Two New Protocols for Fault Tolerant Agreement

Poonam Saini[1] and Awadhesh Kumar Singh[2],
[1,2]Department of Computer Engineering, National Institute of Technology, Kurukshetra, India
nit.sainipoonam@gmail.com , aksinreck@rediffmail.com

ABSTRACT

The paper attempts to handle failures effectively, while reaching agreement, in a distributed transaction processing system. The standard protocols such as BFTDC [3], Zyzzyva [4] and PBFT [5] handle the problem to a greater extent. However, the limitation with these protocols is that they incur increased message overhead as well as large latency. Moreover, the nodes are evacuated from the transaction system after being declared faulty. We propose a novel proactive based agreement which identifies the tentative failures in the system. To improve the failure resiliency with minimum execution overhead, we also propose an optimized reactive view change mechanism. Both mechanisms have been analyzed and compared. The dynamic analysis of the protocol reflects that, in a faulty scenario, the proactive approach is computationally more efficient with reduced latency as compared to reactive one. Moreover, unlike PBFT and BFTDC, our agreement protocol runs in two phases, which leads to reduced message overhead and total execution time. The protocol treats the fail-silent (i.e. crashed) nodes in the system.

KEYWORDS

Distributed transactions, Two-phase commit, Byzantine agreement, Proactive view change, Reactive view change

1. INTRODUCTION

Transaction processing is a basic application of distributed computing. Like other applications, fault tolerance is a major concern in transaction processing also. Moreover, the transaction handling protocols should maintain atomicity, *i.e.*, either all operations of the transaction commit, or none of the operations is carried out, *i.e.* the transaction aborts. The standard commit protocol for distributed transaction is two-phase commit protocol, popularly referred as 2PC [1, 2], which works correctly in presence of benign faults only. The present work aims at designing an efficient agreement algorithm that successfully handles the fail-silent faults. In addition, it significantly reduces the time taken to complete the transaction. Two protocols, one based on proactive and the other on reactive approach, has been presented and analyzed. Both the protocols achieve increased system availability and enhanced throughput.

There has been an explosive growth in the number of connected hosts in recent times [3]. Also, the network hosts and servers are exposed to the public access through online transactions. It results in a number of lead chances of transactions have become the first mode of communication Traditional Byzantine fault tolerant protocols such as BFTDC [5] and Zyzzyva [6] deal with failures in a *reactive* manner, i.e., they rely on the specification of the faults to initiate view change. In a particular view, one of the replicas is chosen as primary and other replicas work as backups. In the middle of agreement, if time out occurs for current view because of delay in message propagation or the primary is found faulty then view change occurs. The *proactive* approach, on contrary, is designed to minimize the transaction discontinuity and latency while ensuring stability as well as availability of replicas through failure notifications in advance. Towards this goal, we build a system model to analyze the failure resiliency of our protocol under both reactive and proactive approaches.

1.1 Motivation

As the large scale distributed computing systems are more prone to failures, our protocol runs agreement on every request without involving the clients. It makes the protocol faster in the presence of increased number of faults and makes it more useful for large networks. In most of the contemporary works, the protocol replaces the replica from the system when it is diagnosed as faulty. This leads to increased message overhead for the overall execution of the protocol. Although, the protocols produce desired result, they incur latency in order to initiate the view change mechanism which results in short-lived (i.e., transient) halts during the transaction processing. We have attempted to devise a technique which is able to detect, in advance, the tentative fault in the system. The protocol fulfills all the requirements that are agreement, validity, and termination. We use a Transaction Manager, which itself is assumed to be trusted and fault free.

1.2 Organization of the paper

The section 2 describes the related work existing in the literature succeeded by system model and problem definition in section 3. Section 4 presents the analysis of the *reactive* and *proactive* view change protocol. A detailed discussion of optimized agreement protocol is given in section 5. In section 6, simulation results are shown followed by conclusion in section 7.

2. RELATED WORK

The Byzantine faults [4] are extremely challenging to tolerate, especially for long-running systems. Hence, although considerable amount of literature exist in this field, researchers have still interest. After many years of diligent work, Castro and Liskov [5] came up with three phase Byzantine agreement algorithm. Further, based on their approach, Zhao [6] introduced BFTDC protocol. These protocols handle the problem to a greater extent; however, they incur increased message overhead as well as large latency. We give a brief account of other noteworthy agreement protocols in the following section.

2.1 Zyzzyva: Speculative Byzantine Fault Tolerance [7]

The protocol uses speculation to reduce the message count and simplify the design of Byzantine fault Tolerant state machine replication. In Zyzzyva, the replicas optimistically adopt the decision proposed by the primary and respond immediately to the client’s request without running an expensive three-phase commit protocol to reach agreement. As a result, correct replicas’ states may diverge, and faulty replicas may send contradictory responses to clients. Nonetheless, the applications at client site observe the traditional and powerful abstraction of a replicated state machine that executes requests in a linearizable order. The replies carry with them sufficient history information for clients to determine whether the replies and history are stable and are guaranteed to be eventually committed. If a speculative reply and history are stable, the client uses the reply. Otherwise, the client waits until the system converges on a stable reply and history. The challenge in Zyzzyva is to ensure that the responses to correct clients become consistent. In fact, the replicas are responsible for ensuring that all requests from a correct client complete eventually.

2.2 An Optimistic Commit Distributed Transaction Protocol for Management [8]

The work is an improvement over traditional two phase commit protocol, popularly known as 2PC algorithm. Silberschatz-Korth'91, proposed a family of protocols, which eliminated the major disadvantage of potential unbounded delay in 2PC protocol that the transactions may have to endure if certain failures occur. By using compensating transactions, they proposed a revised 2PC protocol that overcomes these difficulties. In the revised protocol, locks are released as soon as a site votes to commit a transaction, thereby solving the indefinite blocking problem of 2PC. Finally, if the transaction is to be aborted, then its effects are undone semantically using a compensating transaction. Therefore, semantic, rather than standard, atomicity is guaranteed. Accordingly, a correctness criterion was proposed that was most appropriate when atomicity is given up for semantic atomicity. The protocols restricted only global transactions, and did not incur extra messages other than the standard 2PC messages. However, these protocols were not meant to handle any type failures in the system.

2.3 HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance [9]

HQ is also a state machine replication protocol designed to handle deterministic as well as arbitrary operations. The protocol employs best features of two categories of protocols, namely, the agreement based BFT and quorum based protocols. It uses query/update (Q/U) method in the absence of contention and BFT method, in case, contention occurs. In the normal case of no failures and no contention, write operations require two phases to complete while reads require just one phase. In the first phase, the client requests for a grant from each replica. If it receives 2f+1 matching replies it uses the same as certificate to approve the agreement decision to the replicas in second phase. However, if various replicas have granted the same sequence number to different clients, there would be write contention. In such case, BFT will be employed. Thus, it performs well in the absence of contention because of its speed. However, in real time situations, where contention is common, the protocol takes more time than BFT in decision making.

2.4 Commit Barrier Scheduling [10]

Commit barrier scheduling is a concurrency control protocol that allows the system to guarantee correct behavior while achieving high concurrency in the presence of Byzantine faults. The protocol constrains the order in which queries are sent to replicas in order to prevent conflicting schedules, while preserving most of the concurrency in the workload. Additionally, it ensures that users see only correct responses for transactions that commit, even when some of the replicas are Byzantine faulty. This scheme requires neither any modification to any database replica software nor does it require any additional software to run on any machine hosting a replica database.

2.5 Scaling Byzantine Fault-Tolerant Replication to Wide Area Networks [11]

The work extended the Byzantine fault tolerance to wide area networks. It uses hierarchical Byzantine fault-tolerant replication architecture, which confines the effects of any malicious replica to its local site, reduces message complexity of wide area communication, and allows read-only queries to be performed locally within a site for the price of additional hardware. Byzantine fault-tolerant protocol is employed within each site and a lightweight benign fault-tolerant protocol among wide area sites. Each site, consisting of several potentially malicious replicas, is converted into a single logical trusted participant. To order the operations locally, Byzantine agreement protocol is run within the site, and they agree upon the content of any message leaving the site for the global protocol. The protocol also eliminates the ability of malicious replicas to misrepresent decisions that take place in their site. Towards this goal, the messages between servers at various sites carry a threshold signature attesting that the sufficient number of servers at the originating site agreed with the content of the message.

3. SYSTEM MODEL

We assume a 2-tier architecture between the coordinator replicas and the participants. The protocol is started for a transaction when a commit/abort request is received from the initiator. To ensure safety and liveness properties, certain synchrony has been assumed among the replicas. As Byzantine faults are considered, only at the coordinator site, participants are not replicated. There are $3f + 1$ coordinator replicas, among which at most f can be faulty during a transaction. We assume a Transaction Manager TM, which itself is trusted and possesses the power to diagnose and replace the faulty coordinator as well as the faulty replicas. Each coordinator replica is assigned a unique id i, where i varies from 0 to $3f$. The *id* is required to identify the primary in a particular view and also for verification of the replica during message transmission. Fig 1 illustrates the schematic view change architecture where the replica labeled *P* is primary and replicas labeled *R* are backups. The rounded-corner rectangle represents the semantic view of Transaction Manager, TM.

The agreement for initial transaction request starts from view 0. After the first phase (prepare), the coordinator and replicas execute the agreement protocol. Subsequently they send their decision to coordinator replica and enter into the second phase (commit). Our view change mechanism is run by the Transaction Manager under both, reactive and proactive, approaches. Thus, agreement and view change protocols run in the interleaved manner.

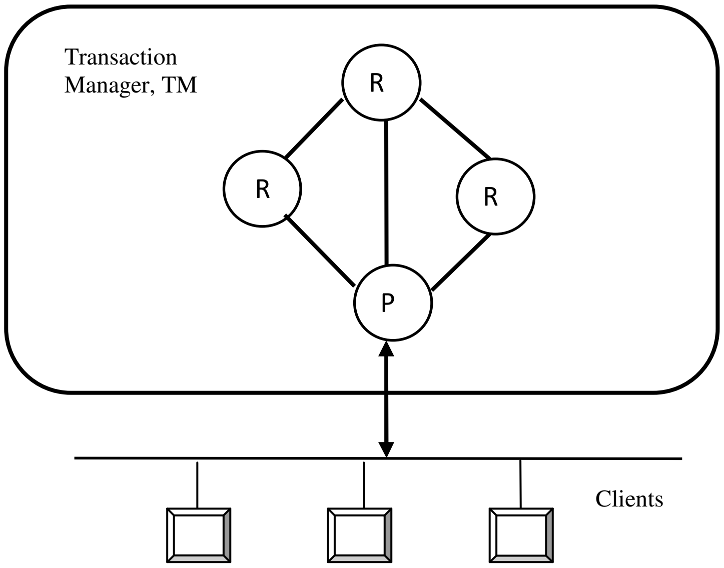

Fig.1. The schematic view change architecture

3.1 Problem Definition

Consider a protocol wherein a primary replica, say *P*, is activated for a particular transaction request by receiving a commit/abort message from among the population of clients. The message is propagated to rest 3*f* replicas for their proposed decision. Now, the agreement protocol among the replicas is run. The three basic properties that an agreement protocol must satisfy are *agreement, validity, and termination*.

Agreement: Any two non-faulty replicas that decide on a value (commit/abort) for a particular id, *i*, must decide on the same value. More specifically, a faulty replica, if any, is computationally infeasible to alter the decision of two non-faulty replicas.

Validity: If all non-faulty replicas have been activated on a given id, *i*, with the same initial value, then all non-faulty replicas that decide must decide on this value.

Termination: All non-faulty replicas eventually decide.

At the end of agreement protocol, all the replicas send their decision to the clients. This starts the commit phase. A client waits for *f*+1 matching messages before taking commit decision on transaction. After receiving the required number of matching replies, the client commits the transaction.

4. The Approach

This section elaborates, in detail, the reactive and proactive approach to view change mechanism.

4.1 The Proactive Approach

In this approach, in a particular view, one of the replicas is chosen as primary and other replicas work as backups. During the initial phase of registration, all replicas register themselves to the Transaction Manager, TM with their unique id's. Following this, the TM assigns the responsibility of the coordinator to the lowest id replica and designates it as primary, *P*. The current view message that contains the current view number *v*, primary replica *P* and transaction id *i*, is then broadcast to each participating replica. Finally, if 3*f* +1 replicas respond with an acknowledgement of current view, the TM sends a begin-transaction message to primary *P* in order to start the transaction processing.

The proactive approach depends mainly on two entities, namely *ping_time* and *status_flag*. The *ping_time* has been used to implement, essentially, a time out mechanism. It is an additional message that is attached only in the message field of the primary (coordinator) replica. Now, the coordinator replica is bound to declare its status to the TM within *ping_time*. It works as a failure detector in order to detect crash failure with the required level of accuracy. Although, for byzantine faulty replica, the failure detector has to know the semantic of the protocol as it may send some spurious messages to other replicas. However, the *ping_time* corresponds to failure detector that helps to ensure completeness in the protocol rather than accuracy. This would help to detect if primary *P* would be able to successfully participate in further transaction processing.

Pseudo code for proactive view change

```
At Replica site
Replica_id-Registration ( )
{
        Sends Registration Request in form of Join-Request to TM
        If Receives 2f+1 matching Join-Approved then
                Registration completes;
        Else
                Failure;
}
At Transaction manager site
Nearest replica-Search ( )
{
        Store every replica by unique Id;
        Select primary, P = lowest (Id);
        Sends Current view (v, P, i) to every replica;
        If 3f+1 matching View-ACK received then
                Send Begin-Agreement to P;
        Else
                Failure;
}
Proactive view-change ( )
{
        Declare ping_time and status_flag;
        For message = = Begin-Agreement to P,
                Set P(ping_time) = 1000ms
        For backups R,
                Set R(status_flag) = alive;
If P fails to report TM with defined ping_time,
        TM sets primary, P = faulty;
Else
Switch (Agreement);
}
If status (P) = faulty,
        Proactive view change occurs;
        Reinitiates registration phase ( );
        New primary, P = Id+1;
        Send new member-notification ( );
}
 New member-notification ( )
{
        Send New View to all replica nodes (v+1, P, i+1);
        If 3f+1 matching View-ACK received then
                Send Begin-Agreement to New primary, P;
}
At client side
Notification-Acknowledgement ( )
{
        Sends acknowledgement for notification;
}
```

Fig.2. Proactive View Change Approach

Each of the backup replicas has a special state variable *status_flag*, which represents the status of the replica. If the replica is non-faulty then it would set its *status_flag* as *alive*. However, any other value assumed by *status_flag* is considered to be don't care value. Initially, all replicas' *status_flag* value has been assumed to be *alive*. After the completion of each transaction round, the backup replicas inform their status to the TM. The transaction proceeds further, only, if the status of backup replica is *alive*; otherwise, it is suspected as a faulty. A call for new view change is initiated and the faulty replica is removed at an early stage. Otherwise, the faulty replica would have been detected after executing some rounds. The pseudo code for view change approach is shown in the Fig 2.

To this end, both of the above entities play key role in detecting failures, proactively, in a transaction processing system.

4.2 The Reactive Approach

In this approach, a faulty replica is treated only when it is identified and informed using time out mechanism. In a sense, this approach restricts the faulty replica to deviate from the specified behavior as any message, further, passed by faulty replica is not accepted by other nodes. The *reactive* approach works as follows.

Pseudo code for reactive view change

```
At replica site
Node_id-Registration ( )
{
        Sends Registration Request in form of Join-Request to TM
        If Receives 2f+1 matching Join-Approved then
                Registration completes;
        Else
                Failure;
}
At Transaction manager site
Nearest replica-Search ( )
{
        Browse & Store every Node by Id;
        Select primary = lowest (Id);
        Sends View- Message to every replica;
        If 3f+1 matching View-ACK received then
                Send Begin-Agreement to primary;
        Else
                Failure;
}
Reactive view-change ( )
{
        If (timeout)
                TM replace primary;
                New primary = Id + 1;
                Assign it T and proceed;
                Assign transaction log to new replica;
}
 New member-notification ( ){
        Send New View to all replica nodes;
        If 3f+1 matching View-ACK received then
                Send Begin-Agreement to New primary;
        }
At client side
Notification-Acknowledgement ( ){
        Sends acknowledgement for notification;
}
```

Fig.3. Reactive View Change Approach

Initially, all participating replicas register themselves to the Transaction Manager, TM with their id's. Afterwards, one of the replicas, the replica with lowest id, is selected to serve as the primary, say *P*. The current view is then broadcasted to everyone involved in the transaction processing.

The message format of current view contains the current view number *v*, primary *P*, transaction id *i*, and with a predefined timeout *T* to run the protocol. If TM receives the acknowledgement, in response to the current view message, from $3f+1$ replicas, it allows primary *P* to begin transaction processing. While running agreement, if timeout occurs for current view because of delay in message propagation or the primary is found faulty then view change occurs. If $f+1$ replicas inform to TM that the view change is due to fault in primary then the TM decides the new timeout to be *T* (i.e., timeout for previous view), $2T$ otherwise. Thus, it keeps same timeout for each view change in case of primary being suspected as faulty. Now, the transaction proceeds to reach an atomic decision commit or abort. For the understanding of reactive view change approach, the pseudo code is given in Fig 3.

Both of the view change approaches, *proactive* and *reactive*, are designed to minimize the discontinuity in the transaction processing. The comparative analysis of the performance of both mechanisms brings out the potential benefits of *proactive* over *reactive* in terms of latency, message overhead, and throughput.

5. Simulation Results

We have conducted simulation in order to evaluate the performance of *reactive* and *proactive* view change mechanism on the execution time (i.e., latency). Also, the agreement protocol is run and simulated to evaluate the performance of two-phase agreement over the standard three-phase agreement. We have used BFTSim [12]. It uses a back-end simulator which is based on ns-2 [13]. The front-end uses a declarative overlog language P2. Fig 4 shows latency-throughput curves for the protocols with *proactive* and *reactive* view change mechanism.

As we have used exponential distribution of faults, when the number of faults is less, both approaches deliver comparative performance in terms of latency. However, with the increase in number of faults, the latency gradient is significantly less in proactive approach.

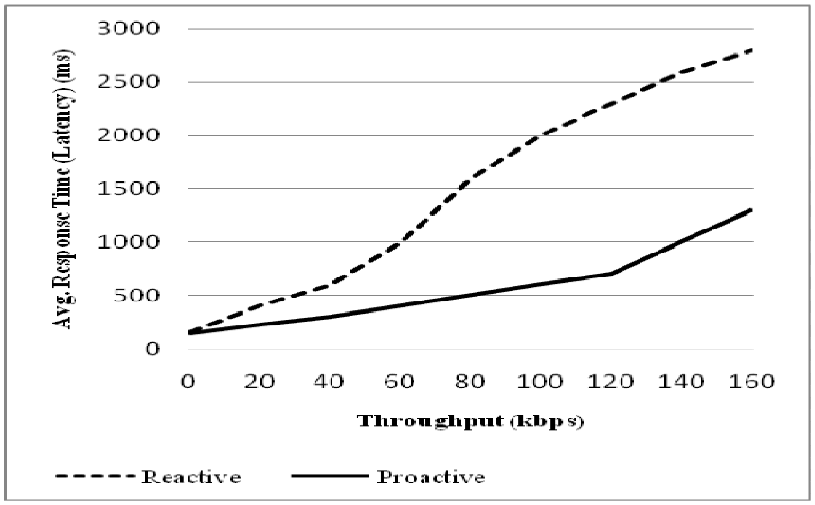

Fig.4. Latency vs. Throughput Curve for Reactive and Proactive Approach

6. CONCLUSION

The major contribution of this paper is the novel solution to view change mechanism. Our method uses a Transaction Manager, TM, to *proactively* detect the crash of the primary as well as backup replicas. To compare the performance, we also presented a *reactive* approach to view change mechanism. Both approaches have also been analyzed and experimentally evaluated. The *proactive* approach always exhibits the better performance in a faulty scenario that makes it suitable for long-lived applications. The proposed approach dramatically reduces the latency of the protocol and leads to enhanced throughput. In the end, the proposed agreement protocol reduces the overall message overhead as well as total execution time to a greater extent.

REFERENCES

[1] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, 2003.

[2] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*- Part I.

[3] K.B Chandradeep, "A Novel Approach for Protecting Exposed Intranets from Intrusions", *IJCNC*, vol. 2, no. 4, July 2010.

[4] Leslie Lamport et.al., "The Byzantine Generals Problem", *ACM Transaction on Programming Languages and Systems*, vol. 4, No. 3, pp. 382-401, July 1982

[5] M. Castro, B. Liskov, "Practical Byzantine Fault Tolerance and Proactive Recovery", *ACM Trans on Computer Systems*, vol. 20, no. 4, Nov, 2002.

[6] Wenbing Zhao, "A Byzantine Fault Tolerant Distributed Commit Protocol", *IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pp. 37-46, Sep, 2007.

[7] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, E. Wong, "Zyzzyva: Speculative Byzantine Fault Tolerance", *ACM Proceedings of twenty-first Symposium on Operating Systems and Principles*, vol. 41, no. 6, pp. 45-48, Oct, 2007.

[8] A.Silberschatz, Henry F. Korth, E. Levy, "An optimistic commit protocol for distributed transaction management", *ACM SIG Management of Data*, vol. 20, no. 2, June, 1991.

[9] J. Cowling, D. Myers, B. Liskov, R. Rodrigues and L. Shrira, "HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance", *Proceedings of the Seventh Symposium on Operating Systems Design and Implementation (OSDI '06)*, pp. 177-190, Seattle, WA, USA. November 2006.

[10] Ben Vandiver, Hari Balakrishnan, Barbara Liskov, "Tolerating byzantine faults in transaction processing systems using commit barrier scheduling", *Proceedings of twenty-first ACM SIGOPS symposium on Operating Systems Principles*, pp 59-72, 2007.

[11] Yair Amir et.al., " Scaling byzantine fault-tolerant replication to wide area networks", Technical Report CSD TR #05-029, December 2005.

[12] BFTSim, http://bftsim.mpi-sws.org/, April 2008

[13] The NS-2 Project. http://www.isi.edu/nsnam/ns, Oct. 2007.

Authors

Poonam Saini received B.Tech degree in Information Technology from Kurukshetra University, Kurukshetra, India in 2003. She received M.Tech in the area of Software Engineering from the same university in 2006. Currently, she is pursuing PhD in the area of Fault Tolerant Distributed Computing from National Institute of Technology, Kurukshetra, India. Her research interests include fault tolerant distributed computing and byzantine consensus.

Awadhesh Kumar Singh received B. E. degree in Computer Science & Engineering from Gorakhpur University, Gorakhpur, India in 1988. He received M.E. and PhD (Engg) in the same area from Jadavpur University, Kolkata, India. Currently, he is Associate Professor in the Department of Computer Engineering, National Institute of Technology, Kurukshetra, India. His present research interest is mobile distributed computing systems.