International Journal of Applied Operational Research
Vol. 3, No. 1, pp. 95-108, Winter 2013

Journal homepage: www.ijorlu.ir

Optimization of majority protocol for controlling transactions concurrency in distributed databases by multi-agent systems

S. M. Almasi Mousavi*, H. R. Naji, R. Ebrahimi Atani

Received: 30 July 2012 ; **Accepted:** 29 October 2012

Abstract In this paper, we propose a new concurrency control algorithm based on multi-agent systems which is an extension of majority protocol. Then, we suggest a clustering approach to get better results in reliability, decreasing message passing and algorithm's runtime. Here, we consider *n* different transactions working on non-conflict data items. Considering execution efficiency of some different transactions simultaneously is one of our algorithm features. For evaluating performance of the proposed algorithm we applied it using the MPICH2 mechanism.

Keywords Distributed Database, Concurrency Control, Majority Protocol, Multi-Agent, MPICH2.

1 Introduction

During recent years "distribution" has been evaluated as a critical and important issue for databases. This issue has many logical reasons such as natural distribution of organizations. A distributed database is a set of several parts that correlate with each other logically over a network of interconnected computers. Collections of data (e.g. in a database) can be distributed across multiple physical locations. Since the database is distributed, different users can access it without interfering with each other. However, the *DBMS* must periodically synchronize the scattered databases to make sure that they all have consistent data.

The database system through a scheduler must monitor, examine, and control the concurrent accesses so that the overall correctness of the database is maintained [1]. There are two criteria for defining the correctness of a database: database integrity and serializability [2]. Database integrity ensures that data entered into a database is accurate, valid, and consistent. The serializability ensures that database transitions from one state to the other are based on a serial execution of all transactions [1].

Concurrency Control is applied to solve these criteria. Concurrency control problems and solutions have been formalized in [3] and implemented and used in a variety of real world applications [4]. Generally algorithms and protocols of concurrency control resident in two

* **Corresponding Author.** (✉)
E-mail: mehrzad.almasi@gmail.com (S. M. Almasi Mousavi)

S. M. Almasi Mousavi
Department of Computer, Science and Research Branch, Islamic Azad University, Kerman, Iran

H. R. Naji
College of Electrical and Computer Engineering, Kerman Graduate University of Technology, Kerman, Iran

R. Ebrahimi Atani
Department of Computer Engineering, University of Guilan, Rasht, Iran

schedule categories: Conservative and Aggressive. In the first one, a delay can be occurring between transactions if necessary but in the other one, transactions should run immediately.

The most important concurrency control protocols can be categorized as follows: Graph based, Timestamp based and Lock based Protocols, all considered in this study. The traditional approach to concurrency control is based on locking [5]. In this method that is based on the allocation of data to transaction, when a transaction wants to access data for writing or reading it should send a corresponding lock request to a section called lock manager. Concurrency control mechanisms in distributed databases can be single-copy (without replication) or multiple-copies. In the second case concurrency control is more difficult than the first one as in distributed databases with replication, mutual consistency should meet (i.e. all copies of a data should always have the same value).

In this paper we have proposed a new concurrency control algorithm based on multi-agent systems which is an extension of majority protocol. We have used the message passing mechanism between nodes to let them communicate with each other and finally, a vote method is used to determine which transaction can be executed and which cannot. Using a multi-agent approach we can parallelize activities between nodes; therefore activities are shared almost equally between nodes, and thus workload imposed to the central node is eliminated. We have also proposed a clustering model for decreasing the number of messages and algorithm's runtime and have prepared schemes for increasing reliability in this model.

The rest of the paper is organized as follows: in section 2, we discuss an overview of applied concepts. In section 3, we present related works on concurrency control algorithm. Our new algorithm with an illustrative example and clustering model are given in sections 4 and 5 respectively. Implementation and results are discussed in section 6 and finally conclusion is presented in section 7.

2 Background

2.1 Multi Agent systems

An agent is a component situated in an environment and capable of autonomous action in this environment in order to meet its design objectives [6]. According to Wooldridge, an agent has to fulfill some properties in order to become intelligent. These properties are: reactivity (the ability to perceive its environment and respond to changes in a timely fashion), pro-activeness (the ability to exhibit goal-directed behavior by taking the initiative), and social ability (the ability to interact with other agents) [6]. Intelligent agents (IA) can present some other properties such as temporal continuity (an agent operates continuously and unceasingly), reasoning (decision-making mechanism, by which an agent decides to act on the basis of the information it receives, and in accordance with its own objectives to achieve its goals), rationality (an agent's mental property that attract it to maximize its achievement and to try to achieve its goals successfully), veracity (mental property that prevents an agent from knowingly communicating false information), mobility (i.e. the ability for a software agent to migrate from one machine to another) [7].

A multi-agent system consists of a group of agents that can potentially interact with each other [8]. By exploiting this feature several advantages such as reliability and robustness, modularity and scalability, adaptively, concurrency and parallelism, and dynamism [9] can be reached.

2.2 Majority protocol

Majority protocol is one of the methods in distributed lock management for concurrency control. In this protocol, local lock manager is responsible for locking and unlocking of data. Majority protocol has the same behavior with both shared locks and exclusive locks. When a transaction wants to lock data Q in site S_j that has no copies (i.e. without replication), it sends a message to the manager of the site S_j. If the corresponding data is locked inconsistently prior to the presented request, the next request will be suspended until the compliance time. When the request is confirmed, the lock manager sends a message to the request sender to inform it of the acceptance of the request. But if data Q is replicated on n sites, the request will be sent to more than half of them. A transaction can use data Q only if most of sites consisting of data will lock the copy of data. Majority protocol has some advantages and disadvantages. If a site failure occurs and some data are inaccessible, this method still can continue its work which is one of the advantages of the majority protocol but too much message passing and deadlock occurrence are its disadvantages.

2.3 Primary copy protocol

In this protocol, one of the replications of each data is recognized as main version and the site including main version is called main site. It is clear that different sites may have different main sites. Primary copy protocol has the same behavior with both shared locks and exclusive locks. A transaction sends a request to main site Q when it wants to lock corresponding data Q. In this protocol sending request to all sites is not necessary. The important advantage of this method is that it has the same concurrency control for both replicated data and data without replication. Implementation of this protocol is also simple, but primary copy protocol has yet some disadvantage. If the main site Q fails, data Q cannot be accessible even if other sites including replication Q are accessible.

3 Related works

Much work has been done in studying characteristics of centralized schedulers. An interesting model and simulation results can be found in [10] by Agrawal, Carey and Livny. The work that has been done is mostly theoretical, but some interesting simulation models have been developed and simulated at the University of Wisconsin. In [11] Carey and Livny described a distributed DBMS model, an extension to their centralized model. Different simulation parameters are examined through simulations. Several papers about concurrency control have also been written by Thomasian *et al.,* [12, 13].

Breitbart and *et al.*, [14] proposed an optimistic approach to transaction management for replicated database. They proposed a new transaction management protocol that guarantees global serializability and freedom from distributed deadlocks without relying on any properties of the DBMSs running at the local sites. In comparison to prior protocols, this protocol reduces the communication required to coordinate transaction by a factor of r, where r is the average number of operations per transaction. They also considered implementation issues in reducing message overhead and discussed failure recovery [14].

There are many other approaches for improving performance of concurrency control that meet characteristics of some different approaches simultaneously but combination of such approaches isn't necessary. For example, Tramura and *et al.,* [15] proposed a generation system of concurrency control program by using genetic programming (GP). This system generates concurrency control program according to the features of transactions, which are collections of database operations. Functions and terminals of trees representing program in GP, and the fitness measure function used in GP are proposed in that article. The functions and the terminals include those changing and testing variables attached to data items and transactions as well as those checking the kind of operation. These will bring a general concurrency control program, which is beyond the combination of the parts of traditional concurrency control program. As the granularity of functions and terminals are small. The sub-trees, which are used for the popular concurrency control protocol, prepared in advance, are used. The fitness measure function considers the goodness of concurrency control program. The experiments show that a concurrency control program using locks could be generated under the concurrent environment, while a concurrency control program better than the two-phase locking protocol could be generated under the not-so-concurrent environment [15].

Concurrency control algorithms also are used in real time databases. For example, Chen and *et al.,* [16] proposed a new concurrency control protocol for real-time database (RTCC). The protocol is based on the traditional speculative concurrency control protocol (SCC). It dynamically establishes the maximum of shadow to reasonably use the resources of system and add the quasi-commit phase to avoid many unnecessary restarting and enhance the concurrency of transaction. The theory of Petri net proof and results of experiment show that this protocol is feasible and effective, and it can meet the needs of real-time transaction [16].

One of the other environments in which concurrency control can be used is mobile computing environments, since clients can access shared data and update them regardless of their physical location. So, data inconsistency may occur in such environments for which several concurrency control techniques are proposed. For examples, Niazmuddin and *et al.,* [17] analyzed the existing scheme of concurrency control without locking and justify its Performance limitations. A new priority based scheme is proposed in which priority is given to the older transaction whenever conflict arises which decreases the current load of database server. Experimental results show performance benefits and increases in commit rate of the transactions [17].

4 The Proposed Algorithm

In this section we propose a new concurrency control algorithm based on collaboration between some nodes that are distributed into a network geographically. Our algorithm is based on majority protocol but has some additional features. We use message passing mechanism for exchanging messages between nodes like majority protocol, but here all nodes collaborate with each other for concurrency control. This means that nodes exchange messages to each other in contrast to the majority protocol in which sites have peer to peer communications with initiative nodes. One of the important characteristic of our algorithm is that some different transactions can be considered in a single iteration of the algorithm simultaneously. This means that concurrency control progress can be applied to some different transactions at the same time to determine which transaction can be run. The node that initiates the algorithm is called *monitoring node*.

In this method we have used multi-agent system concept, which each of the nodes in distributed database environment is an agent that interacts with others. So, massage passing between agents and counting votes for each transaction can be done in parallel. Parallelism in counting votes and message passing gives us better results on timing and reduction of workload on monitoring agent.

This algorithm can be used to control concurrency in a set of agents, so that each pair of agents in the set is within the radio range of each other as it is shown in Fig. 1 [9]. Here, in our proposed algorithm no node sends valuable information during message passing. That is, we send metadata instead of valuable information for message passing mechanism to reduce bandwidth load which is one of our algorithm's features. Dashed circles between nodes in fig 1 represent two nodes are within the radio range of each other. An example of a set of nodes in such network is shown in Fig. 2.

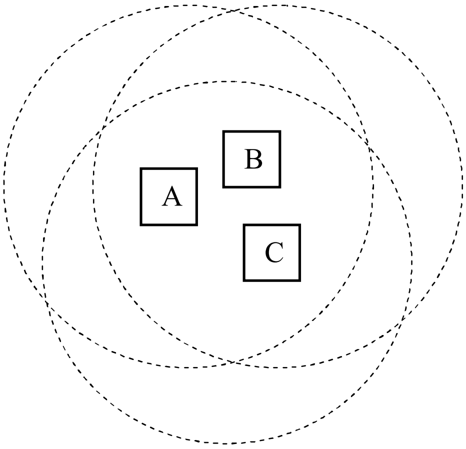


Fig. 1 Three nodes within radio range of each other [18]

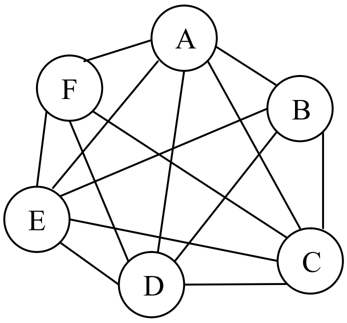


Fig. 2 A set of nodes in network: an edge between two nodes denotes they are within the radio range of each other

To present the algorithm we make the following assumptions: Assume *n* as the number of transactions in each iteration where *n* is the number of agents in the network. Each data Q is replicated on all local databases. Without loss of generality, we also assume that the initiative node of this algorithm (i.e. the monitoring node) is an honest node and participates in algorithm as an ordinary node. In each iteration of algorithm, *n* requests for locking *n* different data can be considered. Each transaction has a unique ID like [0, … , n-1] while nodes can be found by [0, …, n-1] IDs.

Steps of our algorithm are described as follows:

Step 1: The monitoring node, *M* broadcasts a message (Metadata) to the other *n-1* nodes asking them to check status of data Q_1, … , Q_n. Nodes should check for acceptance of the requested data Q_i. With this message the algorithm is initiated.

Step 2: Upon receiving the message (Metadata), node *i* and also the monitoring node check whether transaction *j* can lock its requested data or not. If node *i* locks data copy Q_j by its local lock manager, then it will send a message for transaction j to another agent where destination agent ID is *j*. If i=j then this massage sending will be implicit.

Step 3: In this step, each node does this: For each *i* and each *j* ($j \neq i$), let M_j be the message node *j* has received from node *i* in step 2. So, this message is a vote for transaction *j*. Therefore, agent *j* counts received votes for transaction j. All agents do this in parallel. If number of votes for each transaction is at least n/2+1, transaction *j* can be run.

Step 4: In this step, each agent informs result of counting votes and sends a commit message to other nodes for running corresponding transaction.

Step 1 is used to send the message by the monitoring node for initiating algorithm. In step 2, depending on status of data Q (whether can be locked or not) each agent sends a message to the other agents where destination nodes are selected by a rule as follows. Node *i* sends a message to *j* if it can lock the requested data of transaction *j*. This is done for all transactions on each agent. In step 3, each agent counts the number of its received messages during step 2. These counts can be done in parallel. In fact, there is no boundary between steps 2 and 3, i.e. upon receiving message by agents, counting of votes is started, however, it is possible that some agents have received no message. We presented our algorithm in several steps for better understanding of the reviewers. In the final step, agents will send a commit message for executing transaction if at least n/2+1 nodes send their votes for locking the requested data of transaction during previous steps.

3.1. An Illustrative Example

To understand how this algorithm works, we consider a sample case when there are 6 nodes in distributed database environment. For simplicity of the example we consider our new algorithm in which all *n* transactions can lock their requested data but due to failure of some links all messages cannot reach destination. Table 1 shows which links have failed and which haven't. Fig. 3 illustrates the messages passed between nodes during the first two steps of our algorithm where node 0 is the monitoring node. In step 1, node 0 sends out a message (represented by solid lines), to the other five nodes (1, 2, 3, 4 and 5). Message passing during step 2 can be done as mentioned above (represented by dashed lines).

Table 1 showing node i can lock which data copy Q_j

Request Data Q_j / Node i	0	1	2	3	4	5
0	Ok	Ok	failure	Ok	failure	Ok
1	Ok	Ok	failure	failure	failure	failure
2	failure	failure	Ok	Ok	failure	failure
3	failure	failure	failure	Ok	Ok	Ok
4	Ok	Ok	failure	failure	Ok	failure
5	Ok	Ok	Ok	Ok	Ok	Ok

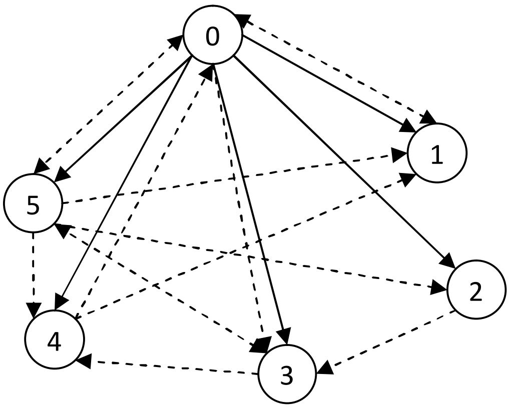


Fig. 3 Message passing during step 1 and 2

Fig. 4 illustrates the number of received messages by the *n* nodes during steps 1 and 2 of the algorithm. Messages passed between nodes illustrated by number 1 and number 0 show there is no message exchange between two corresponding nodes in step 2.

A row denotes the received messages by a node. For example, the first row of the matrix denotes that node 0 has received messages (Metadata) from nodes 0, 1, 4 and 5. Similarly, the third row denotes that node 2 has received messages (Metadata) from nodes 2 and 5 and so on.

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \end{array}\begin{array}{c} \text{number of votes for transaction} \\ 4 \\ 4 \\ 2 \\ 4 \\ 3 \\ 3 \end{array}$$

Fig. 4 Matrix showing messages passed according to Fig. 3

After counting votes by each agent, it is clear that nodes 0 to 5 have 4, 4, 2, 4, 3 and 3 votes respectively. As mentioned above each request which is gaining at least n/2+1 (here 4) votes can be run.

5 The clustering model

The purpose of clustering is to decrease number of exchanged messages between nodes for reducing network traffic and overload on monitoring node and other nodes. Here, each cluster is called as an external agent for which concurrency control algorithm mentioned in previous section can be run separately. Nodes in clusters are called internal agents and the node initiating the algorithm is called monitoring agent. Use of external agents can lead increasing the speed of algorithm due to parallelism and concurrency control level. However clustering nodes can sometimes lead to reduction of efficiency and this occurs when the number of failures is too much.

To present the clustering model we make the following assumptions:

Number of nodes in system should be an odd number. This algorithm has the most efficiency when it has the least failures in the links. So, when the number of link failures is too much, this algorithm doesn't have the best performance. Increase of link failures increases the number of message passing and hence, the execution time.

Here, we consider an example with 11 nodes where node 0 is the monitoring node. Monitoring agent initiates the algorithm and does not stand in any cluster. Ten other nodes stand in two clusters with 5 nodes in each one. First, monitoring agent broadcasts a message (metadata) to 10 other nodes. After receiving the message from the monitoring agent by inner agents, the algorithm mentioned in previous section runs in each cluster in parallel. Note that only transactions can be considered in each cluster that theirs corresponding nodes exist. For example in the first cluster (cluster 0) with nodes 1 to 5 only transactions 1 to 5 can be considered and transactions 6 to 10 are considered in the other cluster. In this stage, votes belonging to each transaction are counted by their corresponding node. Upon receiving at least n/2+1 votes, each inner agent broadcasts a commit message to all other nodes in clusters and also the monitoring node to let them lock required data of transaction (the transaction which possesses the same ID as the message sender) so that transaction can be done perfectly. There is a threshold time for sending commit message by inner agents. If monitoring agent doesn't receive commit message from inner agents until reaching threshold time, it means votes for corresponding transaction aren't enough. Therefore the algorithm goes to the next stage and transaction *i* which hasn't received enough votes will be considered in the other cluster. The monitoring node determines which transactions should be considered in this stage by sending a message to other clusters. Here, each inner agent in cluster k_1 sends its votes to corresponding node (j) of node i (i is in cluster $k_{2)}$ where j is in cluster k_1 and derived as $j=i+(|k_1-k_2|*m)$ where *m* is number of node in each cluster. This is true if i<j otherwise this formula changes as $j=i-(|k_1-k_2|*m)$.

So, corresponding node *j* is responsible for receiving votes for transaction i. Upon receiving at least n/2+1 votes by agent j, it broadcasts a message to all others nodes in all clusters and the monitoring node as well.

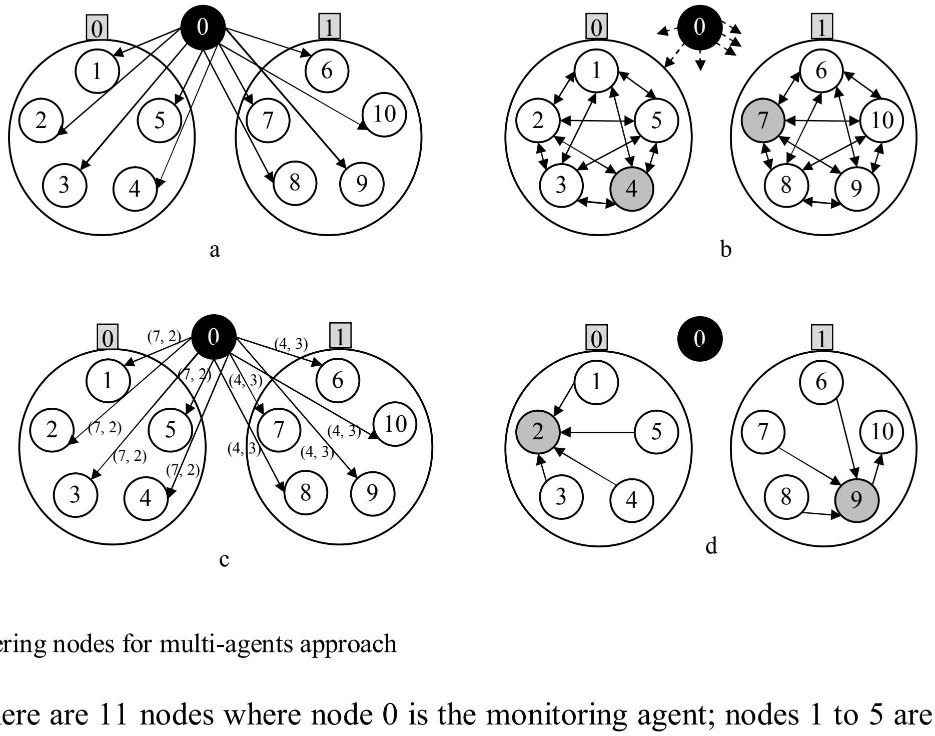


Fig. 5 Clustering nodes for multi-agents approach

In Fig 5 there are 11 nodes where node 0 is the monitoring agent; nodes 1 to 5 are in the first cluster (cluster 0) and others are in cluster 1. First, monitoring agent broadcasts a message to all other nodes for initiating the algorithm and parallel execution of algorithm in both clusters (Fig 5a). Then, transactions 1 to 5 are considered in cluster 0 and transactions 6 to 10 are considered in cluster 1. Here, monitoring node can receive votes for transaction 0 from n-1 other nodes as one of inner agents. As described in previous section each node which can lock required data of transaction i sends a message to node i (Fig 3b). Upon receiving n/2+1 messages from nodes (including monitoring node, shown with dashed arrow) each inner agent broadcasts a commit message to other n-1 nodes which means that corresponding transaction can be run. If monitoring agent doesn’t receive commit message from some inner agents (here, node 4 and 7, shown by colored nodes) until reaching a threshold time, monitoring node broadcasts a message as (7, 2) to cluster 0 for transaction 7 in which the first argument is ID of the node and the second is number of votes received previously for transaction 7 (fig 3c). This is also true for transaction 4 in cluster 1. Here the second argument is for each node in other cluster to be informed of the number of received votes of corresponding transaction in previous step. Therefore, the corresponding node announces the beginning of transaction by sending a commit message as soon as it has received the rest of votes and reached the minimum number required.

In the next stage, nodes 1 to 5 in cluster 0 send their votes for transaction 7 to node 2 because according to the aforesaid formulas, when i>j, corresponding node (j) of node 7 (i) equals to 2; (j=i-(|k1-k2|*m)=2).

6 Implementation

We have implemented our algorithm on a core 2 duo PC by MPI. The Message Passing Interface (MPI) is a standard developed by the Message Passing Interface Forum (MPIF) [19]. It specifies a portable interface for writing message-passing programs, and aims at practicality, efficiency, and flexibility at the same time [19]. For details about MPI and MPIF, see [19]. We implemented the algorithm with 5, 9, 15 and 20 nodes in MPICH2 at 20 iterations. The experimentations were done in the same condition in each iteration for all algorithms: majority, primary copy protocol and our proposed algorithm. We also considered our algorithm in clustering model and compared it with other algorithms. Fig 6 shows number of exchanged messages between nodes in each algorithm for *n* transactions where *n* is between 130 and 150.

By increasing number of nodes in all algorithms the number of messages passed are increased as well. As shown in Fig 6 the number of required messages for concurrency control of *n* transaction in primary copy and majority protocols is n×n and 3n(n-1) respectively while in proposed algorithm that is based on majority protocol this value equals (n-1) ×(1+2n), but this value in clustering model is $\frac{3\times\left(n^2-1\right)}{2}$. These formulas are gained from analytic computations when all links are intact and all messages are passed. In the majority protocol all messages are exchanged between initiating node (here called monitoring node) of algorithm and all other nodes. Therefore monitoring node has extra workload. That is also the case for primary copy algorithm and central node, which is responsible for locking data, which has too much work load, but here in the proposed algorithm, messages passing are shared between all nodes and we have good load balancing. In the proposed algorithm only 4(n-1) messages are exchanged between monitoring node and others. As it is known one of the factors for increasing algorithm’s runtime is number of message-passing. So here, by decreasing number of messages in comparison to the majority protocol we try to decrease algorithm's runtime. In clustering model, number of message-passing is minimum compared to the majority and proposed algorithm by applied external agent approach. For example in a system with 11 nodes there are 330, 230 and 180 messages for the majority protocol, the proposed normal algorithm and the clustering model respectively when all link are intact.

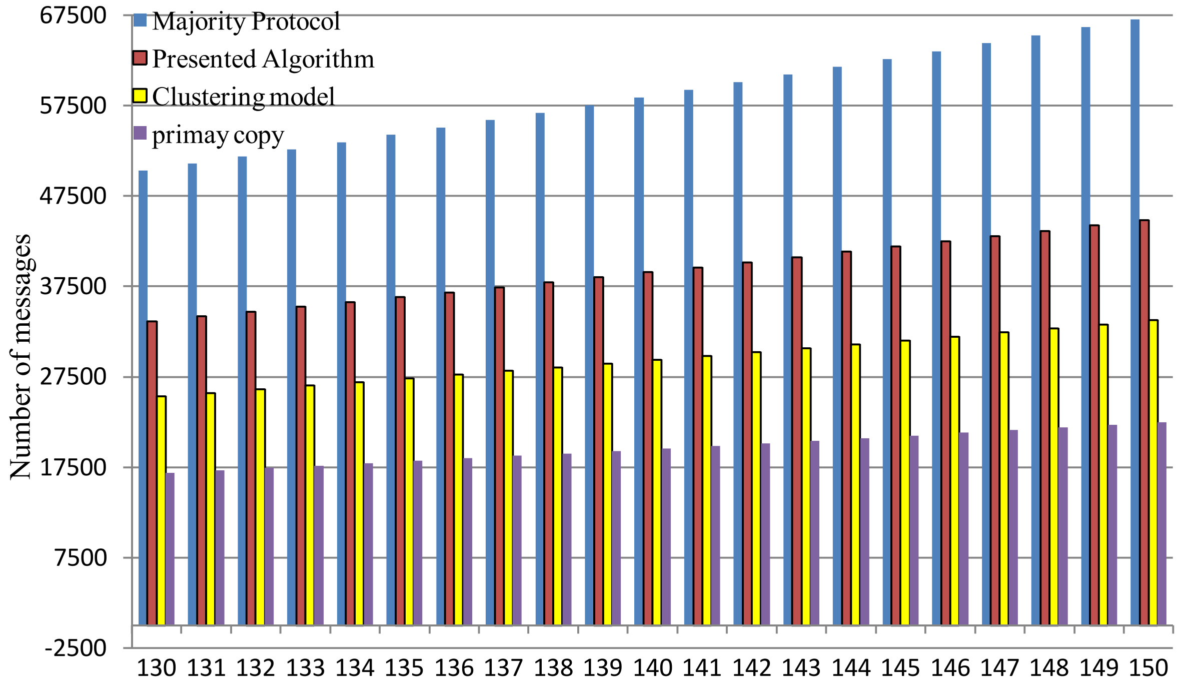


Fig. 6 Number of exchanged messages between nodes in the algorithms separately

As mentioned above one of the factors for increasing algorithm’s runtime is number of exchanged messages. Another factor is serial computation in primary copy and majority protocol for *n* transactions and these computations are done by central node in primary protocol and monitoring node in majority protocol. Computations’ Parallelism in proposed algorithm by multi agent approach is a way to decrease algorithm runtime. Since computations are done by all nodes, we have good load balancing in performance as well. Algorithms 1 to 3 represent concurrency control for primary copy, majority protocol and proposed algorithm respectively.

Algorithm 1 primary copy protocol

```
For each transaction
    Initiated node of transaction sends a message to central node
    Lock manager of central node checks for locking required data of transaction
    if its lock manager locks required data of transaction
        Central node will broadcast a message to all other nodes
        Transaction is run
End of checking transaction
```

Algorithm 2 Majority protocol

```
For each transaction
    Initiated node of transaction (monitoring node) broadcasts a message to all other nodes
    Each node checks for locking required data of transaction by its local lock manager
    if local lock manager locks data
        Each node will send a message (vote) to monitoring node
        Monitoring node counts received votes
        if it receives at least n/2+1 votes where n is number of nodes
            Monitoring node will broadcast a commit message to other nodes
End of checking transaction
```

Algorithm 3 Proposed algorithm

```
For n transactions
    monitoring node broadcasts a message to all other nodes (for initiating the algorithm and sends transactions' IDs)
    Each node checks for locking required data of n transactions by its local lock manager
    Agent i will send a message (metadata) to agent j if it can lock required data of transaction j (done in parallel)
    Each agent counts received votes, i.e. each agent i counts votes of transaction i (done in parallel)
    Each agent will broadcast a commit message to other agents if it receives at least n/2+1 votes for its corresponding transaction where n is number of nodes (done in parallel)
End of checking transactions
```

As previously mentioned and shown in algorithm 3 stages of algorithm (counting votes and sending messages) can be done in parallel and *n* transactions can be considered in a single iteration in contrast to the two other algorithms. This will increase algorithm's performance. Some main commands of algorithm in MPICH2 are described here.

Figs 7 to 10 show algorithms' runtime in 20 iterations separately in environments with 5, 9, 15 and 20 nodes.

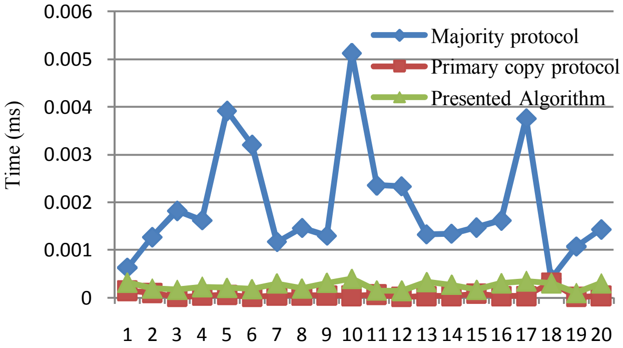


Fig. 7 Algorithms runtime in 20 iteration with 5 Node

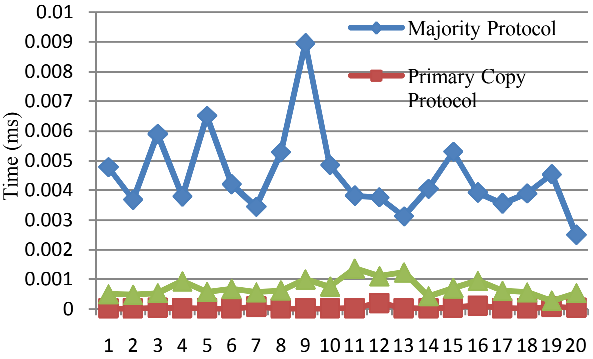
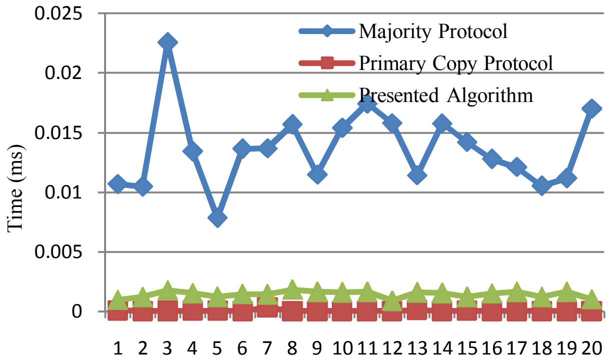


Fig. 8 Algorithms runtime in 20 iteration with 5 Node



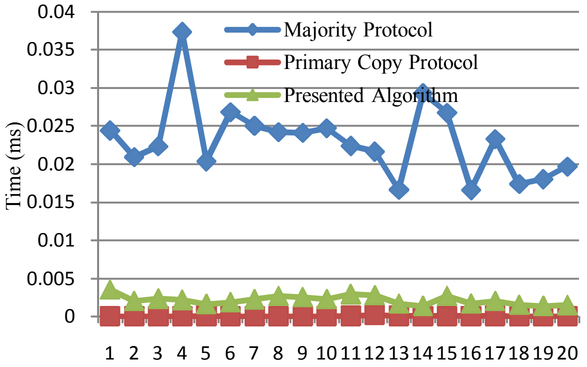


Fig. 9 Algorithms runtime in 20 iteration with 15 Node **Fig. 10** Algorithms runtime in 20 iteration with 20 Node

As it can be seen from these Figures, algorithm's runtime for primary copy protocol in each environment with different number of nodes is nearly the same. We will have a little increase in runtime if number of nodes increases; because we consider *n* transactions in each iteration where n is number of nodes.

In fig 11 we have compared the proposed algorithm in normal and clustering model with the majority protocol in an environment with 11 nodes and 20 iterations, and have compared their runtime in a case where all links are intact.

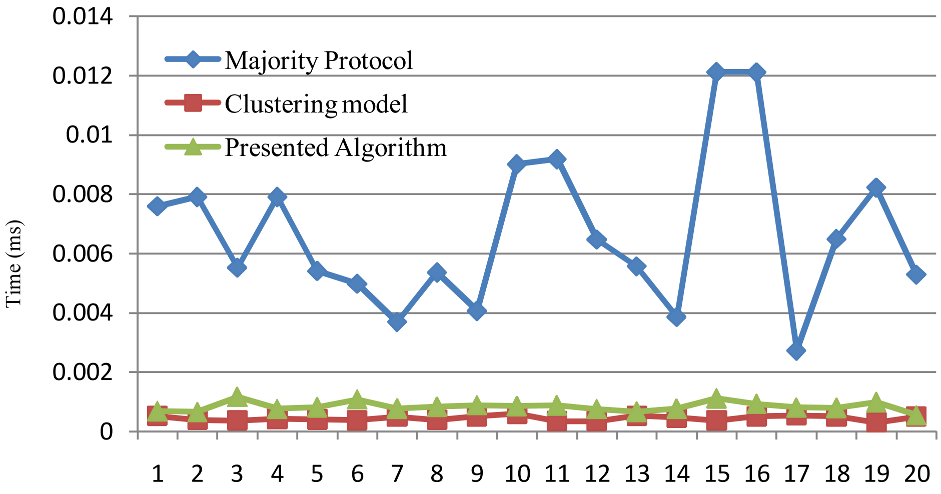


Fig. 11 comparing the proposed algorithm in normal and clustering model in an environment with 11 nodes and 20 iterations

As shown, runtime duration of algorithm in clustering model is lower than the two others due to decrement of message passing and parallel computation.

As represented in the clustering algorithm in the case of link failure and lack of enough votes for transaction, transaction will be considered in the other cluster and runtime will be variable depending on the number of link failures.

```
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

The subroutine MPI_COMM_SIZE returns the number of processes (here, number of Nodes) belonging to the communicator specified in the first argument. A *communicator* is an identifier associated with a group of processes. MPI_COMM_WORLD defined in mpif.h represents the group consisting of all the processes participating in the parallel job. MPI_COMM_RANK returns the rank of the process within the communicator given as the first argument.

Line 2 of algorithm 3 is done by the following command which broadcasts *m* elements from the root process (its rank is 0, the forth argument) to the other processes in the communicator MPI_COMM_WORLD. The triplet (buf, m, data type) specifies the address of the buffer, the number of elements, and the data type of the elements.

```
MPI_Bcast(buf, m, data type, 0, MPI_COMM_WORLD);
```

Sending and receiving message in other step can be done by following commands where buf is the initial address of the send buffer, count is the number of elements in the send buffer, datatype is the data type of each send buffer element, dest is the rank of the destination process in comm, tag is the message tag (can choose any integer in the range of $0..2^{32}-1$), comm is the communicator and request is the communication request. Arguments in the receive command are the same as send. The only difference is in the fourth argument that is the rank of source instead of destination.

```
MPI_Isend (buf, count, datatype, dest, tag, comm, request)
MPI_WAIT (req, status)
```

As defined in [19] this routine starts a nonblocking send at standard mode. The send buffer may not be modified until the request has been completed by MPI_WAIT, MPI_TEST, or one of the other MPI wait or test functions. The message sent by MPI_ISEND can be received by either MPI_RECV or MPI_IRECV.

```
MPI_IRECV (buf, count, datatype, source, tag, comm, request, ierror)
MPI_WAIT (req, status)
```

As defined in [19] this routine starts a non-blocking receive and returns a handle to a request object. You can later use the request to query the status of the communication or wait for it to complete. A nonblocking receive call means the system may start writing data into the receive buffer. Once the nonblocking receive operation is called, it does not access any part of the receive buffer until the receive is complete. The received message must be less than or equal to the length of the receive buffer. If all incoming messages do not fit without truncation, an overflow error occurs. If a message arrives that is shorter than the receive buffer, then only those locations corresponding to the actual message are changed. If an overflow occurs, it is flagged at the MPI_WAIT. For more details see [19].

7 Conclusions

In this paper, we have presented a new algorithm for concurrency control in distributed database systems. Our algorithm uses collaborative effort from a group of nodes to determine which transactions can be executed. First, the new collaborative approach based on multi-agent systems' approach has been represented, and then its clustering model has been offered for decreasing the number of messages in the represented approach. The usage of some of multi-agent systems features like cooperation between the nodes, independency and goal orientation, led the presented algorithm which is in fact an extension of majority protocol to better performance.

We have also concluded some advantages and disadvantages from our approach. One of the advantages of our proposed algorithm in comparison to both primary copy and majority algorithms is better load balancing. Decreasing the number of passed messages and algorithms' runtimes are other ones.

Generally, it can be seen from the illustrative example that our algorithm indeed determines the acceptance of requests successfully and shows the algorithm works efficiently even when some links are failed.

References

1. Bhargava, B., (1999). Concurrency Control in Database Systems. IEEE transactions on knowledge and data engineering, 11(1).
2. Bhargava, B., (1983). Concurrency Control and Reliability in Distributed Database System. Software Eng. Handbook, Van Nostrand Reinhold, 331-358.
3. Papadimitriou, C. H., (1979). The Serializability of Concurrent Database Updates. J. ACM, 26(4), 631-653.
4. Gray, J. N., Reuter, A., (1993). Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Mateo, Calif.
5. Bernstein, P. A., Hadzilacos, V., Goodman, N., (1987) Concurrency control and recovery in database systems. Addison Wesley, Reading.
6. Wooldridge, M., (2002). An introduction to multiagent systems. John Wiley & Sons Ltd.
7. Vidoni, R., Garcia-Sanchez, F., Gasparetto, A., Martinez-Béjar, R., (2011). An intelligent framework to manage robotic autonomous agents. Expert Systems with Applications, 38, 7430–7439.
8. Vlassis, N., (2007). Synthesis lectures on artificial intelligence and machine learning. Morgan and Claypool Publishers. Ch. A concise introduction to multiagent systems and distributed artificial intelligence.
9. Elamy, A., (2005). Perspectives in agent-based technology. AgentLink News (18), 19–22.
10. Agrawal, R., Carey, M. J., Livny, M., (1987). Concurrency Control Performance Modeling: Alternatives and mplications. ACM Transactions on Database Systems, 12(4).
11. Carey, M. J., Livny, M., (1989). Parallelism and Concurrency Control Performance in Distributed Database Machines. In Proceedings of the 1989 ACM SIGMOD.
12. Franaszek, P. A., Robinson, J. T., Thomasian, A., (1992). Concurrency Control for High Contention Environments. ACM Transactions on Database Systems, 17(2).
13. Thomasian, A., (1991). Performance Limits of Two-Phase Locking. In Proceedings of the IEEE International Conference on Data Engineering.
14. Breitbart, Y., Korth, H. F., Silberschatz, A., (2009). An optimistic concurrency control protocol for replicated databases. Fundamental Problems in Computing, Part II, 327-351.
15. Tamura, S., Hochin, T., Nomiya, H., (2011). Generation Method of Concurrency Control Program by Using Genetic Programming. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2011 12th ACIS International Conference on, Sydney, Australia, pp 175-180.
16. Chen, J., Wang, Y. F., Wang, J. P., (2011). Concurrency Control Protocol for Real-Time Database and the Analysis Base on Petri Net. Advanced Materials Research, Vols 143-144, 12-17.
17. Nizamuddin, M. K., Sattar, S. A., (2011). Algorithm for priority based concurrency control without locking in mobile environments. Electronics Computer Technology (ICECT), 2011 3rd International Conference on. Kanyakumari, 108 – 112.
18. Marchang, N., Datta, R., (2008). Collaborative techniques for intrusion detection in mobile ad-hoc networks. Ad Hoc Networks, 6, 508–523.
19. Aoyama, Y., Nakano, J., (1999). RS/6000 SP: Practical MPI Programming. International Technical Support Organization. IBM. 238.