Journal of Advances and Scholarly Researches in Allied Education
Vol. 16, Issue No. 6, May-2019, ISSN 2230-7540

A Review of Real-Time Database System and Distributed Database System

Ashis Ranjan Paul[1]*, Dr Prerna Nagpal[2]

[1] Research scholar, CMJ University, Jorabat, Meghalaya

[2] Assistant Professor, CMJ University, Jorabat, Meghalaya

Abstract - Today's Real-Time Systems (RTS) are defined by managing massive amounts of scattered data making Real-Time distributed data processing a reality. Large company houses need to undertake dispersed processing for numerous reasons, and they frequently must do it in order to stay competitive. So, efficient database management methods and protocols for accessing and changing data are essential to fulfil temporal limitations of supported applications. Therefore, new research in Distributed Real-Time Database Systems (DRTDBS) is needed to study various approaches of adapting database systems technology to Real-Time systems. In this paper the Real-Time Database System and Distributed Database System are discussed.

Keywords - Real-Time Database System, Distributed Database System, Database

----------X----------

1. INTRODUCTION

The Relational Database Model was initially introduced by IBM researcher Dr. E.F. Codd in 1970, and it instantly gained attention and popularity for its simplicity. In this view, the database is shown as a Table / Relation. Data are gathered in a form of Tuple/ Record. Many common corporate database applications have been successfully constructed using this design. For more complex database applications, however, there are significant drawbacks to this approach. Examples of complex database applications include CAD/CAM & CIM, scientific research, telecommunications, Geographical Information Systems (GIS), multimedia, etc. Complex object hierarchies, extended transaction durations, and user-defined data types for storing graphic objects and massive text items distinguish these systems from traditional commercial applications. Object Oriented databases were required for these applications (OODBs). With an OODB system, you don't have to be constrained by data types and query languages when it comes to meeting these requirements. An advantage of OODB over Relational Databases is the ability to specify the structure of complex objects and the methods that may be applied to these objects. Additionally, OODBs are notable due of their strict adherence to Object Oriented programming principles. OOP languages, such as C++, SMALLTALK/JAVA, etc., are becoming more popular in software development, making it easier to integrate OODB into Object Oriented projects. ORION, IRIS, Open OODB, and others are among the prototypes. Examples of commercial databases are GEMSTONE/OPAL, ONTOS, and Objectivity. The ODMG, a standard framework for handling objects' data, is used extensively across the OODB. For our thesis project, we want to utilize the Object Oriented database rather of the relational one, due to the aforementioned advantages.

2. REAL-TIME SYSTEMS

The accuracy of Real-Time systems relies not only on logical correctness of computation but also on the time it takes to produce a certain output. System failure occurs if the system's time limitations are not satisfied. There must be logical and temporal consistency in Real-Time databases. As a result of this, it is necessary to maintain the authenticity of data items which represent the status of the environment regulated by the system. The qualities of Real-Time systems may be used to divide them into hard and soft systems.

3. HARD VS SOFT REAL-TIME SYSTEM

As long as you're in a hard Real-Time system, you can't be late (miss the deadline). If the deadline is missed, the consequences will be disastrous and very costly. This kind of system should be able to meet all of the predetermined time limitations. An aeroplane's digital fly-by-wire control system is an example of a Real-Time System that is challenging to operate. In the event of a catastrophic failure as a consequence of being late, a hole in the ground may develop. The lives of passengers and crew

members are at danger in the case of an aircraft malfunction.

The price will rise if a deadline is not met under a soft Real-Time system. It is not possible to have a catastrophic collapse. For example, if a video or audio decompression system is designed, the system has a rather high tolerance for missing deadlines. When these faults are corrected quickly, the presentation remains mostly intact, despite a short-term decrease in output quality. Glitchiest, on the other hand, is often accepted by customers as long as difficulties occur seldom and for short periods of time.

4. DYNAMIC VS. STATIC SYSTEMS

System requirements are known in advance and do not alter while a static system is running. The term "dynamic system" refers to a system that may have its specifications updated or added to even after it has been launched.

4.1 Real-Time Databases

Traditional databases are equipped to deal with the complexities of persistent data. Transactions ensure that the integrity of the data is maintained. Serializability is a typical criterion for these transactions. A database's transaction and query processing is targeted on increasing throughput and response time. Time-limited and time-validated data access are both necessary in many real-world applications. There are a number of instances of this, including telephone switching systems, stock trading algorithms and automated factories. These applications need the use of Real-Time Database Systems. Real-Time Database Systems have attracted the attention of both database researchers and those working on Real-Time Systems. Real-Time data management may take use of several database technology benefits, which is what motivates database researchers.

Data that is no longer relevant after a given amount of time or data that only applies for a certain amount of time are the most common types of temporal data in Real-Time databases. There are inherent constraints in Real-Time databases due of the time-sensitive nature of the data, as well as the response time requirements of a given environment. However, Real-Time databases, on the other hand, have been developed to meet the specific time constraints of the activities they are utilized in. When scheduling transactions in Real-Time database systems, both data consistency and transaction timeliness must be addressed. Because of the constraints imposed by having to continually monitor the environment, Real-Time database systems have timing requirements that must be met in order to make data available to the controlling system. From this, the idea of temporal consistency is born.

All the benefits of both traditional real time and Object Oriented databases will be combined in Object Oriented real time databases, but a number of current issues will be exacerbated. Many Real-Time apps deal with complex data in Real-Time mode. A Real-Time Object Oriented Data Base System can resolve many complex data difficulties (RTOODB).

5. TRANSACTIONS IN REAL-TIME DATABASES

In Real-Time databases, the nature of transactions may be defined in three ways:

1) **The way in which transactions make use of data:** There are three transaction types in a standard database system: Read, Write, and Update.

- Transactions that write to the database are known as write-only transactions.
- Update transactions generate new data and save it in the database.
- Transactions that can only be read and not written to send data from the database to actuators by reading.

2) **Constraints on time:** The nature of transactions is sometimes constrained by the response time of the system, while other times they are constrained by the need for temporal consistency. Periodicity requirements are a sort of temporal consistency. Time limits put on aperiodic transactions are the most common kind of system reaction requirements.

3) **Executing transactions on time:** It is very important. The impact of missing a transaction's deadline may also be used to differentiate across transactions. They're a mix of Hard, Soft, and Firm.

- Hard deadline transactions are ones whose failure to meet the deadline might have catastrophic consequences. The actions that fall under this category tend to be mission-critical ones, such as responding to life-or-death or environmentally hazardous emergencies.
- Transactions with a soft deadline still have some value after the deadline has passed. After a given number of beers, the value is usually zero.
- When a firm deadline transaction expires, the transaction's value reduces to zero, meaning that it has no value beyond the deadline.

In Real-Time systems, transaction processing is more complicated than in traditional database

systems. When dealing with time restrictions, understanding how transactions are scheduled and their timing is critical. In other words, periodicity constraints are a result of requiring data to be absolutely genuine.

6. DISTRIBUTED DATABASE

Distributed databases are systems in which logically connected databases are deployed over a network. It is a set of databases stored on multiple computers that typically appears to applications as a single database. Consequently, an application can simultaneously access and modify the data in several databases in a network. It is possible to administer distributed databases with the help of DBMSs, which are software systems that keep users informed about the status of their data. In certain cases, DDBMS (Distributed Database Management System) is referred to as DDBS (Distributed Database Management System). Distribution makes the design and execution of a system much more difficult. There must be an additional set of features that can be provided by a DDBS, as indicated in the diagram below.

- Accessing remote places and exchanging data between them through a communications network is possible.
- In order to maintain tabs on the DDBMS catalogue's data distribution and replication.
- In order to plan out the execution of queries and transactions that need access to data from several sources,
- A data item may be retrieved from one of the many copies.
- Maintaining a consistent data item is a primary reason for this.
- The distributed database's conceptual model must be maintained in order to avoid confusion.
- In order to recover from individual site failures as well as other issues, such as a communication link failure.

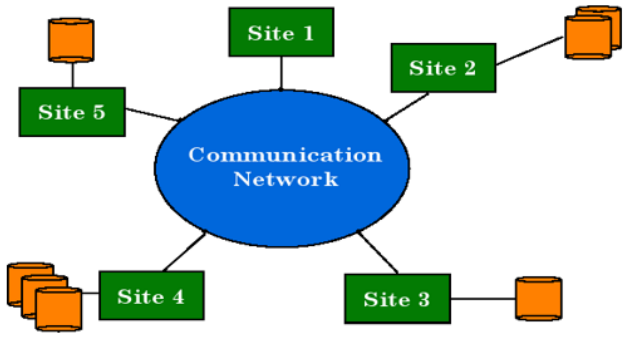
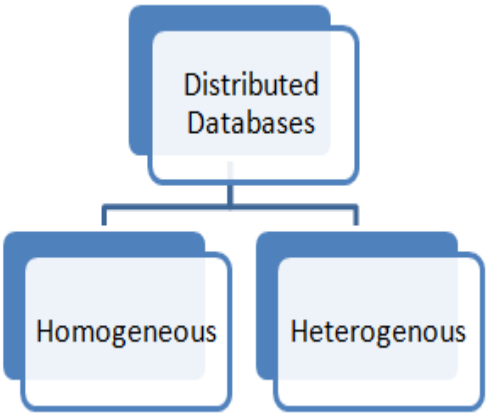


Figure 1: Architecture for a distributed database.

7. FEATURES OF DDBS

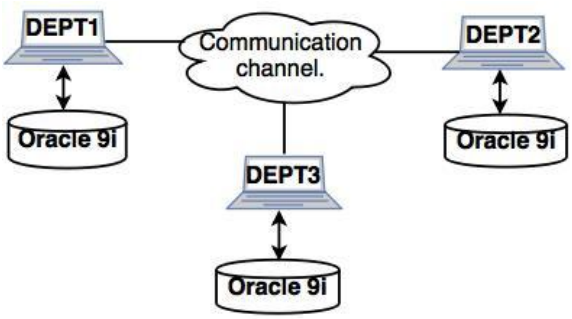
- Location independent
- Distributed query processing
- Distributed transaction management
- Hardware independent
- Operating system independent
- Network independent
- Transaction transparency
- DBMS independent

8. TYPES OF DISTRIBUTED DATABASES (BASED ON ARCHITECTURE)



Homogeneous Distributed Database:

In a homogenous distributed database system, all the physical locations have the same underlying hardware and run under the same operating systems and database applications. Homogenous distributed database systems appear to the user as a single system, and they can be much easier to design and manage.



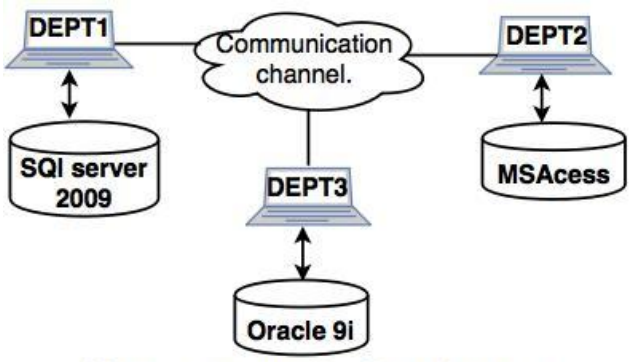
Homogeneous distributed system

Key Points:

- User queries may be processed across all sites since they all run on the same software platform and are aware of each other.
- Each site gives up a portion of its ability to make changes to its own software or schemas.
- A single system appears to the user.

Heterogeneous Distributed Database:

In a heterogeneous distributed database system, all the physical locations have not the same underlying hardware and run under the different operating systems and database applications. The hardware, operating systems or database applications may be different at each location. Different sites may use different schemas and software. Users at one location may be able to read data at another location but not upload or alter it.



Heterogeneous distributed system

Key Points:

- There may be a wide range of software and schemas used on various websites.
- Query processing is hindered by a lack of consistency in schema.
- Transaction processing is hampered by software differences.
- Only a limited number of transaction-processing capabilities are available for collaboration across sites.

9. COMPONENTS OF DISTRIBUTED DBMS

Figure 1.2 shows the components of a Peer-to-Peer Distributed Database Management System (DBMS). There are two main components: i) User Processor and ii) Data Processor. The contact with users is handled by one component, while the storage is handled by the other.

User Processor:

There are four parts to this processor, which is termed the user processor. These are as follows:

- **User Interface Handler:** As soon as a user command is received, the user interface handler interprets it and formats it so that the user can see it.
- **Semantic Data Controller:** To determine whether a user query may be handled, the semantic data controller refers to the integrity constraints and authorizations established in the global conceptual schema. Setting Permission is the responsible of this part.
- **Global Query Optimizer:** The global query optimizer and decomposer use the global and local conceptual schemas and the global directory to build an execution strategy to minimise a cost function and transform global queries into local ones. It is the job of the global query optimizer to come up with the most efficient way to carry out distributed join operations.
- **Global Execution Monitor:** The distributed execution monitor is responsible for coordinating the execution of a user request across several machines. The distributed transaction manager is also known as the execution monitor. When running queries in a distributed manner, the monitors at different locations may interact with one other.

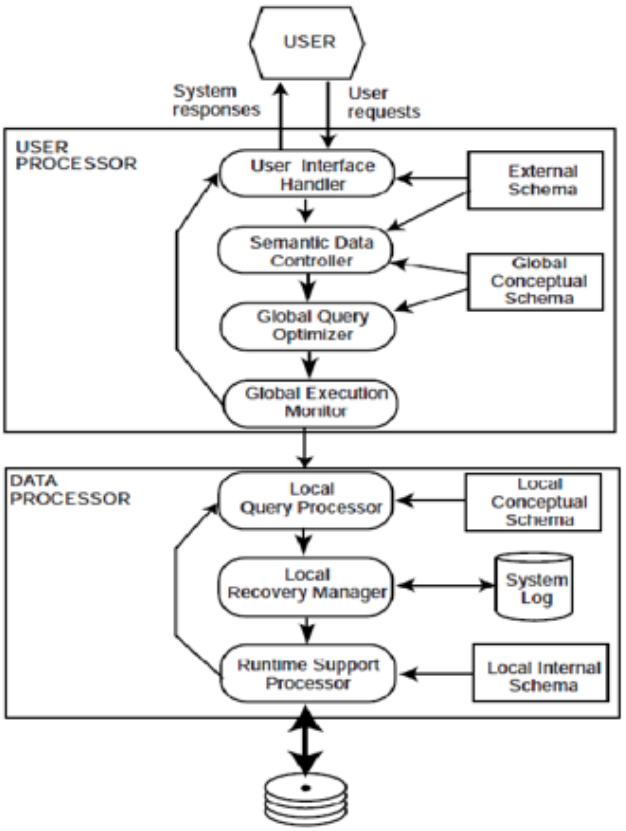


Figure 1.2: Components of a Distributed DBMS.

Data Processor:

There are three key components of the data processor in a distributed DBMS:

- **Local Query Processor:** The optimum route to any data item is selected by the local query optimizer, which is also the access path selector.
- **Local Recovery Manager:** Even in the event of a failure, the local recovery manager ensures the consistency of the local database.
- **Runtime Support Processor:** It executes the physical command schedule prepared by the query optimizer in order to get data from a database. The database buffer (or cache) manager is a part of the run-time support processor and is responsible for maintaining the primary memory buffers and handling data accesses to the database.

10. CONCLUSION

Distributed Real Time Database System's (DRTDBS) performance depend on various parameters such as transaction priority, buffer management, replica placement and replacement techniques as well as commit procedures, scheduling transactions with deadlines, Deadlocks, communication delays between different sites as well as predictability and consistency of data access mechanisms and image processing capabilities. It is our goal in this thesis to enhance the overall performance of the DRTDBS system by addressing some of the most critical concerns such as replica management; priority assignment policy; image processing; and dynamic buffer management. In the next sections, the conclusion of this thesis and the scope for future study are laid forth in further detail.

REFERENCES

1. Pratik Shrivastava, Udai Shanker (2019) on "Predicting Processing Time of Real Time Transaction in Replicated DRTDBS via Middleware", International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249-8958, Volume-8 Issue-5
2. Udai Shanker (2017) on "Replica Update Technique in RDRTDBS: Issues & Challenges", volume 4, no. 3
3. S. Kashyap 2016) on "Issues in Distributed Real Time Replication Database System", Corpus ID: 22320159
4. Ozgur Ulusoy (2012) on "Processing Real-Time transactions in a replicated database system", Distributed and Parallel Databases, volume 2, Number 3
5. Tarun, S., (2012). A Reputation Replica Propagation Strategy for Mobile Users in Mobile Distributed Database System. International Journal of Grid and Distributed Computing Vol. 5, No. 4
6. Khanli, L.M., Isazadeh, A., Shishavanc, T.N. (2010). PHFS: A Dynamic Replication Method, To Decrease Access Latency in Multi Tier Data Grid. Future Generation Computer Systems 27, 233–244.
7. Gorla, Narasimhaiah (2010). Sub query Allocations in Distributed Databases Using Genetic Algorithms. Journal of Computer Science and Technology, Vol.10 No.1, pp.31- 37.
8. Hauglid, Jon Olav, Norvald, H. Ryeng and Norvag,, K. (2010). Dynamic Fragmentation and Replica Management in Distributed Database Systems. Journal of Distributed and Parallel Databases, Vol. 28 No. 3, pp. 1- 25
9. Manoj Mishra (2009) on "Some Performance Issues in Distributed Real Time Database Systems", vol. 5, No. 6
10. Ceri, Houtsma, Keller, Samrati. (2008). A Classification of Update Methods for replicated Databases. http://citeseer.ist.psu.edu/ceri94classification.html
11. Huang, Y., Sistla, P., Wolfson, O.,(2008). Data Replication for Mobile Computers. Vol 5, pages 13-24, SIGMOD.
12. Gray, H., Neil, O., Shasha (2008). The Dangers of Replication and a Solution, International Conference on Management of Data. Proceedings of the 1996 ACM SIGMOD international conference on Management of data, Pages: 173 – 182.

Corresponding Author

Ashis Ranjan Paul*

Research scholar, CMJ University, Jorabat, Meghalaya