the storage required for a centralized system.
b. **Increased Cost and Complexity of Data Updating**: Each time a data item is updated, the update needs to be reflected in all the copies of the data at the different sites. This requires complex synchronization techniques and protocols.
c. **Undesirable Application Database coupling**: If complex update mechanisms are not used, removing data inconsistency requires complex co-ordination at application level. This results in undesirable application database coupling.

iv. Fragmentation

In this design, a table is divided into two or more pieces referred to as fragments or partitions, and each fragment can be stored at different sites. This considers the fact that it seldom happens that all data stored in a table is required at a given site.

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called "constructiveness."

Moreover, fragmentation increases parallelism and provides better disaster recovery. Here, there is only one copy of each fragment in the system, i.e. no redundant data.

The three fragmentation techniques are:

a. Vertical fragmentation
b. Horizontal fragmentation

c. Hybrid fragmentation

a. Vertical Fragmentation

In vertical fragmentation, the fields or columns of a table are grouped into fragments. In order to maintain re-constructiveness, each fragment should contain the primary key field(s) of the table. Vertical fragmentation can be used to enforce privacy of data.

For example, let us consider that a company (Figure 2) "XYZ" database keeps records of all registered projects in a Projects table having the following schema.

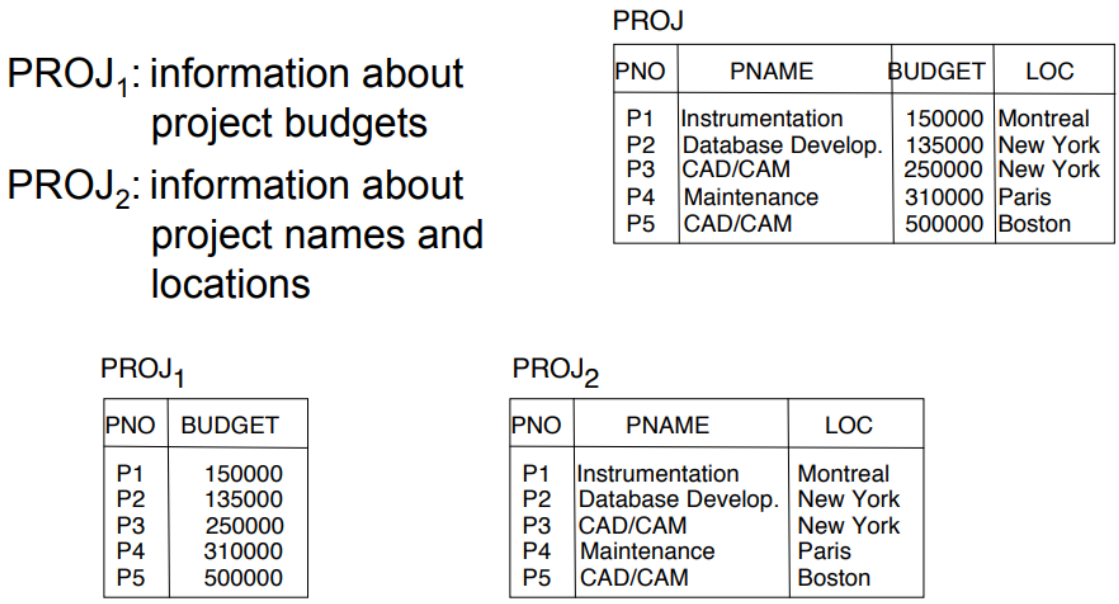

Figure 2: Vertical Fragmentation of Company "XYZ"

b. Horizontal Fragmentation

Horizontal fragmentation groups the tuples of a table in accordance to values of one or more fields. Horizontal fragmentation should also confirm to the rule of re-constructiveness. Each horizontal fragment must have all columns of the original base table.

Figure 3 project schema, if the details of all projects of project "CAD/CAM needs to be maintained, then the designer will horizontally fragment the database as follows:

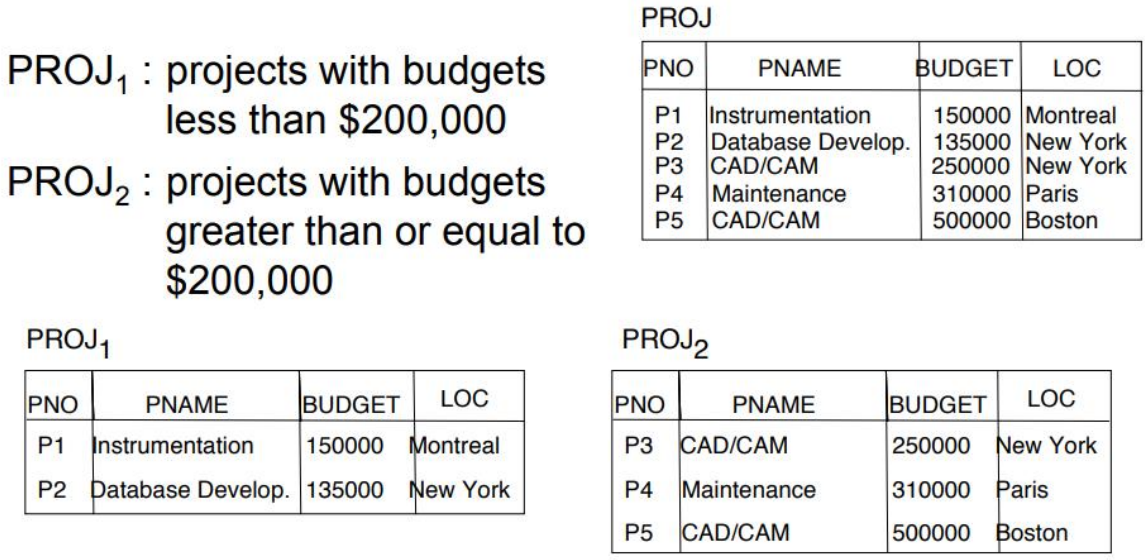

Figure 3: Horizontal Fragmentation

c. Hybrid Fragmentation

In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques is used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task.

Hybrid fragmentation can be done in two alternative ways

a. At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.
b. At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.

Advantages of Fragmentation

a. Since data is stored close to the site of usage, efficiency of the database system is increased.

b. Local query optimization techniques are sufficient for most queries since data is locally available.
c. Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages of Fragmentation

a. When data from different fragments are required, the access speeds may be very high.
b. In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
c. Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

v. Mixed Distribution

This is a combination of fragmentation and partial replications. Here, the tables are initially fragmented in any form (horizontal or vertical), and then these fragments are partially replicated across the different sites according to the frequency of accessing the fragments.

4.2 Query Processing Strategy

In a distributed database system (Figure 4), processing a query comprises of optimization at both the global and the local level. The query enters the database system at the client or controlling site.

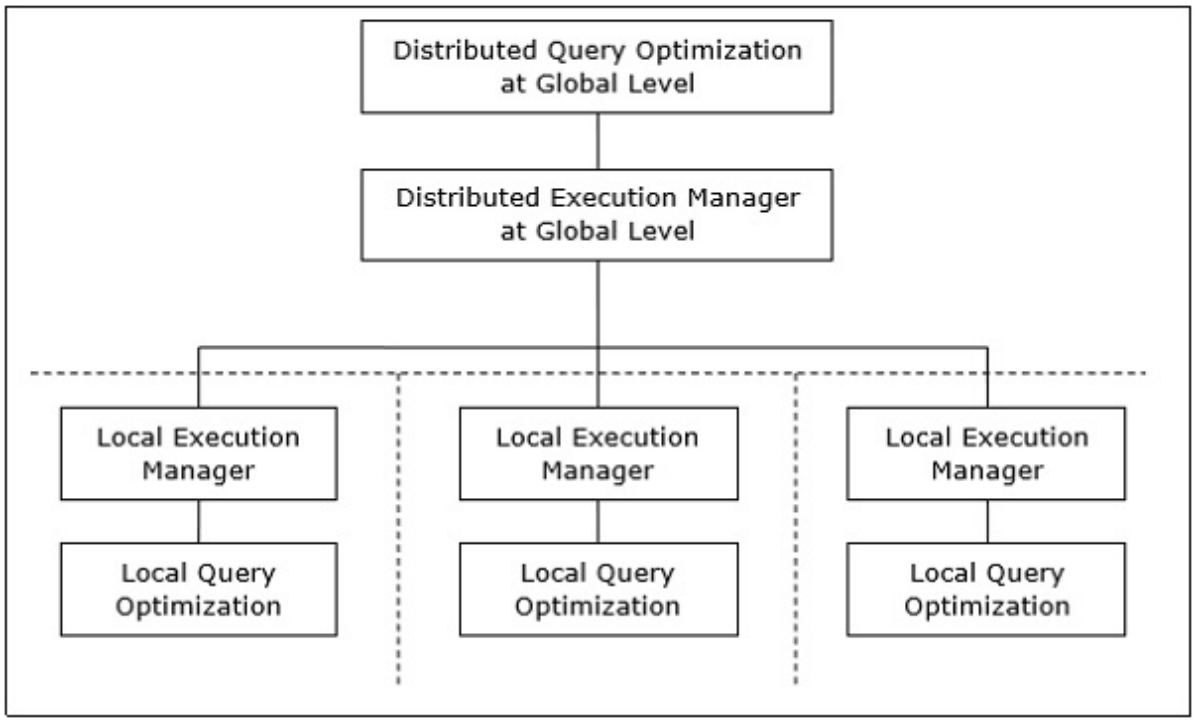

Figure 4: Distributed Query Processing

Mapping Global Queries into Local Queries

The process of mapping global queries to local ones can be realized as follows:

i. The tables required in a global query have fragments distributed across multiple sites. The local databases have information only about local data. The controlling site uses the global data dictionary to gather information about the distribution and reconstructs the global view from the fragments.

ii. If there is no replication, the global optimizer runs local queries at the sites where the fragments are stored. If there is replication, the global optimizer selects the site based upon communication cost, workload, and server speed.

iii. The global optimizer generates a distributed execution plan so that least amount of data transfer occurs across the sites. The plan states the location of the fragments, order in which query steps needs to be executed and the processes involved in transferring intermediate results.

iv. The local queries are optimized by the local database servers. Finally, the local query results are merged together through union operation in case of horizontal fragments and join operation for vertical fragments.

Figure 5, Query processing involves the conversation and translation of high-level SQL query into some low-level instructions that database engine can read and execute the query (Zloof, 2017).

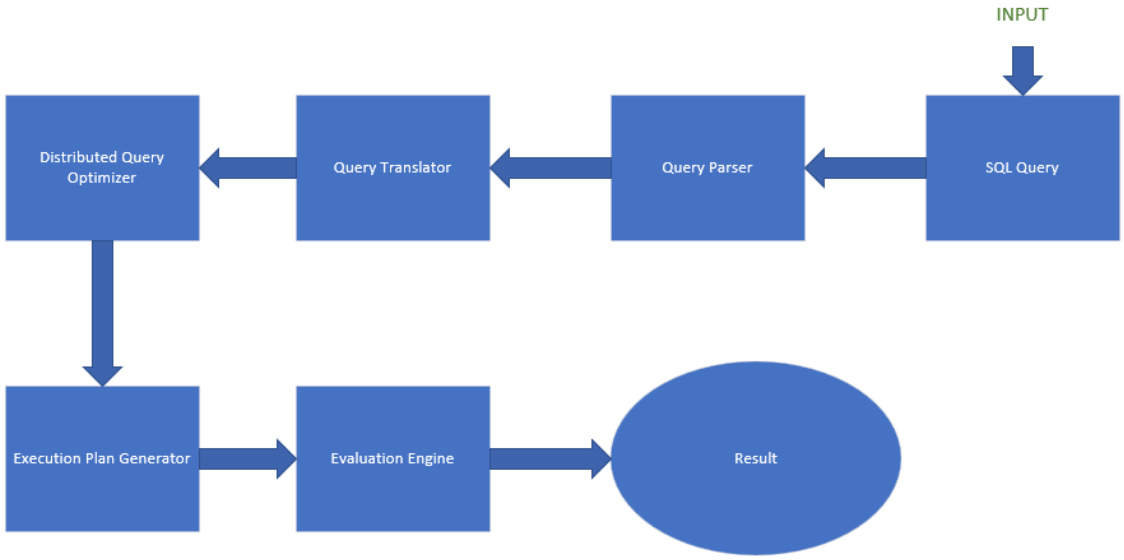

Figure 5: Distributed Query Processing

As shown in Figure 5, the inputted SQL query is first parsed by Query Parser and then translated by Query Translator. It is optimized then and move towards Execution Engine where required result is achieved. In distributed database system environment, the query is broken down in different steps and then forwarded towards the fragments for execution.

2.1 Components of query optimizer

Query Optimizer obtains query from query translator after parsing from query parser and works on three components:

i. Search Space
ii. Search Strategy
iii. Cost Model

4.3 Query Optimization Strategy

In the centralized database system, there are many ways for executing the queries. Its expected cost is mainly the CPU cost and I/O price. It aims at making queries cost lowest. In the distributed database system, the query optimization includes two parts:

i. Query strategy optimization
ii. Local processing optimization.

And the query strategy optimization is more important between them. There will be several strategies in the same query due to that the data are stored in different sites. The system resource and response time while each strategy costs are also different. So, the expected cost should include corresponding communication cost. Abhijeet R., et al (2013)

Distributed query optimization requires evaluation of a large number of query trees each of which produce the required results of a query. This is primarily due to the presence of large amount of replicated and fragmented data. Hence, the target is to find an optimal solution instead of the best solution.

The main issues for distributed query optimization are:

i. Optimal utilization of resources in the distributed system.
ii. Query trading.
iii. Reduction of solution space of the query.

i. Optimal Utilization of Resources in the Distributed System

A distributed system (Figure 6) has a number of database servers in the various sites to perform the operations pertaining to a query. Following are the approaches for optimal resource utilization.

a. Operation Shipping

In operation shipping, the operation is run at the site where the data is stored and not at the client site. The results are then transferred to the client site. This is appropriate for operations where the operands are available at the same site.

b. Data Shipping

In data shipping, the data fragments are transferred to the database server, where the operations are executed. This is used in operations where the operands are distributed at different sites. This is also appropriate in systems where the communication costs are low, and local processors are much slower than the client server.

c. Hybrid Shipping

This is a combination of data and operation shipping. Here, data fragments are transferred to the high-speed processors, where the operation runs. The results are then sent to the client site.

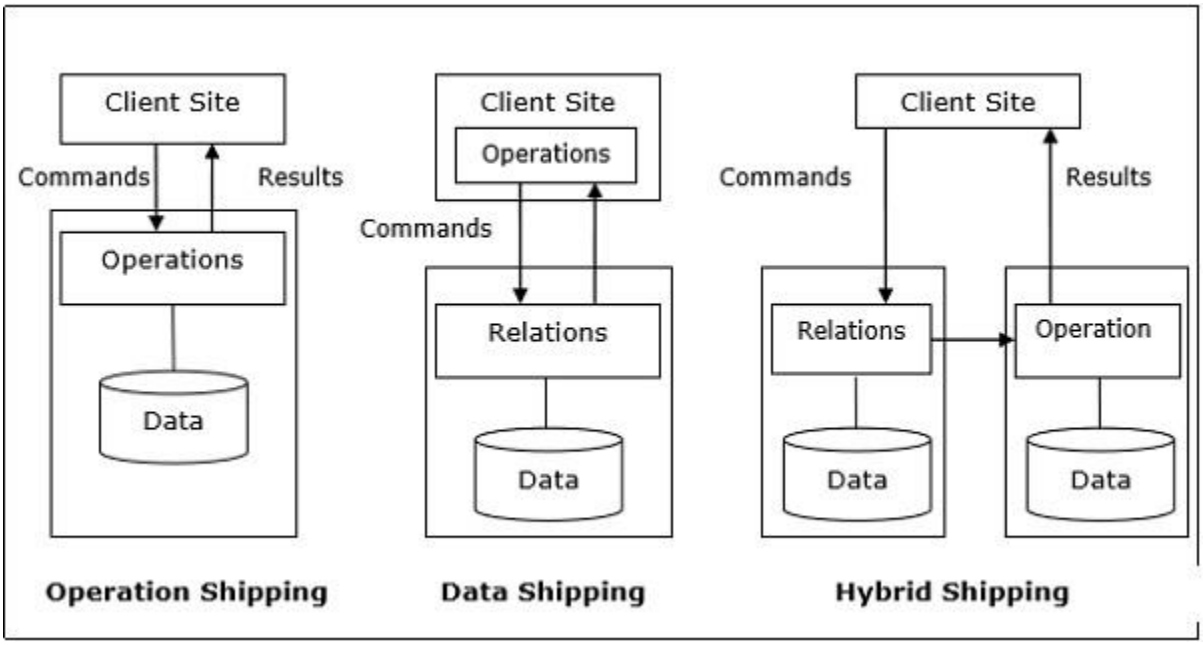

Figure 6: Optimal Utilization of Resources in the Distributed System

ii. Query Trading

In query trading algorithm for distributed database systems, the controlling/client site for a distributed query is called the buyer and the sites where the local queries execute are called sellers. The buyer formulates a number of alternatives for choosing sellers and for reconstructing the global results. The target of the buyer is to achieve the optimal cost. The algorithm starts with the buyer assigning sub-queries to the seller sites. The optimal plan is created from local optimized query plans proposed by the sellers combined with the

communication cost for reconstructing the final result. Once the global optimal plan is formulated, the query is executed.

iii. Reduction of Solution Space of the Query

Optimal solution generally involves reduction of solution space so that the cost of query and data transfer is reduced. This can be achieved through a set of heuristic rules, just as heuristics in centralized systems.

Following are some of the rules:

a. Perform selection and projection operations as early as possible. This reduces the data flow over communication network.

b. Simplify operations on horizontal fragments by eliminating selection conditions which are not relevant to a particular site.

c. In case of join and union operations comprising of fragments located in multiple sites, transfer fragmented data to the site where most of the data is present and perform operation there.

d. Use semi-join operation to qualify tuples that are to be joined. This reduces the amount of data transfer which in turn reduces communication cost.

e. Merge the common leaves and sub-trees in a distributed query tree.

4.3 Transaction Management Strategy

The global and local transaction management software modules, along with the concurrency control and recovery manager of a DDBMS, collectively guarantee the ACID properties of transactions.

As can be seen in Figure 2, an additional component called the **global transaction manager** is introduced for supporting distributed transactions. The site where the transaction originated can temporarily assume the role of global transaction manager and coordinate the execution of database operations with transaction managers across multiple sites. Transaction managers export their functionality as an interface to the application programs.

The manager stores bookkeeping information related to each transaction, such as a unique identifier, originating site, name, and so on. For READ operations, it returns a local copy if valid and available. For WRITE operations, it ensures that updates are visible across all sites containing copies (replicas) of the data item. For ABORT operations, the manager ensures that no effects of the transaction are reflected in any site of the distributed database. For COMMIT operations, it ensures that the effects of a write are persistently recorded on all databases containing copies of the data item. Atomic termination (COMMIT/ ABORT) of distributed transactions is commonly implemented using the two-phase commit protocol.

The transaction manager passes to the concurrency controller the database operation and associated information. The controller is responsible for acquisition and release of associated locks. If the transaction requires access to a locked resource, it is delayed until the lock is acquired. Once the lock is acquired, the operation is sent to the runtime processor, which handles the actual execution of the database operation. Once the operation is completed, locks are released and the transaction manager is updated with the result of the operation.

Two-Phase Commit Protocol

The *two-phase commit protocol* (**2PC**) requires a **global recovery manager**, or **coordinator**, to maintain information needed for recovery, in addition to the local recovery managers and the information they maintain (log, tables).

The two-phase commit protocol has certain drawbacks that led to the development of the three-phase commit protocol.

Three-Phase Commit Protocol

The biggest drawback of 2PC is that it is a blocking protocol. Failure of the coordinator blocks all participating sites, causing them to wait until the coordinator recovers. This can cause performance degradation, especially if participants are holding locks to shared resources. Another problematic scenario is when both the coordinator and a participant that has committed crash together. In the two-phase commit protocol, a participant has no way to ensure that all participants got the commit message in the second phase. Hence once a decision to commit has

been made by the coordinator in the first phase, participants will commit their transactions in the second phase independent of receipt of a global commit message by other participants. Thus, in the situation that both the coordinator and a committed participant crash together, the result of the transaction becomes uncertain or nondeterministic. Since the transaction has already been committed by one participant, it cannot be aborted on recovery by the coordinator. Also, the transaction cannot be optimistically committed on recovery since the original vote of the coordinator may have been to abort.

These problems are solved by the three-phase commit (3PC) protocol, which essentially divides the second commit phase into two sub-phases called:

i. Prepare-To-Commit
ii. Commit

Prepare-To-Commit Phase: It is used to communicate the result of the vote phase to all participants. If all participants vote yes, then the coordinator instructs them to move into the prepare-to-commit state.

Commit Sub-Phase: It is identical to its two-phase counterpart. Now, if the coordinator crashes during this sub-phase, another participant can see the transaction through to completion. It can simply ask a crashed participant if it received a prepare-to-commit message. If it did not, then it safely assumes to abort. Thus, the state of the protocol can be recovered irrespective of which participant crashes. Also, by limiting the time required for a transaction to commit or abort to a maximum time-out period, the protocol ensures that a transaction attempting to commit via 3PC releases locks on time-out.

The main idea is to limit the wait time for participants who have committed and are waiting for a global commit or abort from the coordinator. When a participant receives a pre-commit message, it knows that the rest of the participants have voted to commit. If a pre-commit message has not been received, then the participant will abort and release all locks.

4.3.1 Operating System Support for Transaction Management

The following are the main benefits of operating system (OS)-supported transaction management:

Typically, DBMSs use their own semaphores to guarantee mutually exclusive access to shared resources. Since these semaphores are implemented in user space at the level of the DBMS application software, the OS has no knowledge about them. Hence if the OS deactivates a DBMS process holding a lock, other DBMS processes wanting this lock resource gets queued. Such a situation can cause serious performance degradation. OS-level knowledge of semaphores can help eliminate such situations.

i. Specialized hardware support for locking can be exploited to reduce associated costs. This can be of great importance, since locking is one of the most common DBMS operations.
ii. Providing a set of common transaction support operations though the kernel allows application developers to focus on adding new features to their products as opposed to re-implementing the common functionality for each application. For example, if different DDBMSs are to coexist on the same machine and they chose the two-phase commit protocol, then it is more beneficial to have this protocol implemented as part of the kernel so that the DDBMS developers can focus more on adding new features to their products. Distributed databases incorporate transaction processing, but are not synonymous with transaction processing systems.

Figure 7: Distributed DBMS Architecture (Local Transaction Steps)

Local Transaction Steps

i. Application makes request to distributed DBMS

ii. Distributed DBMS checks distributed data repository for location of data. Finds that it is local

iii. Distribute DBMS sends request to local DBMS

iv. Local DBMS processes request.

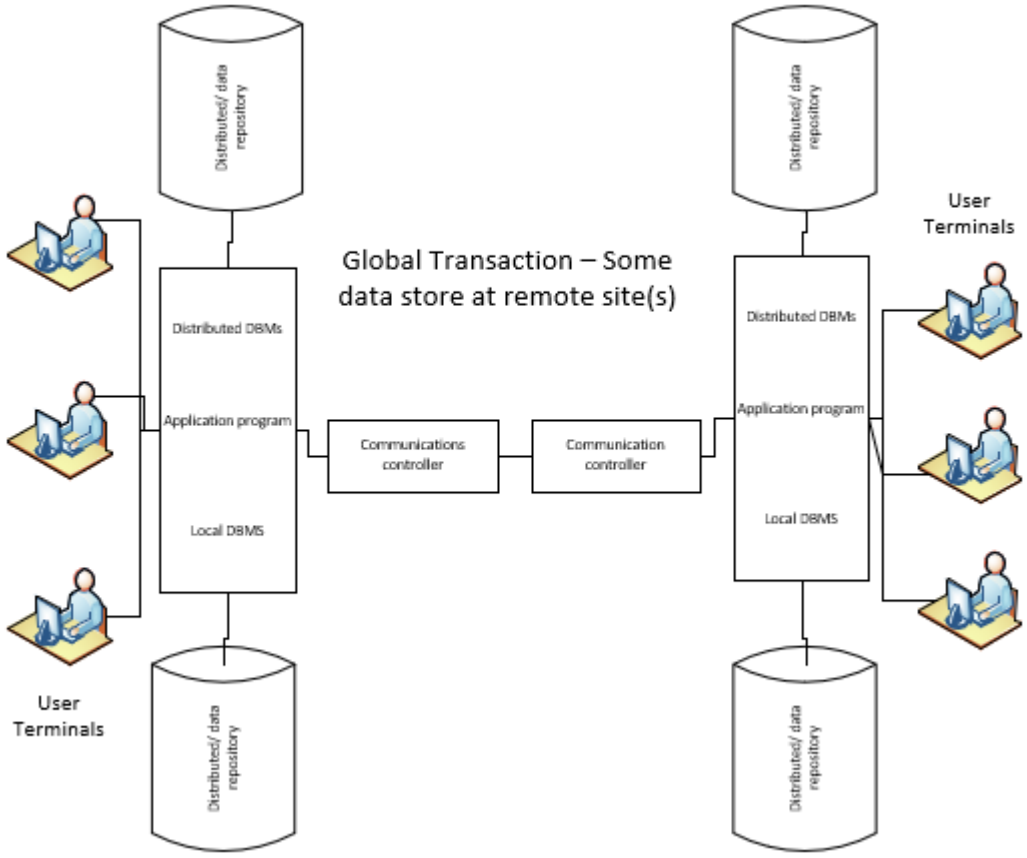

Figure 8: Distributed DBMS Architecture (Global Transaction Steps)

Distributed DBMS Architecture (Global Transaction Steps)

i. Application makes request to distributed DBMS
ii. Distributed DBMS checks distributed data repository for location of data. Finds that it is remote
iii. Distributed DBMS routes request to remote site
iv. Distributed DBMS at remote site translates request to local DBMS
v. Local DBMS at remote site processes request
vi. Local DBMS at remote site sends results to distributed DBMS at remote site
vii. Remote distributed DBMS sends results back to originating site
viii. Distributed DBMS at originating site sends results to application

5. Advantages of Distributed Databases

Organizations resort to distributed database management for various reasons (Sese, 2017). Some important advantages are listed below.

Improved Ease and Flexibility of Application Development: Developing and maintaining applications at geographically distributed sites of an organization is facilitated owing to transparency of data distribution and control.

Increased Reliability and Availability: This is achieved by the isolation of faults to their site of origin without affecting the other databases connected to the network. When the data and DDBMS software are distributed over several sites, one site may fail while other sites continue to operate. Only the data and software that exist at the failed site cannot be accessed. This improves both reliability and availability. Further improvement is achieved by judiciously replicating data and software at more than one site. In a centralized system, failure at a single site makes the whole system unavailable to all users. In a distributed database, some of the data may be unreachable, but users may still be able to access other parts of the database. If the data in the failed site had been replicated at another site prior to the failure, then the user will not be affected at all.

Improved Performance: A distributed DBMS fragments the database by keeping the data closer to where it is needed most. **Data localization** reduces the contention for CPU and I/O services and simultaneously reduces access delays involved in wide area networks. When a large database is distributed over multiple sites, smaller databases exist at each site. As a result, local queries and transactions accessing data at a single site have better performance because of the smaller local databases. In addition, each site has a smaller number of transactions executing than if all transactions are submit-ted to a single centralized database. Moreover, inter-query and intra-query parallelism can be achieved by executing multiple queries at different sites, or by breaking up a query into a number of sub-queries that execute in parallel. This contributes to improved performance.

Improved Potential for Share and Local Autonomy

The distribution of data can represent the geographical distribution of an organization; users at one site can access data that are stored on other sites. The data may be put in the nearby location of the users who use the data normally. This allows users to manage the data locally and thus create and enact local policies about the use of the data. The entire system is managed by a global database administrator (DBA). In general, the local level is assigned part of this responsibility so the local DBA is able to manage the local DBMS.

6. Disadvantages of Distributed Databases

Whilst organizations resort to distributed database management for various reasons. There are some important disadvantages which are listed below.

Complexity: A distributed DBMS which hides the distributing nature of user performance and reliability, is inherently more complicated than a centralized DBMS. It adds an additional amount of complexity to the distributed DBMS since the data can be repeated. If the software does not properly manage the duplication of data, the availability, reliability and efficiency of the software would become disadvantageous compared with the centralized system.

Control of Integrity Is Harder: The credibility of the database refers to the authenticity and accuracy of the data stored. Integrity is generally represented by limits, which are consistency laws that are not allowed to be broken in the database. Implementing honesty limitations requires normally access to a wide quantity of data that determines limitations. The communication and processing costs in a distributed DBMS are high in comparison to centralized systems, as these are needed to impose integrity constraints.

Cost: More complexity means that we should anticipate that the cost of the DDBMS to be higher than that of a centralized DBMS for procurement and maintenance. In order to create a network between the sites, a DBMS distributed requires additional hardware. The use of this network entails continuing connectivity costs. The management and maintenance of local DBMSs and the base network often includes increased labour costs.

Lack of Standards: While distributed DBMSs rely on efficient communication, the appearance of standard protocols for communication and data access is only now beginning to be seen. The lack of requirements greatly decreased the capacity of DBMSs distributed. No tools and methodologies are available to help users turn a centralized DBMS into a distributed DBMS.

Security: Access to the data can easily be managed in a centralized system. However, access to replicated data must not only be managed at different locations within a distributed DBMS but the network must also be guarded itself. Networks have historically been considered an unreliable platform for communication. Whereas partial progress has been made to secure networks, substantial advances have been made.

7. Conclusion

Clearly, the importance of DBMS design and distributed data independence in the database community is well known. In particular, in the multidimensional spatial knowledge bases, data independence is the key direction for analysis.

The key conclusion is that the modern style of knowledge organization needs special attention. In the article, we addressed the generalization and application of data independence techniques in a particular category of information bases (multi-dimensional numbered information spaces).

References

[1]. Abhijeet Raipurkar, G.R. Bamnote. (February 2013).Query Processing In Distributed Database Through Data Distribution. International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 2.

[2]. Codd E.F., Codd S.B., and Salley C.T. 1993. "Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate". Codd & Date, Inc.

[3]. Joseph Grech. (2009). Designing and Implementing a Distributed Database for a Small Multi-Outlet Business Regis University.

[4]. Patrick Valduriez. (2021). Principles of Distributed Data Management in 2020. INRIA and LIRMM, Montpellier – France.

[5]. Pudn. (2007). Distributed Database Systems: Distributed DBMS Architecture. Retrieved from:http://read.pudn.com/downloads158/sourcecode/database/703209/DDB/04.%20Distributed%20DBMS%20Architecture.pdf.

[6]. Sese Tuperekiye E. Zuokemefa Enebraye P. (2015) Framework For Client-Server Distributed Database System For An Integrated Pay Roll System, Department of Computer Science, Bayelsa State College of Arts and Science, Yenagoa, Bayelsa State, Nigeria.

[7]. Singole. (2016). Concept and Overview Distributed Database system. Retrieved from: https://www.ques10.com/p/17568/explain-distributed-database-architecture-1/, on 30 June 2020.

[8]. Thomas Haigh (2016). Historical Reflections How Charles Bachman Invented the DBMS, a Foundation of Our Digital World. His 1963 Integrated Data Store set the template for all subsequent database management systems. VOL. 59 NO. 7. Communications of the ACM.

[9]. Zloof MM. Query-by-Example: A Data Base Language. IBM Systems Journal 2017; 14:324-43.

