ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Scalable Commit Protocol for Authorized and Secure Transactions over Clouds

Shaista Mubeen
M.Tech Student (CSE)
Kottam College of Engineering,
Chinnatekuru (V), Kallur (M),
Kurnool District-518218.

Mrs.V.Sujatha
Assistant Professor
Kottam College of Engineering,
Chinnatekuru (V), Kallur (M),
Kurnool District-518218.

ABSTRACT:

Transaction management in homogeneous distributed database system generates complexity and creates replication and distribution of data. It has been widely used in the area of distributed database system with multiple sites. One phase commit protocol was commonly used in transaction management. When a transaction runs across multi sites one site may commit and another one may fail due to an inconsistent state of the transaction. The choice of commit protocol is an important design decision for distributed database system. A commit protocol in a distributed database transaction which should uniformly commit to ensure that all the participating sites agree to the final outcome and the result may be either a commit or an abort situation. In this paper, we have enhanced the one phase commit protocol based on multi phase commit protocol. This phases either the "commit" or the "abort" both sides which results the query process of transaction management.

Index Terms—Database system, transaction management, Homogeneous distributed database system, two phase commit protocol.

1. Introduction

A Transaction is a sequence of operations that takes the database from a consistent state to another consistent state. It represents a complete and correct computation. Two types of transactions are allowed in our environment: query transactions and update transactions. Query transactions consist only of read operations that access data objects and return their values to the user [1-4]. Thus, query transactions do not modify the database state. Two transactions conflict if the read-set of one transaction intersects with the write-set of the other transaction. During the voting process [3], Update transactions consist of both read and write operations. Distributed database system is a technique that is used to solve a single problem in a heterogeneous computer network system. A major issue in building a distributed database system is the transactions atomicity [5-8]. When a transaction runs across into multi sites. It may happen that one site may commit and other one may fail due to an inconsistent state of transaction. Multi-phase commit protocol is widely used to solve these problems [7]. The choice of commit protocol is an important design decision for distributed database system. A Commit protocol in a distributed database transaction should uniformly commit to ensure that all the participating sites agree to the final outcome and the result may be either a commit or an abort situation.

In this work mainly centered on the Simulation of the M-PC for ensuring atomicity in distributed transactions [9]. RMI was used in our message-passing and communication model, instead of using Socket to handle communication. Some other considerations related to this protocol are also taken into account and improved upon in order to construct an optimized Simulation [8]. In a generic system is simulated in a distributed environment to represent the real world scenario more widely. the fact that distributed transaction processing systems are widely used in many different organizations of varying size, as well as

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

the nature of task distribution in a networking environment.

Transaction Management is an old concept in distributed database management systems (DDBMS) research [11]. However, Oracle was the first commercial DBMS to implement a method of transaction management the multi phase commit. Though it was very difficult to obtain information on Oracle's implementation of this method. Many organizations do not implement distributed database because of its complexity [10]. However, with global organizations and multi-tier network architecture, distributed implementation becomes a necessity. Organization in the implementation of distributed databases when installing Oracle DBMS, or encourage organizations to migrate from centralized to distributed DBMS. Universities could also contribute to this process by having graduates with the knowledge of Oracle DBMS capabilities.

In terms of transactions management, the system most related to Camelot is Argus and Quicksilver. Camelot has taken certain techniques, especially those related to communication support, from another IBM research system [12], R*. Camelot and Argus have nearly the same transaction model; the only major difference is that in Argus a transaction can make changes at only one site. Diffusion must be done within a nested transaction [13]. Argus has paid close attention to the performance of their implementation of multi-phase commit.

2. DISTRIBUTED DATABASE SYSTEMS (DDBS)

Distributed DBMS can also be integrated as a multiple process, single data n/w called MPSD, to allow more than one computer to access a single database. Large corporations may require an enterprise database to support May users over multiple departments. This would require the implementation of a multiple process, multiple data scenario, or MPMD, in which many computers are linked to a fully distributed client/server DDBMS. The DDBMS offer more reliability by decreasing the risk of a single site failure. If one computer in the n/w fails, the workload is distributed to the rest of the computers.

Furthermore, a DDBMS allows replication of data among multiple sites, data from the failed site may still be available at one sites. In a centralized database can be implemented as a single process, single data scenario or SPSD [14], in which one computer is linked to the host DBMS to retrieve data .A centralized DBMS different because a failed computer that houses the database will debilitate the entire system. A distributed database is a database in which storage devices are not all attached to a common processing unit such as the CPU. Controlled by a distributed database management system (together sometimes called a distributed database system). It may be stored in multiple computers, located in the same physical location; or may be dispersed over of interconnected computers. Unlike parallel systems, in which the processors are tightly coupled and constitute a single database system, a distributed database system consists of loosely-coupled sites that share no physical components. System administrators can distribute collections of data (e.g. in a database) across multiple physical locations.

A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks [15]. Because they store data across multiple computers, distributed databases can improve performance at end-user worksites by allowing transactions to be processed on many machines, instead of being limited to one. Which all above are given in figure I.

2.1 Merits of Distributed database system (DDBS)

- Management of distributed data with different levels of transparency like network transparency, fragmentation transparency, replication transparency, etc.

- Increase reliability and availability.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

- Economics — it may cost less to create a network of smaller computers with the power of a single large computer.
- Hardware, operating system, networks, fragmentation, DBMS, replication and location independence.
- Distributed query processing can improve performance.
- Distributed transaction management.
- Single-site failure does not affect performance of system.
- All transactions follow A.C.I.D. property: A-atomicity, the transaction takes place as a whole or not at all.

C-consistency, maps one consistent DB state to another. I-isolation, each transaction sees a consistent DB. D-durability, the results of a transaction must survive system failures.

2.2 Demerits of Distributed DBS

Economics — increased complexity and a more extensive infrastructure means extra labour costs Security — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (for example, by encrypting the network links between remote sites).
Difficult to maintain integrity — but in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible Additional software is required.

2.3 Types of DDBMS

- Homogeneous the same DBMS (eg.Oracle) is used at each node.
- Heterogeneous Potentially different DBMS (eg.Oracle and DB2) are used at each node.

Fig I: Distributed DBMS architecture

3. FUNDAMENTAL OF TRANSACTION MANAGEMENT

Transaction Management deals with the problems of keeping the database in a consistent state even when concurrent accesses and failures occur.

A. What is a Transaction?

A transaction consists of a series of operations performed on a database. The important issue in transaction management is that if a database was in a consistent state prior to the initiation of a transaction, then the database should return to a consistent state after the transaction is completed. This should be done irrespective of the fact that transactions were successfully executed simultaneously or there were failures during the execution. A transaction is a sequence of operations that takes the database from a consistent state to another consistent state. It represents a complete and correct computation. Two types of transactions are allowed in our environment: query transactions and update transactions. Query transactions consist only of read operations that access data objects and return their values to the user. Thus, query transactions do not modify the database state. Multi transactions conflict if the read-set of one transaction intersects with the write-set of the other transaction. During the voting process, Update

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

transactions consist of both read and write operations. Transactions have their time-stamps constructed by adding 1 to the greater of either the current time or the highest time-stamp of their base variables. Thus; a transaction is a unit of consistency and reliability. The properties of transactions will be discussed later in the properties section. Each transaction has to terminate. The outcome of the termination depends on the success or failure of the transaction. When a transaction starts executing, it may terminate with one of multi possibilities: 1.The transaction aborts if a failure occurred during its execution 2. The transaction commits if it was completed successfully example of a transaction that aborts during process 2 (P2). On the other hand, an example of a transaction that commits, since all of its processes are successfully completed.

B. Properties of Transactions

A Transaction has four properties that lead to the consistency and reliability of a distributed data base. These are Atomicity, Consistency, Isolation, and Durability.

Atomicity

This refers to the fact that a transaction is treated as a unit of operation. Consequently, it dictates that either all the actions related to a transaction are completed or none of them is carried out. For example, in the case of a crash, the system should complete the remainder of the transaction, or it will undo all the actions pertaining to this transaction. The recovery of the transaction is split into multi types corresponding to the two types of failures: the transaction recovery, which is due to the system terminating one of the transactions because of deadlock handling; and the crash recovery, which is done after a system crash or a hardware failure.

Consistency

Referring to its correctness, this property deals with maintaining consistent data in a database system. Consistency falls under the subject of concurrency control. For example, —dirty data‖ is data that has been modified by a transaction that has not yet committed. Thus, the job of concurrency control is to be able to disallow transactions from reading or updating "dirty data".

Isolation

According to this property, each transaction should see a consistent database at all times. Consequently, no other transaction can read or modify data that is being modified by another transaction. If this property is not maintained, one of two things could happen to the data base. a. Lost Updates: this occurs when another transaction (T2) updates the same data being modified by the first transaction (T1) in such a manner that T2 reads the value prior to the writing of T1 thus creating the problem of loosing this update. b. Cascading Aborts: this problem occurs when the first transaction (T1) aborts, then the transactions that had read or modified data that has been used by T1 will also abort.

Durability

This property ensures that once a transaction commits, its results are permanent and cannot be erased from the database. This means that whatever happens after the COMMIT of a transaction, whether it is a system crash or aborts of other transactions, the results already committed are not modified or undone.

4. HOMOGENEOUS DISTRIBUTED DBMS

In homogeneous distributed database system, the sites involved in distributed DBMS use the same DBMS software at every site but the sites in heterogeneous system can use different DBMS software at every site. While it might be easier to implement homogeneous systems, heterogeneous systems are preferable because organizations may have different DBMS installed at different sites and may want to access them transparently. Distributed DBMS can choose to have multiple copies of relations at different sites or choose to have only one copy of a relation. The benefits of data replication are increased reliability - if one site fails, then other sites can perform queries for the relation. The performance will increase, as transaction

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

can perform queries from a local site and not worry about network problems. The problem with data replication is decreased performance when there are a large number of updates, as distributed DBMS have to ensure that each transaction is consistent with every replicated data. This adds additional communication costs to ensure that all copies of the data are updated at the same time.

A homogeneous environment is typically defined by the following characteristics (related to the non-autonomous category described previously):

- Data are distributed across all the nodes.
- The same DBMS is used at each location.
- All data are managed by the distributed DBMS (so there are no exclusively local data).

5. Multi PHASE COMMIT PROTOCOL

The Multi-phase commit (M-PC) protocol is a distributed algorithm to ensure the consistent termination of a transaction in a distributed environment. Thus, via M-PC a unanimous decision is reached and enforced among multiple participating servers whether to commit or abort a given transaction, thereby guaranteeing atomicity. The protocol proceeds in multi phases, namely the prepare and the commit phase, which explains the protocols name.

The protocol is executed by a sender process, while the participating servers are called participants. When the transaction initiator issues a request to commit the transaction, the sender starts the first phase of the M-PC protocol by querying—via prepare messages—all participants whether to abort or to commit the transaction The master initiates the first phase of the protocol by sending PREPARE (to commit) messages in parallel to all the receivers. Each receiver that is ready to commit first force-writes a prepare log record to its local stable storage and then sends a YES vote to the master. At this stage, the receiver has entered a prepared state wherein it cannot unilaterally commit or abort the transaction but has to wait for the final decision from the master. On the other hand, each receiver that decides to abort force-writes an abort log record and sends a NO vote to the master. Since a NO vote acts like a veto, the cohort is permitted to unilaterally abort the transaction without waiting for a response from the master. After the master receives the votes from all the receivers, it initiates the second phase of the protocol. If all the votes are YES, it moves to a committing state by force writing a commit log record and sending COMMIT messages to all the receivers. Each receiver after receiving a COMMIT message moves to the committing state, force-writes a commit log record, and sends an ACK message to the master. If the master receives even one NO vote, it moves to the aborting state by force-writing an abort log record and sends ABORT messages to those receivers that are in the prepared state. These receivers, after receiving the ABORT message, move to the aborting state, force write an abort log record and send an ACK message to the master.

In a "normal execution" of any single distributed transaction, i.e., when no failure occurs, which is typically the most frequent situation; the protocol consists of multi phases:

A. The commit-request phase (or voting phase), in which a sender attempts to prepare all the transaction's participating processes (named participants, receivers, or workers) to take the necessary steps for either committing or aborting the transaction and to vote, either "Yes": commit (if the transaction participant's local portion execution has ended properly), or "No": abort (if a problem has been detected with the local portion), and

B. The commit phase (or completion phase), in which, based on voting of the receivers, the sender decides whether to commit (only if all have voted "Yes") or abort the transaction (otherwise), and notifies the result to all the receivers.
The receiver then follow with the needed actions (commit or abort) with their local transactional resources (also called recoverable resources; e.g.,

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

database data) and their respective portions in the transaction's other output (if applicable).

Success: If the sender received an agreement message from all receivers during the commit-request phase:
a) The sender sends a commit message to all the receivers.
b) Each receiver completes the operation, and releases all the locks and resources held during the transaction.
c) Each receiver sends an acknowledgment to the sender.
d) The sender completes the transaction when all acknowledgments have been received.

Failure: If any receiver votes No during the commit-request phase (or the sender's timeout expires):
a) The sender sends a rollback message to all the receivers.
b) Each receiver undoes the transaction using the undo log, and releases the resources and locks held during the transaction.
c) Each receiver sends an acknowledgement to the sender.
d) The sender undoes the transaction when all acknowledgements have been received.

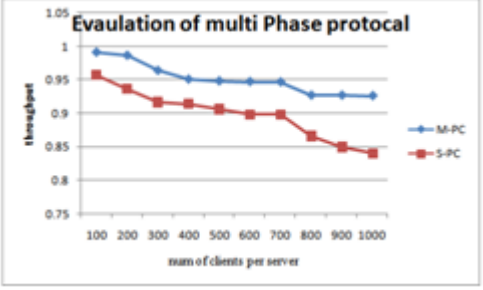

Fig 2: Throughput analysis of multi phase

6. Conclusion

Distributed Database Systems, where data is distributed and replicated. In this paper, based on two phase commit protocol technique are used for, when a transaction runs across into two sites. It may happen that one site may commit and other one may fail due to an inconsistent state of transaction. “Multi phase commit protocol (M-PC)” technique is widely used to solve these problem .The choice of commit protocol is an important design decision for distributed database system. A commit protocol in a distributed database transaction should uniformly commit to ensure that all the participating sites agree to the final outcome and the result may be either a commit or an abort situation.

REFERENCES

[1].This article incorporates public domain material from the General Services Administration document "Federal Standard 1037C".

[2].O'Brien, J. & Marakas, G.M.(2008) Management Information Systems (pp. 185-189). New York, NY: McGraw-Hill Irwin.

[3].Ozsu, Tamer M., and Valduriez, Patrick [1991], Principles of Distributed Database Systems, Prentice H.

[4].Mohan, C.; Lindsay, B.; and Obermarck, R. [1986], "Transaction Management in the R* Distributed Database Management System." ACM Transaction on Database Systems, Vol. 11, No. 4, December 1986, 379-395.

[5].G. Coulouris, J. Dollimore, T. Kindberg: Distributed Systems, Concepts and Design, Addison-Wesley, 1994.

[6].S. Ceri, M.A.W. Houtsma, A.M. Keller, P. Samarati: A Classification of Update Methods for Replicated Databases, via Internet, May 5, 1994.

[7].D. Agrawal, A.EI. Abbadi: The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. in Proc. of VLDB Conf. pp 243-254, 1990.

[8].W. Cellary, E. Gelenbe, and T. Morzy. Concurrency Control in Distributed Database Systems. North-Holland, 1988.

[9].R. Ramakrishnan. Database Management Systems. McGraw-Hill Book Company, 1998.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

[10].Oberg R., Mastering RMI: Developing Enterprise Applications in JAVA and EJB, John Wiley & Sons, 2001.

[11].Pitt E. and MCNiff K., java.rmi: The Remote Method Invocation Guide, Addison-Wesley, 2001.

[12].Oracle8 Server Distributed Database Systems, Oracle, 3-1-3-35.

[13].B. Lindsay et. al., Computation and Communication in R*: A Distributed Database Manager . ACM Trans. on Computer Systems, 2(1): 24-38, February 1984.

[14].R. Obermarck C. Mohan and B. Lindsay. Transaction Management in the R* Distributed Database Management System. ACM Trans. on Database Systems,11(4): 378-396, December 1986.

[15].B. H. Liskov B. M. Oki and R. W. Scheifler. Reliable Object Storage to Atomic Actions. In Proc. Tenth Symp. on Operating System Principles, page 147-159, ACM, December 1985.