Journal of Computer Science 7 (1): 101-107, 2011
ISSN 1549-3636

Proactive and Reactive View Change for Fault Tolerant Byzantine Agreement

Poonam Saini and Awadhesh Kumar Singh
Department of Computer Engineering, National Institute of Technology,
Kurukshetra, 136119, India

Abstract: Problem statement: Dealing with arbitrary failures effectively, while reaching agreement, remains a major operational challenge in distributed transactions. In the contemporary literature, standard protocols such as Byzantine Fault Tolerant Distributed Commit and Practical Byzantine Fault Tolerance handles the problem to a greater extent. However, the limitation with these protocols is that they incur increased message overhead as well as large latency. **Approach:** To improve the failure resiliency with minimum execution overhead, we propose two new protocols based on *proactive* view change and *reactive* view change. Also, both approaches have been analyzed and compared. **Results:** Our dynamic analysis reflects that, in a faulty scenario, the *proactive* approach is computationally more efficient with reduced latency as compared to *reactive* one. **Conclusion/Recommendations:** Moreover, unlike PBFT and BFTDC, our agreement protocol runs in two phases, which leads to reduced message overhead and total execution time.

Key words: Distributed transactions, Two-phase commit, Byzantine agreement, Proactive view change, Reactive view change, dynamic analysis

INTRODUCTION

A distributed system is a collection of independent computers, which are capable of collaborating on a task (Swaroop and Singh, 2007). Distributed Computing Systems (DCS) provide an efficient way to achieve fault-tolerance and share system resources such as processing elements, memory modules, data files, and so on. A successful execution of a distributed program usually requires one or more of the resources that reside on multiple hosts at different geographic sites of the DCS (Xing and Shrestha, 2010). It is possible that some faults of hosts or communication links may not be adequately detected. This can be categorized under Byzantine i.e., arbitrary faults.

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 (Silberschatz *et al*., 1991), which study correctly in presence of benign faults only. The present study aims at designing an efficient agreement algorithm that successfully handles Byzantine faults (El Emary and Al Rabia, 2005). In addition, it significantly reduces the time taken to complete the transaction. Two protocols, one based on proactive and the other on reactive approach, have been presented and analyzed. Both the protocols achieve increased system availability and enhanced throughput.

Traditional Byzantine fault tolerant protocols such as BFTDC (Zhao, 2007), PBFT (Castro and Liskov, 2002) and Zyzzyva (Kotla *et al*., 2007) 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 study 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 (Saini and Singh, 2010), 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. Failures may be caused by power exhaustion as well as due to malicious security attacks (Sundararajan and Shanmugam, 2010) like message replication, passing wrong information, fluctuating status of replicas etc. The proactive protocols maintain the state information in advance while, on contrary, the reactive one reduces the impact of frequent failures whenever the fault in the system is notified by the active participating replicas in the system (Vijayaragvan *et al*., 2009).

Towards this goal, we build a system model to analyze the failure resiliency of our protocol under both reactive and proactive approaches.

Motivation: In most of the contemporary study, the protocol replaces the replica from the system when it

Corresponding Author: Poonam Saini, Research Scholar, Department of Computer Engg, National Institute of Technology, Kurukshetra-136119. Haryana, INDIA. nit.sainipoonam@gmail.com

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.

Contribution of paper: 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.

Agreement-based protocols, such as BFTDC (Zhao, 2007), run a three phase agreement protocol among replicas before it executes a request. However, in our proposed system model, the agreement process finishes in two phases, namely *ready-to-commit* and *set-commit*. Thus, the protocol has reduced message overhead and it broadcasts the right decision to all clients. The protocol has been shown to be computationally more efficient.

System model: We assume client/server 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

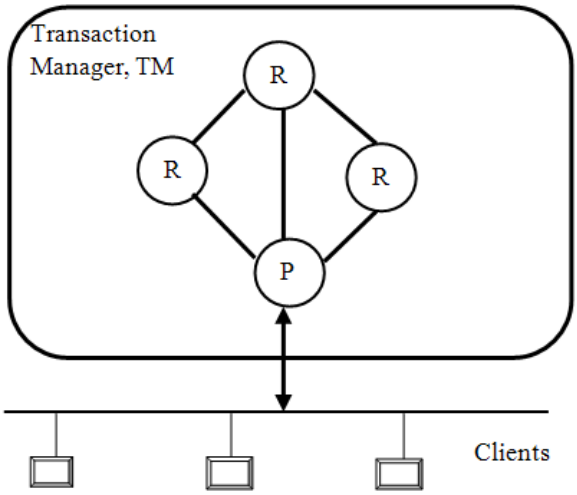

Fig. 1: The schematic view change architecture

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.

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 $3f$ replicas for their proposed decision. Now, the agreement protocol among the replicas is run. Three basic properties that an agreement protocol must satisfy are:

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.

MATERIALS AND METHODS

The proactive approach: In this approach, in a particular view, one of the replicas is chosen as primary and other replicas study 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.

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 following Fig. 2.

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

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

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. 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 reactive view change, the pseudo code is given in the 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.

The optimized agreement: The system handles one transaction request at a time. The initiator is responsible for initializing a transaction. During the prepare phase, the primary sends a prepare request to every participant in the transaction. The prepare request is piggybacked with a prepare certificate, which contains the commit request sent by the initiator.

After the prepare phase, the replicas engage in the agreement round for the transaction. The agreement protocol works in two phases namely, *ready-to-vote* and *set-commit*. During the first phase, the primary *p* sends an awake-to-vote message and its decision to all other replicas. The message has the format $<awake\text{-}to\text{-}vote, v, t, o, C, At_i>$, where *v* is the current view number, *t* is the transaction id, *o* is the proposed transaction outcome (*i.e.*, commit or abort), *C* is the decision certificate and At_i is the predefined

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

upper bound in order to reach agreement. There is one decision certificate corresponding to each participant. The transaction id is included in each registration and vote record so that the final outcome given by correct participants is atomic. A backup replica suspects, the primary *P* to be faulty, on following basis: if awake-to-vote message does not fall into the same view number *v*, and the same transaction id *t* that has not been executed earlier. If the primary replica fails the above verification then the backup replica initiates a view change

immediately; otherwise, it accepts the awake-to-vote message. It then logs the accepted message and multicasts act-commit message with the same decision *o*, which is in the awake-to-vote message. This initiates the *set-commit* phase. The act-commit message has the format <*act-commit, v, t, o, D*>, where *D* is the digest of the decision certificate *C* and other contents are same as awake-to-vote message. A coordinator replica i.e., primary *P* accepts act-commit message provided the backup replica is in the current view *v* and the current transaction is *t*. Also, the digest value on both sides must be same. The following Fig. 4 represents the agreement algorithm.

If a replica has collected 2*f*+1 matching awake-to-vote messages from different replicas (including the replica's own, if it is backup), it executes act-commit on transaction *t*. This marks the end of *set-commit* phase and also the agreement protocol. During agreement, if any conflict occurs and a non-faulty replica *i* could not advance to *set-commit* state till the timeout, it broadcast a view change message to the TM as well as to all other replicas. If the primary is found faulty, TM assigns to next lowest id the responsibility of the coordinator replica i.e., primary, *P*. Otherwise, the transaction is reinitiated from the agreement protocol with the previously selected primary replica.

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

The idea of agreement phase removal: In literature, PBFT (Castro and Liskov, 2002) is the first protocol that employs a three-phase agreement between the replicas. The protocol is designed for total ordering of multiple transactions and runs in three-phase to reach agreement. Our protocol handles one transaction at a time and runs agreement to decide on commit or abort for a particular transaction. The three phases in BFTDC are *ba-pre-prepare*, *ba-prepare* and *ba-commit*. In *ba-pre-prepare* phase, the secondary replicas check for the validity of the primary replica by comparing the messages they received from the clients and messages received from the primary. If the numbers of messages fail to match, the replicas suspect the primary to be faulty and request for a view change in order to select new primary. However, in our protocol the faulty primary,

Pseudo code for Agreement

At Transaction manager site ( )
If status(*R*) = *alive*;
Send *Begin-agreement* to *P*;

At Primary replica site ( )
If (Message = = *Begin-agreement*),
Begin ready-to-vote phase;
Send message = (*awake-to-vote*, *v*, *t*, *o*, *C*, *At*$_i$);

Set ready-to-vote-complete = true;
else
Make entry in *awake-to-vote-fail*;

At Backup replica site ( )
If (Message = = *act-commit*, *v*, *t*, *o*, *D*),
If (verify (message) = = true) &&
digest(*awake-to-vote*) = = digest(*act-commit*)
act-commit-count++;
If 2f+1 *act-commit* are received then send decision to all
If (act-commit-count >= 2f+1 && ready-to-vote-complete = = true),
Set *act-commit-complete* = true
Send result to all participants
else
Make entry in *act-commit-fail*;
If (timeout)
If (*v* is in act-commit-fail),
Send *view_change* to TM;
Wait for new view;

Fig. 4: The Optimized Agreement Protocol

if exists, is treated by the TM, in advance, before it gets involved in further transaction processing. It would result in the increased efficiency of the system in terms of time overhead.

Secondly, in both *ba-prepare* and *ba-commit* phase, messages are exchanged. The number of messages being compared at each replica is different in both phases. In *ba-prepare* phase this number is 2*f* while in *ba-commit* phase it is 2*f* + 1. By running the protocol for different number of faulty replicas, it is found that the *ba-commit* phase (with 2*f* +1 comparisons) also fulfills the minimum matching criteria for *ba-prepare* phase (with 2*f* comparisons). Therefore, one of the phases from the agreement protocol can be avoided or merged with any of the other phase while maintaining the minimum conditions to reach an atomic decision.

Our protocol reduces the three-phase agreement in two-phases only while achieving the necessary and minimum requirements to complete the protocol execution. Thus, the message overhead of agreement protocol reduces drastically. This also results in less execution time overhead of the protocol. These analytical inferences have also been substantiated by the experimental results.

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.

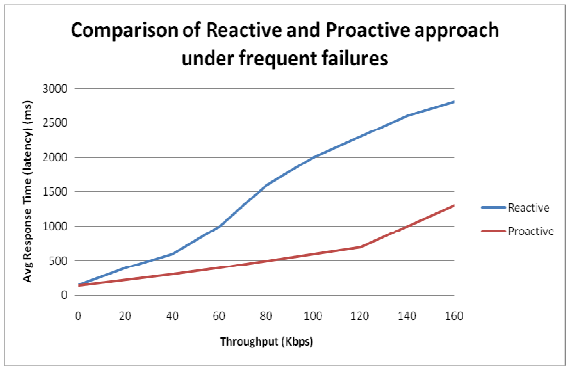

Fig. 5: Latency-Throughput Curve

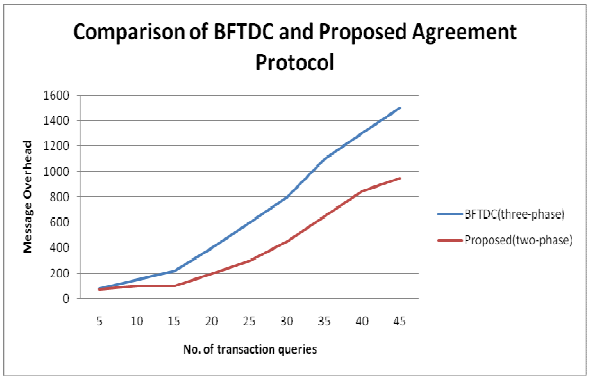

Fig. 6: Message overhead

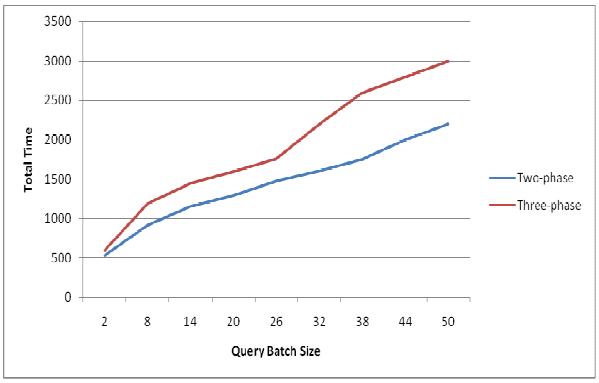

Fig. 7: Time overhead

We have used BFTSim (2008). It uses a back-end simulator which is based on ns-2. The front-end uses a declarative overlog language P2. The experiment is carried out for queries with different batch size in order to view the differences in message overhead and time overhead involved in the transaction query processing.

The protocols are implemented based on the pseudo code description. Figure 5 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.

In the next experiment, protocol has been simulated to compare the message overhead of our protocol with BFTDC (Silberschatz *et al*., 1991). The output plot is shown in Fig. 6.

For small-sized batch of transaction queries, the message overhead is nearly same. However, as the batch size of transaction query increases, the message overhead increases with faster rate in case of BFTDC (Silberschatz *et al*., 1991). This phenomenon leads to reduction in total time consumed to complete the transactions. This is evident in the plot shown above in Fig. 7.

DISCUSSION

The study evacuates the overhead involved in terms of latency as well as message overhead to a greater extent. Moreover, the optimized reactive view change with the proposed novel idea of proactive view change helped in designing a more failure-resilient protocol.

CONCLUSION

The major contribution of this study 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. The study also presents an optimized Byzantine agreement protocol which is able to reach agreement in two-phases in comparison to widely used three-phase like in BFTDC protocol. In the end, the proposed agreement protocol reduces the overall message overhead as well as total execution time to a greater extent.

REFERENCES

BFTSim, 2008. BFTSim: A simulation framework for comparing BFT protocols. http://bftsim.mpi-sws.org/, April 2008

Castro, M. and B. Liskov, 2002. Practical byzantine fault tolerance and proactive recovery. ACM Trans Comput. Sys., 20: 398-461. DOI: 10.1145/571637.571640

El Emary, I.M.M. and A.I. Al Rabia, 2005. Fault detection of computer communication networks using an expert system. Am. J. Applied Sci., 2: 1407-1411. DOI: 10.3844/.2005.1407.1411

Kotla, R., L. Alvisi, M. Dahlin, A. Clement and E. Wong, 2007. Zyzzyva: Speculative byzantine fault tolerance. Proceeding of the 21st ACM Symposium on Operating Systems Principles, Oct 14-17, 2007, Stevenson, WA., pp: 45-48.

Saini, P. and A.K. Singh, 2010. An efficient byzantine fault tolerant agreement. Proceeding of the International Conference on Methods and Models in Science and Technology, Nov. 6, National Institute of Technology, India, pp: 162-165. DOI: 10.1063/1.3526183

Silberschatz, A., H.F. Korth and E. Levy, 1991. An optimistic commit protocol for distributed transaction management. Proceeding of the 1991 ACM SIGMOD International Conference on Management of Data, May 23-29, ACM New York, NY, USA., pp: 88-97. ISBN: 0-89791-425-2

Sundararajan, T.V.P. and A. Shanmugam, 2010. Novel intrusion detection system for wireless body area network. J. Comput. Sci., 6: 1355-1361. ISSN: 1549-3636

Swaroop, A. and A.K. Singh, 2007. A token-based fair algorithm for group mutual exclusion in distributed systems. J. Comput. Sci., 3: 829-835. ISSN: 1549-3636

Vijayaragavan, S., K. Duraiswamy, B. Kalaavathi and S. Madhavi, 2009. A performance study of reactive multicast routing protocols in virtual class room using mobile ad hoc network. J. Comput. Sci., 5: 788-793. ISSN: 1549-3636

Xing, L. and A. Shrestha, 2010. Reliabilty evaluation of distributed computer systems subject to imperfect coverage and dependent common-cause failures. J. Comput. Sci., 2: 473-479. DOI: 10.3844/jcssp.2006.473.479

Zhao, W., 2007. A byzantine fault tolerant distributed commit protocol. Proceedings of 3rd IEEE International Symposium on Dependable, Autonomic and Secure Computing, Sep 25-27, Cleveland State University Cleveland, Columbia, pp: 37-46. ISBN: 978-0-7695-2985-1