Seminar Report

On

Distributed Database System Based on Middleware

Submitted by

Ajumal PA (182167)

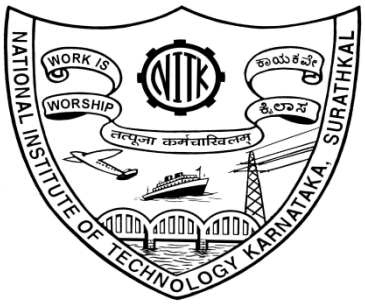
Under the Guidance of

NAME OF THE GUIDE

Dept. of Computer Science,

NITK, Surathkal

Date of Submission: 31 October 2018



Department of Computer Science
National Institute of Technology Karnataka, Surathkal.
2018-2019

Contents

1 Introduction

Databases has become an essential component in our everyday life. Actually in a day, the number of times we make use of database is enormous. If we take applications in the current society, starting with Google search, Whatsapp chat, Internet websites, Wikipedia and anything that you take will be from databases only. The fast access of them is the key to success in any business these days. People spend millions to take care of there databases consistent and fast.

Todays applications which uses in industries are not the old-styled applications which make use of batch processing kind, rather they are interested to do the online analytics or the distributed databases querying. Why it is happening can be a good question, Answer would be the volume of data has been increased very highly. It lead the entire world to talk about Big Data analysis. We should make use of a very sophisticated system to carry out the analysis actually. But affording such a system will not be possible for all. In that scenario distributed systems, which is a collection of set of cheap machines, which are configured to work as one will give the similar or better performance than one sophisticated system came into picture.

Database systems actually can be said as a collection of data or information which are shared among many users. two broadway to classify them would be Centralized and Distributed databases. Centralized can be think as a database which works from a single node where as distributed database system can be viewed as a collection of nodes which work together via communicating through network or some other means and the users can actually access this data from anywhere in the network as if it is single system working only for them but the performance and availability will be very high.[2,3,4]

2 Literature Survey

2.1 Challenges and Disadvantages in DDBS

Distributed database allows people to store their data any in database among the databases connected via network, the location will not be a matter. Various issues/challenges can be pointed out like network problem, deadlock problem, concurrency issues, data allocation problems and replication issues(it is not a exhaustive list)

2.1.1 Disadvantages of Distributed Database[5]

1) Difficulty:- Complexity of the database developer is very high compared to the centralized one. The transparency assurance is also a very hectic and complex task, because end of the work, user should feel it is like a normal single database system which gives very good performance. Maintenance of multiple nodes will also be a challenging task. Because networking them to a cluster itself will be a complicated task. Extra work will have to be done to make the loosely coupled machines to work perfect. Take the example of Joins, It will be highly expensive operation in distributed environment.

2) Cost :- The number of machines and the complexity to interconnect them and maintaining them is a tedious task and this will increase the labour cost.

3) Security :- All the servers, databases, network and all the communication system should be secured to make sure that the entire distributed database system is secure which is a challenge. Encrypting the network can be seen as an example.

4) Integrity Maintenance :- In a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.

5) Lack of experience :- Properly practiced and experienced people in the distributed systems is very less due to the difficulty involved in understanding the concepts and configurations.

6) Database design complexity :- In addition to usual centralized database design problems, making a distributed database will have to consider about the load balancing, distribution of the data, communication among them and replication problems.(Not an exhaustive list)

7) Additional software/middleware is required.

8) OS should support distributed environment.

2.1.2 Challenges of Distributed Database[5]

1) Distributed-database - Concurrency-and-Consistency :: As we know distribution itself is a challenge. Distributed Concurrency Control tells that synchronization among the transaction is required to make sure the integrity does not get affected. To maintain Consistency in distributed database systems we will have to come up with techniques which is not used in the traditional systems. Timestamp based protocol must be the simplest and most effective one to use it.

2) Distributed-Deadlock-Management :: In distributed database many people will be accessing some resources almost all the time. So when some system try to access the data/ resource DB will have to check whether the resource is available or not. If not available the user will have to wait till it gets freed. But if the system all the process is in wait state and no process is getting progress means there is a deadlock. In distributed environment, finding and avoiding/preventing deadlock is a big challenge because the systems are loosely coupled.

3) Replication-Management :: We always prefer to have a copy of our data to recover from our loss. In Distributed database system the replication is very awesome feature will do the same thing I explained above. In DDS replication means select a small/entire portion of the data and store it in multiple locations. The challenge here is the consistency among the copies. So each write should make sure it will not make any consistency issues with any of the copy available in the entire database systems. So it really leads to a very expensive write scenario.

4) Operating-Environment :: To Implement Distributed Database Environment a Specific Operating System is requirement as per Organizational needs. Operating system plays and important role for managing the distributed database. Some time Operating system is not supported for Distributed database.

5) Transparent-Management :: The distribution of the data should not be visible to the user end. So transparency has to be managed to improve the usability of the system and also it should not forget about the consistency while implementing the transparency in the system.

6) Security-and-privacy :: Applying the security policies to the system is still remains a challenge since the number of systems and the location which they situated are unpredictable. The database administrator and the employees nearby any database server could be a breach of the data stored in the database. If we use the Internet as a communication medium among the nodes of system, it will enormously increase the complexity to assurance of the security and privacy. So we will have to implement some kind of encryption or other techniques to resolve all these problems.

7) Resource-management :: The heterogeneity of the system will increases the resource management complexity. We can not make one protocol and work for all. We will have to make one protocol for each different machines and there should be some kind of a bridge protocol to interact with one another.

2.2 Solution Proposed

The main issues and some solutions proposed are discussed in this section.

2.2.1 Distributed Concurrency Control and Consistency

We can use synchronization mechanism for concurrency and consistency issues. Actually we have 3 type of synchronization mechanisms possible.

Master-Slave mode Peer to Peer mode Hybrid Mode [8]

Master-slave synchronization mode talks about two different type of nodes and they have their own functionality. Master can actually take commands to update the database where as the slave can only read the database data. So there will be more than one slaves which will work under a single master. The performance for reading will be very high compared to the normal system because all the machines will be working for a single query of a user which will results in the reduction of processing time. The problem here is with the single master. A single master will lead to a bottleneck for write operation. And it will make the problem of single point of failure also. So this system is not robust as usual distributed systems.

In Peer to peer mode, all the database nodes are same in power. It means that any system in the network of distributed database can provide services to both update and read operations. It clearly shows that it does not have single point of failure problem and in turn it solves the fault tolerance problem. Here the challenge will be to assure the consistency of the data.

Hybrid mode is an existing mode which is available as a DBMS from Postgres called Postgres-XL. Its architecture is a mix of both master slave and peer to peer mode. It will have as many masters we want and as many slaves we want. Among masters it will be like peer to peer mode, that is we can update/query from all those masters and among slaves they are like usual slaves where it has only the read permission. Slaves are called datanodes, Masters are called coordinator and to make the transaction consistency we have global transaction manager. It is highly scalable and currently using in Industries.[8]

A centralized version of the solution for consistency problem could be like this. A transaction submits its operations to a Transaction Manager. The TM delivers each Read(x) or Write(x) operation of that transaction to the scheduler which is responsible to schedule which transaction should work first. When a scheduler tells which process to do Read(x) or Write(x), it sends the operation to its local DM (Data Manager), It will talk to the global manager and find out whether this operation is legal or not. If it is illegal it will be rejected. If legal, it can access data it requires and it returns/read the value or write if the required operation is write operation. The Commit or Abort operation is sent to all the other nodes where this transactions data present and being used. It has disadvantage of centralization in distributed environment[15][16]

To solve the problem of data consistency in distributed database system based on Peer-to-Peer model, My base paper[1] designed a middleware-based database synchronization System. (Middleware is computer software that provides services to software applications beyond those available from the operating system[9]).[1]

Majority of the existing distributed database systems using two-phase locking for concurrency issues and consistency problems. The two phase locking actually gives a slower performance than the approach mentioned in my base paper, which is a middleware based one.[1]

In this paper[1] When a transaction comes, Do it in the local database of the corresponding Node. In commit phase do this checking: If Transaction contains only Read operation:Commit the transaction. Else: (Push this transaction into a queue called writeQ. this writeQ copy will

be available with every node, so mutex will be implemented to make a write to this queue. Then each node will check for any conflicting operation happening in the node or not. If no conflict operations reported in any node: Do commit and report this commit to all the nodes.(Locks will be taken at checking time itself), Else: Abort the transaction.) Conflict Finding (In each Node):If the TS[i] ¡ element[i].commit or TS[i] ¡ element[i].read/write: Abort Transaction i. Else: Provide lock for that particular row/table/database. This algorithm using a middleware actually gave better performance than 2PL[1]. There are other kind of concurrency requirements in distributed real-time database systems (DRTDBS)[11]. In this kind of systems, all the requirement we mentioned above is applicable. But the difference is, in this kind, the transactions local consistency ensured faster than the global consistency. It really gives priority to local consistency first. But in usual databases, we make the global consistency as first priority than the local one, because we want our data to be correct among all the other users. The typical example for this kind of database requirement is, consider a robot which can do some mechanical work, say if it hits an operator operating this robot, we need to take the action as fast as possible. So we will ask the local database to commit as fast as possible without worrying much about the global synchronization and help the operator to get out of the danger.

2.2.2 Distributed Deadlock Management

When the different processes in distributed systems tries to get a resource, the problem of deadlock comes into picture. Deadlock refers to the situation in which some processes falls into a state where it cannot progress unless it gets the resources which is currently possessed by some other processes in the same state. The deadlock may be detected by analysing the request model of processes. There are mainly four request models- AND model, OR model, k out of n model, OR-AND model and the disjunctive k out of n model. In AND model, a passive process becomes active only when message is received from all the processes in dependent set. In OR model, the transition from passive to active happens when message is received from any of the processes in dependent set. K out of n model refers to the model in which the process becomes active upon receiving at least k requests from the processes in dependent set. OR-AND model talks about a set of processes and corresponding dependent sets. A process becomes active when message is received from all the resources in some dependent set. A process in disjunctive k out of n model becomes active when specific k messages are received from distinct processes in various dependent sets. We are using wait-for-graph(WFG) to find out the presence of deadlock. It also depends on the request model we are using. If a cycle is present in the AND request model, it denotes presence of deadlock.[10]

As the environment is distributed, the different resources and processes in the wait-for-graph may correspond to different sites. So it is very complex to detect deadlock in case of distributed systems. We can build DWFG(Distributed wait-for graph) which contains the information about all the sites, transactions and resources. Similarly LWFG(local wait-for graph) which contains the information about only one site can be built. The problem of deadlock comes into picture when a data item is replicated in different sites(say, two sites) and two processes tries to obtain lock on data item on each of the two sites. So, DWFG can be used to detect that. In that case the process in one site should wait for the lock in other site to be released and vice versa, which results in deadlock. In the event of deadlock, usually the youngest transaction is picked to abort. We should also consider the number of resources being used by the processes in deadlock and number of operations they have performed. We have to undo all of the operations performed by aborted transactions.[11] Deadlock detection can be performed using a centralized or hierarchical control.

Centralized control designates one site to build the DWFG. It is responsible for finding deadlock, if any. Other sites should find if there is any local deadlock using LWFG. In order to build the DWFG, each sites should send their LWFG. The global deadlock detector uses this information to build DWFG and analyse it to detect deadlocks. The main drawback of centralized approach is that the deadlock detection depends on the availability of one site. In hierarchical control, a tree like topology is used. Non-leaf nodes acts as Non Local Deadlock Detectors(NLDD) and leaf nodes acts as Local Deadlock Detectors(LDD). LDDs sends information to the NLDD present just above them in the hierarchy. NLDDs finds if there is any deadlock in their children. The root node can find the deadlock with respect to the whole system.[11][12]

Deadlock prevention can also be applied in of distributed systems which avoids need of deadlock detection. It is similar to the deadlock prevention concept in a multiprocessing system. The concept uses timestamp for this purpose. In nonpreemptive method, older transactions waits for younger transaction to finish and younger transactions cant wait(It will get abort signal) for older transactions. In preemptive method, younger transactions wait for older ones and older transaction preempts younger transactions.[11][13]

2.2.3 Resource management

One paper presents a Master Token Resource Management Algorithm for Distributed System. It is applicable for systems where mutual exclusion is necessary and only one node is able to access a resource at a given time. The algorithm maintains only one token for each resource in the network for management of resources. The token contains access rights for the resource. The process having the token can have the rights specified in the token with respect to the resource. If a site requests for a resource which is being used by another site, then it is notified that the resource is busy. In that case, a claim stamp is made on the resource in the token. When a token is given to another site, the information of the token will be stored on that site and an approval packet is sent from the preceding site. The process which is requesting a resource will make an entry in the claim queue. In the event of a site going down, a new token will be created. The access to a resource is given if it is available in the token and the process is the first entry in the resource claim queue. In case the resource is having a high demand, then we should give the token in a time-sharing basis. Otherwise, it may lead to starvation if a process with very long execution time is given the resource. In the time sharing scenario, the access rights of the process will be revoked in the event of timer expiry.[17]

2.2.4 Replication Management

The replication is an inevitable concept in distributed database systems. As we know the whole point of introducing a distributed system is to increase performance, reliability and fault tolerance. To increase all of those, we require replication management. But there are problems with the same, including network overhead, additional storage requirement, concurrency control protocols and much more. The write operation here will be costlier than the usual database system, on the other hand, read operation will be faster, because the user can use the local copy if possible. So we need to find a tradeoff between the challenges and goodput. So a good concurrency control system has to be implemented to rectify all these problems and get the maximum throughput.[21]

Operating environment of the DDBS can be a distributed system environment or Cloud computers or cluster of machines with well managed security and firewalls [18]. The security and privacy issues can be elaborated as main four aspects. 1 confidentiality (information we have should not

be allowed to read/use by anybody else), 2 integrity (information available among all of the users should be same and should not get tampered by unauthorized users), (3) inference (the information given to public should not lead to the actual information prediction), and (4) authentication (user should understand to whom is interacting with(without any doubt)). Confidentiality in the database will be ensured by the view and other mechanisms, whereas integrity on the data can be enforced by integrity constraints and triggers. An access control mechanism will tell who will use what functionality. It can be written with some of the access control flags which will help the system to understand what should be allowed to each users. In database, cloud or any other distributed and centralized systems, the access control format will be available where the administrator has to tweak it to get required access control mechanisms. Modern systems allows much more complicated access control level in which we can define what application should run and what should be security level while running the same application and so on.

There are different types of transparencies such as Access, Concurrency, Location, Replication, Failure, Migration, Performance, Scaling, Execution, Configuration, Network, and Naming. And there are three type of levels transparency which is Fragmentation, Location, and Local mapping[20]. How they are implemented is not included in this document.

3 Conclusion

I have read papers and text books related to issues in Distributed Database Systems. I concentrated on Peer-to-Peer model and gave an idea how to solve synchronization problem using a middleware-based database system. My seminar covered peer to peer model, master-slave model and hybrid model for distributed systems. I also discussed the improvements made by this middleware system comparing to the existing two phase locking protocol. I also discussed about current industrial standards and the gap between my base paper and the current standards.

References

[1] 2015 8th International Conference on Intelligent Computation Technology and Automation "Research and Design of Distributed Database Synchronization System Based on Middleware" Jue Wang, Dong-song Zhang, China.1. 2015 8th International Conference on Intelligent Computation Technology and Automation.

[2] Distributed Databases: Principles and Systems : Stefano Ceri, Giuseppe Pelagatti.

[3] Fundamentals of Database systems Fifth edition : Ramez Elmasri, Shamkant B. Navathe.

[4] Distrib Parallel Databases (2008) 23: 127149 DOI 10.1007/s10619-008-7024-5 Distributed real time database systems: background and literature review Udai Shanker Manoj Misra Anil K. Sarje Published online: 26 January 2008 : Springer Science+Business Media, LLC 2008.

[5] Distributed Database Problem areas and Approaches D.S.Hiremath, Dr.S.B.Kishor IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661,p-ISSN: 2278-8727 PP 15-18 www.iosrjournals.org

[6] Andrew S Tanenbaum, - Distributed systems principles and paradigms

[7] Ceri, S Pelagatti, G, - Distributed Databases: Principles and Systems

[8] https://www.postgres-xl.org

[9] Introduction to middleware Web Services, Object Components, and Cloud Computing : Letha Hughes Etzkorn

[10] Brzezinski J.Helary J.M.RaynalM.Singhal M Deadlock Models and a General Algorithm for Distributed Deadlock Detection,Journal of Parallel and Distributed Computing Volume 31, Issue 2, December 1995

[11] Stefano Ceri and Giuseppe Pelagatti. 1984. Distributed Databases Principles and Systems. McGraw-Hill, Inc., New York, NY, USA

[12] R. Muntz and D. Menasce, "Locking and Deadlock Detection in Distributed Databases," in IEEE Transactions on Software Engineering, vol. 5, no. , pp. 195-202, 1979.

[13] Ron Obermarck. 1982. Distributed deadlock detection algorithm. ACM Trans. Database Syst. 7, 2 (June 1982), 187-208. DOI=http://dx.doi.org/10.1145/319702.319717

[14] S. Andler, J. Hansson, J. Mellin, J. Eriksson, and B. Eftring. An overview of the DeeDS real-time database architecture. In Proc. 6th Intl Workshop on Parallel and Distributed RealTime Systems, 1998.

[15] J. Tao and J. G. Williams, "Concurrency control and data replication strategies for large-scale and wide-distributed databases" Proceedings Seventh International Conference on Database Systems for Advanced Applications. DASFAA 2001, Hong Kong, China, 2001, pp. 352-359. doi: 10.1109/DASFAA.2001.916397

[16] A. R. Ali and H. M. Harb, "Two phase locking concurrency control in distributed database with N-tier architecture," International Conference on Electrical, Electronic and Computer Engineering, 2004. ICEEC '04., Cairo, Egypt, 2004, pp. 149-153.

[17] Master Token Resource Management Algorithm for Distributed System Time Sharing Strategy: Abhijeet Uday Gole, 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies.

[18] J. Han, M. Song and J. Song, "A Novel Solution of Distributed Memory NoSQL Database for Cloud Computing," 2011 10th IEEE/ACIS International Conference on Computer and Information Science, Sanya, 2011, pp. 351-355.

[19] M. -. Lai, "Security on database systems and distributed databases," Proceedings. 25th Annual 1991 IEEE International Carnahan Conference on Security Technology, Taipei, Taiwan, 1991, pp. 219-220. Doi: 10.1109/CCST.1991.202216

[20] Nitesh Kumar, Saurabh Bilgaiyan , Santwana Sagnika - An Overview of Transparency in Homogeneous Distributed Database System ISSN: 2278 1323 International Journal of Advanced Research in Computer Engineering and Technology (IJARCET) Volume 2, Issue 10, October 2013

[21] IEEE Transactions on knowledge and data engineering, vol. 2. no. 2. June 1990 Analysis of Replication in Distributed Database Systems Bruno Ciciani, Daniel M Dias and Philip S Yu