International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.2, Issue.3, May-June 2012 pp-788-791 ISSN: 2249-6645

# An Efficient Non Blocking Two Phase Commit Protocol for Distributed Transactions

V. Manikandan[1], R. Ravichandran[1], R. Suresh[1], F. Sagayaraj Francis[2]
*(Department of Information Technology, Sri Manakula Vinayagar Engineering College, Puducherry, India
** (Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry, India

## ABSTRACT
**The 2-phase commit protocol is a standard algorithm for safeguarding the ACID properties of transaction in the distributed system. In distributed database systems (DDBSs), transaction blocks occurs during two-phase commit (2PC) processing if the coordinator itself fails and at the same time some client has declared itself ready to commit the transaction. Thus the blocking phenomena reduce the availability of the system, since the blocked transactions keep all the resources until they receive the final command from the coordinator after its recovery. To remove the blocking problem in 2PC protocol, three phase commit (3PC) protocol was proposed. Although 3PC protocol eliminates the blocking problem, it involves extra overhead of one more cycle and in turn increases the time taken for the transaction to complete. In this paper we proposed a new architecture for 2PC by employing a Backup coordinator, which will reduce the transaction blocking considerably. However in worst case, the blocking can occur in the Backup coordinator also. In such a rare case occurs, the client has to wait until the recovery of either the coordinator or the backup coordinator. This protocol suits best for DDBS environment in which transaction fail frequently and messages take longer time to deliver.**

***Keywords:*** **availability, blocking, distributed database system, two-phase commit**

## 1. INTRODUCTION
The world of computing is moving towards a trend where tasks are performed in a distributed manner. Distributed database systems implements a transaction commit protocol to ensure transaction atomicity. From last few decades a variety of protocols has been proposed by researchers. To achieve their functionality these commit protocols typically require exchange of multiple messages, in multiple phases, between client and coordinator where distributed transaction is done. A concurrency control mechanism is also applied to ensure synchronized access to various databases by many concurrently running transactions [1]. The performance factor of concurrency control algorithms depends on system throughput and transaction response time. Four cost factors influence the performance: inter-site communication, local processing, transaction restarts, and transaction blocking [3, 14].

The two-phase commit protocol allows the management of transactions in a distributed environment. In addition to that several log record are generated, some of which has to be forced write i.e. they are flushed to disk synchronously. Since two phase protocol suffers from a single point of failure, transaction blocking occurs. We are trying to overcome it by employing a Backup coordinator. Synchronization among the coordinator and Backup coordinator is maintained by connection manager.

In this fast world, transactions have to be committed successfully and data consistency has to be maintained. In 2PC, due to coordinator crash, the resource held locked and transactions are uncompleted till the recovery of coordinator. So resources are not able to use until it is being unlocked.

The 3PC protocol has one extra phase, called the *pre-commit phase*, compared to 2PC. It is this phase that makes this protocol non-blocking but it comes with the extra cost of message transfers [5].Even though 3PC overcomes this problem, 2PC has its own advantages. So we are trying to resolve the problem by using Backup coordinator which performs the functions of coordinator, when it crashes and so, resources are unlocked and transactions are committed successfully.

Here we have employed EJB to create modules and implement what we have proposed in the paper. Since EJB has its own transaction API's, it's easy to implement our concept and to perform transactions in an efficient manner.

## 2. LITERATURE SURVEY
In this section of this paper we have discussed the earlier defined protocols like presumption protocols, single phase commit protocol, optimized commit protocol and non-blocking commit protocol. The efficiency of a commit protocol is associated with the number of communication steps, the number of log writes and its execution time at the coordinator and at each participant. The Blocking or No blocking nature and difference in recovery procedures are other important factors that have a vital impact on the overall commit protocol performance. In this paper an attempt has been made to achieve equally good performance of protocol in the presence of failure.

### 2.1 Variants of 2PC
Both PA(presumed commit) and PC(presumed abort) seek to reduce commit process overhead by reducing acknowledge messages and forced log writes in the decision phase, while the voting phase remains the same as for 2PC. PrA is preferable where the number of aborted transactions is more than the number of committed transaction; PrC is preferred in systems where the number of committed transactions is more than the number of

aborted transactions, a common situation considering present system reliability. A detailed comparison between PrA and PrC is given in [6].

## 2.2 Non-Blocking commit protocol

A number of commit protocols have been designed to attack the fundamental blocking problem. Three-phase commit (3PC) [7, 8, 9] was among the first no blocking protocols. 3PC introduces a new "buffered phase" between the voting phase and the decision phase. In the buffered phase, a preliminary decision is reached about the result of a transaction. Cohorts can reach a global decision from this preliminary decision even in face of a subsequent master failure. However, 3PC achieves the non-blocking property at the expense of increased communication overhead by an extra round of message exchanges. Moreover, both master and cohorts must perform forced writes of additional log records in the buffered phase.

## 2.3 Single Phase Commit

The One-Phase Commit (1PC) protocol has been first suggested in [6] and several variations have been proposed. The *Early Prepare (EP)* protocol [10] forces each cohort to enter a prepare state after the execution of each operation. It makes cohort's vote implicitly YES and this protocol exploits the Presumed Commit as well [11]. But a coordinator may have to force multiple *membership* records, because the transaction membership may grow as transaction execution progresses. Above all, the main drawback comes from the fact that the log of each operation has to be written in the cohort's log disk per operation, it leads to a serious disk blocking time. Only if every sever has a stable storage so that log forces are free, EP can be considered to be used.

## 2.4 Optimistic commit protocol

Optimistic commit protocol [15] concentrates on reducing the lock waiting time by lending the locks the committing transactions hold. Since the lock lending is done in a controlled manner, there is no possibility of cascading aborts even if the committing transaction is aborted. This protocol has a good performance due to its reduction of the blocking arising out of locks held on prepared data.

The circumstances under which distributed transactions are committed or rolled back under the 2-PC protocol are [12].

- When application instructs the transaction to rollback, then the transaction will be roll backed.
- When process failure occurs before all participant votes, then transaction will be roll backed.
- If any participant votes no, then transaction will be roll backed.
- If all participant yes, transaction will be committed.
- If Process failure occurs after all participant have voted and the transaction coordinator has received all voters as yes, then transaction will be committed but is unresolved.

## 2.5 Comparison of 2PC and 3PC

Comparison between 2PC and 3PC regarding message exchanges, log writes and degree of blocking where n is the number of participants [2].

In case of 2PC message exchange is 4(n-1), log write is 2n and degree of blocking is high. Whereas in 3PC message exchange is 5(n-1), log write is 2n and degree of blocking is low.

It is the extra phase in 3PC can gives the extra n-1 message exchanges compared to 2PC. If the distributed system has a lot of transactions to be executed, this will become a significant performance loss [16].

## 3. NON-BLOCKING TWO PHASE COMMIT PROTOCOL

This section is going to deals with how 2PC can be implemented as a non-blocking protocol with the help of Backup coordinator. In normal case clients request will be processed by the coordinator and the transaction will be either committed or aborted.

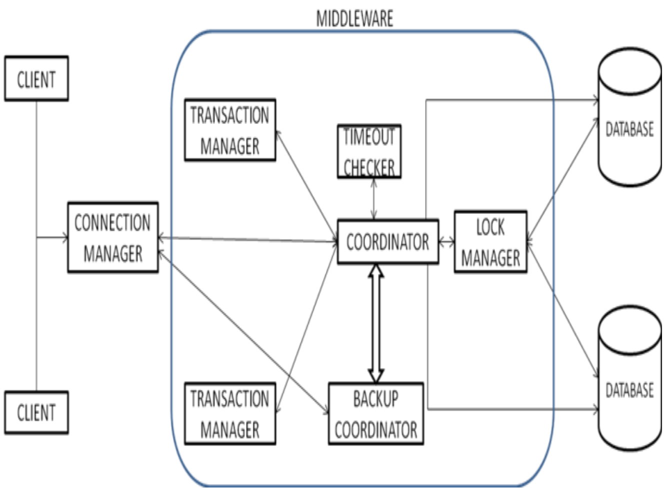

**Fig 1: System Architecture with Backup coordinator**

If suppose the coordinator fails means the transactions will be in a blocked state and resource is also said to be locked, clients has to wait infinite amount of time so this will affect the performance of the distributed system. In our model we included a new thing called as connection manager, will keeps on monitoring the coordinator and Backup coordinator, whenever the coordinator fails the transactions will be automatically transfer to the Backup coordinator with the help of connection manager and in vice verse. In turn connection manager will have a common log file for both coordinator and Backup coordinator. Synchronization between them will be achieved with the help of connection manager.

The components involved in the architecture are discussed below.

**1.** ***Transaction Coordinator*****:** Coordinates and executes atomic transactions and manages data transfer between its replica and other database.

**2. *Transaction manager*:** A transaction manager provides the services and management functions required to support transaction demarcation, transactional resource management, synchronization, and transaction context propagation.

**3. *Database*:** A database consists of a resource manager provides the application access to resources. The resource manager implements a transaction resource interface that is used by the transaction coordinator to communicate transaction association, transaction completion, and recovery work.

**4. *Connection Manager*:** Connection Manager provides a fast and transparent way of making connection. Here the connection manager will keeps on monitoring the coordinator and Backup coordinator, whenever the coordinator fails the transactions will be automatically transfer to the Backup coordinator with the help of connection manager and in vice verse. Users do not have to know which connection path is chosen.

**5. *Timeout checker*:** This is a thread whose responsibility is to watch for currently active transaction that have been inactive for too long, and abort them if so. Upon request by the TM, the checker records the current clock and associated it with the specified transaction. The thread constantly checks whether any transaction's time-to-live has expired, by looking at the difference between the current clock time and the transaction's "clock stamp". If this is greater than a large, fixed value, the timeout thread itself initiates abortion of the transaction.

We have implemented this model and these will be more reliable than the previous model, and so will increase the efficiency of the transaction processing.

We created log files which are used to lists actions that have occurred. Log files in turn said to contain the complete detail of the transaction that is being taken place. In 2PC log files are categorized into two types, they are transaction manager log file and coordinator log file. These files are maintained by connection manager in order to make synchronization among coordinator & backup coordinator and to survive from transaction failure.

## 4. PERFORMANCE DISCUSSION

### 4.1 Failure probability of backup coordinator while coordinator is down

Reliability of a module is statistically quantified as mean-time-to-failure (MTTF). The service interruption of a module is statistically quantified as mean-time-to-repair (MTTR). The module availability is statistically quantified from [5, 13] as:

$$\frac{MTTF}{MTTF + MTTR}$$

Let $MTTF_c$ and $MTTR_c$ represent MTTF and MTTR of the coordinator respectively. Also, $MTTF_b$ represents MTTF of corresponding backup coordinator. Since the backup coordinator and the coordinator are failure independent, the probability that backup coordinator fails when the corresponding coordinator is down is calculated as below.

The probability that the coordinator site is unavailable is:

$$P_c = \frac{MTTR_c}{MTTF_c + MTTR_c}$$

$$\approx \frac{MTTR_c}{MTTF_c} \text{ since } MTTR_c \ll MTTF_c$$

The probability that the backup coordinator fails is:

$$P_b = \frac{1}{MTTF_b}$$

The probability that backup coordinator fails and the corresponding coordinator is down is:

$$P_b \times P_c = \frac{1}{MTTF_b} \times \frac{MTTR_c}{MTTF_c} = \frac{MTTR_c}{MTTF_c \times MTTR_c}$$

From above equation, it can be observed that the probability that backup site fails while corresponding coordinator is down is reduced significantly. Thus, in case of coordinator site failure, with the introduction of the backup coordinator, blocking probability is considerably reduced as compared to 2PC protocol. Further, it can be observed that the purpose of the backup coordinator is to terminate the blocked transactions at the participant sites when the corresponding coordinator is down. After the termination of the blocked transactions, even though the backup coordinator fails, it does not affect the consistency of the database. Let term_time be the time duration required to terminate the blocked transactions by contacting the backup coordinator when the coordinator is down. The above equation denotes the probability that the backup coordinator fails during the entire period ($MTTR_c$) when the coordinator is down. However, in the worst case the blocked transactions are consistently terminated even if the backup coordinator is up only during term_time and then fails. As term_time (few minutes) is much less than the down time (few hours) of the coordinator, the probability that the backup coordinator fails during the term_time while the coordinator is down is further reduced.

## 5. CONCLUSION

Typically, only the coordinator node has all the information necessary to determine whether a transaction should commit or rollback. Therefore, if the coordinator node fails during a distributed transaction, all the participants in the transaction must wait for the coordinator to recover before completing the transaction. Thus, significant delays may be caused when the coordinator fails. To minimize the delay caused by the failed coordinator, some conversional transaction systems use clustering and / or group communication protocols to provide standby coordinators. However, Clustering protocols and group communication protocols add complexity to distributed transactions, and require a change to underlying distributed transaction protocol. In this paper we proposed a new architecture for 2PC relate to distributed transactions, and more specifically to improving reliability of Distributed Transactions.

Transaction processing must ensure transaction integrity for transactions that involve databases. Transactions often involve multiple steps, all of which must be completed before a database commit can be executed. Transaction Based Middleware is critical, because without them, it would be a very difficult job to write the programs necessary to track transactions across multiple platforms and databases. Some of the services provided by Transaction Based Middleware include the following: Transaction integrity, two-phase commits, failure/recovery, and load balancing.

*Load Balancing* is a feature of Transaction Based Middleware in which, the server component manages the workload presented by the clients by fully utilizing the resources available. Load balancing and thread management services are important because Transaction Based Middleware need to process many transactions on many different systems in a very short time period. The Middleware can change traffic patterns, processing parameters, or increase the pool of processors. This enables the middleware to dynamically adjust to the workload [4]. Transaction Based Middleware, generally utilize transaction priorities and multiple database sessions and/or threads to optimize throughput.

## ACKNOWLEDGEMENT

V Premanand, Department of Information Technology, Sri Manakula Vinayagar Engineering College, Puducherry.

## REFERENCES

**Journal Papers:**

[1] Toufik Taibi, Abdelouahab Abid, Wei Jiann Lim, Yeong Fei Chiam, and Chong Ting Ng, "Design and Implementation of a Two-Phase Commit Protocol Simulator", *The International Arab Journal of Information Technology, Vol. 3, No. 1,* January 2006.

[2] Byun T, Moon S, "Nonblocking two-phase commit protocol to avoid unnecessary transaction abort for distributed systems", Cheongryang 130-012 Seoul South Korea, *Journal of Systems Architecture. Volume 43, Issues 1-5,* March 1997, Pages 245-254.

[3] Arun Kumar Yadav and Ajay Agarwal,"A Distributed Architecture for Transactions Synchronization in Distributed Database Systems", *International Journal on Computer Science and Engineering Vol. 02, No. 06,* 2010.

[4] Tarek Helmy and Fahd S. Al-Otaibi," Dynamic Load-Balancing Based on a Coordinator and Backup Automatic Election in Distributed Systems", *International Journal of Computing & Information Sciences Vol. 9, No. 1,* April 2011.

[5] Tabassum k, Taranum F, Damodaram A"A Simulation of Performance of Commit Protocols in Distributed Environment", *in PDCTA, CCIS 203, pp. 665–681, Springer,* 2011.

**Thesis:**

[6] Gray J. N, "Notes on database operating systems". *Operating Systems: an Advanced Course, 60:397–405,* 1991.

[7] Skeen D, "Crash Recovery in a Distributed Database Systems", *PhD thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley,* 1982.

**Proceedings Papers:**

[8] Skeen D, "A quorum-based commit protocol", *in Proc. Of Berkeley Workshop, pages 69–80,* 1982.

[9] Skeen D, "Nonblocking commit protocols " *in Proceedings of the 1981 ACM SIGMOD international conference on Management of data,* Ann Arbor, Michigan, *pp. 133-142* .

[10] Stamos J.W and Cristian F, "A low-cost atomic commit protocol",*in 9th IEEE Symposium on Reliable Distributed Systems (SRDS'90), pages 66–75,* 1990.

[11] Houmaily Y. J. Al and Chrysanthis P. K, "The Implicit-Yes Vote Commit Protocol with Delegation of Commitment," *in Proceedings of 9th International Conference on Parallel and Distributed Computing Systems, pp. 804-810,* 1996.

[12] Boutros B. S. and Desai B. C., "A Two-Phase Commit Protocol and its Performance," *in Proceedings of the 7th International Workshop on Database and Expert Systems Applications, pp.100-105,* 1996.

[13] Reddy K, Kitsuregawa M, "Reducing the blocking in two-phase commit protocol employing backup sites", *IFCIS Conference on Cooperative Information Systems,* 1998.

[14] Meng Qingyuan, Wang Haiyang, Xu Chunyang,"A New Model for Maintaining Distributed Data Consistence", *International Conference on Computer Science and Software Engineering, IEEE* 2008.

[15] Lampson B and Lomet D "A new presumed commit optimization for two phase commit", *in 19th International Conference on Very Large Data Bases,* Doublin, Ireland, 1993.

**Books:**

[16] Silberschatz, Korth, Sudarshan, *"Database System Concepts, Fourth Edition, volume 1",* (The McGraw–Hill Companies, 2001).