Volume 5, Issue 3, March 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Quorum-Based Mutual Exclusion Algorithms: A Survey

[1]Shruti*, [2]Poonam Saini
[1]Research Scholar, [2]Assistant Professor
Department of Computer Science and Engineering
PEC University of Technology, Chandigarh, India

Abstract— The development of highly complex software, communication interfaces and the presence of low-cost processors are key factors towards the design of distributed applications. By distributing a computation, processes are permitted to run concurrently, share resources among themselves and at the same time working independent of each other. Distributed computations that involve sharing of resources require that only one process is allowed to enter its critical section (CS) at a time. Therefore, mutual exclusion is an important coordination issue for distributed algorithms. In the existing literature, different algorithms with varied performance levels have used various techniques to achieve mutual exclusion. Depending on the technique used, these algorithms have been classified as token-based and permission-based. In the paper, we present a broad survey and analysis of well known quorum-based mutual exclusion algorithms under permission-based category. The paper also presents the characteristics and related issues of mutual exclusion algorithms and concludes with their comparative analysis.

Keywords— Critical Section, Distributed Computing, Group Mutual Exclusion, Mutual exclusion, Quorum.

I. INTRODUCTION

A distributed system is generally defined as a collection of autonomous computers connected by a network which appears to its user as a single coherent system. These computers do not share memory, have no global clock and processes communicate through message passing. The features that make these systems different from others are their computational speedup, efficiency, reliability and convenient environment for resource sharing. Message propagation delays in distributed systems are finite, but they are unpredictable. The processes in the distributed systems must share common hardware or software resources to which concurrent access is not permitted. Shared resources must be synchronized such that only one process is allowed to use the resource at a given time. The procedure that allows this is known as Critical Sections (CS). A mutual exclusion algorithm frame rules for CS entry and conciliate any conflict that may arise when two or more sites wish to execute their CS at the same time.

The problem of mutual exclusion was originated back in 1965 when Dijkstra [19] in his work described it and tried to solve the problem. He stated that any solution to the mutual exclusion problem must satisfy the basic requirements [11]:

- *Safety property:* At most one process can execute its CS at a given time.
- *Liveliness property***:** The processes concurrently requesting to enter their respective critical sections should not be postponed indefinitely. If there is no process in CS then any process requesting to enter its CS must be allowed to do so in a finite time.
- *Fairness property:* Every process that wishes to enter the CS should get the chance to do so in finite time.

Dijkstra's algorithm is a deadlock-free mutual exclusion algorithm, however, do not guarantee fairness. Afterwards, various deadlock-free and fair algorithms have been proposed, however, quorum-based algorithms outperformed in terms of message complexity and synchronization delay.

A. Performance metrics of distributed mutual exclusion algorithms

The performance of mutual exclusion algorithms is measured by the following metrics [17]:

- *Message complexity:* Number of messages required by a site per CS execution.
- *Synchronization delay:* The time required after a site leaves the CS and before the next site enters the CS.
- *Response time:* The time interval when the request messages have been sent out and the site exit the CS. It does not include the waiting time of a request at a site before its request messages have been sent.
- *System throughput:* Rate at which the system executes requests for the CS.

Low and high load performance

The load is generally determined by the arrival rate of CS execution requests. There are mainly two types of load: "low load" and "high load."

In *low load* conditions, there is more than one request for the CS present simultaneously in the system.

In *Heavy load* conditions, there is always a pending request for CS at a site. As soon as, a site executes a request it immediately initiates action to execute its next CS request.

B. Group Mutual Exclusion

Mutual exclusion and concurrency are the two fundamental issues in distributed systems. Mutual exclusion allows access to a common resource, whereas concurrency is required to increase the system performance. Group Mutual Exclusion (GME) is a generalization of both of these features. Yuh-Jzer Joung, 2000 [1] studied the model of group mutual exclusion. The model was developed as an extension to the traditional mutual exclusion problem. In GME, each request for CS is related with a group. It concerns with the situation in which a resource is shared by the processes with a familiar property *i.e.*, belonging to the same group, while the processes with different properties must use the resources in mutually exclusive manner. The GME algorithm should satisfy the following properties [8]:

- *Group mutual exclusion:* Two processes belonging to different groups are not allowed to enter CS simultaneously.
- *Freedom from starvation:* A process waiting to enter CS should not wait indefinitely.
- *Concurrency:* If all requests belong to the same group then the requesting process should not wait for other process to leave its CS to gain entry to the CS.

The metrics that have been used to measure the performance are similar to that of traditional mutual exclusion, message complexity, bit-message complexity, synchronization delay, waiting time and concurrency, except concurrency that is specific to GME algorithm. While analyzing the performance message complexity, bit-message complexity, synchronization delay and waiting time should be minimum and concurrency should be maximum.

The distributed mutual exclusion (DME) algorithm has been classified into two groups, namely

- Token-based
- Permission-based

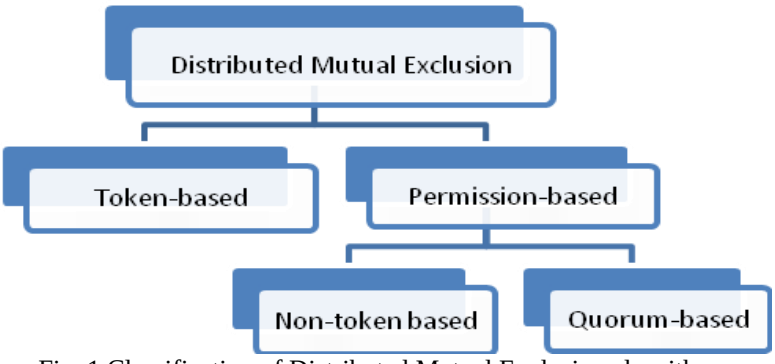

Fig. 1 Classification of Distributed Mutual Exclusion algorithms

The basic principle behind these two categories is the way in which the processes enter the CS. The token-based algorithm achieves mutual exclusion through an object called 'token'; it is unique and is shared among all the sites. A site can enter CS only if it is in ownership of the token. Although, the approach is faster than the other one, produces lesser message traffic and are not deadlock prone, they have poor failure resilience [12]. The token-based approach is at a high risk of issues like loss of tokens, duplicate tokens and regeneration of a unique token. Whereas in the permission-based group, a site can enter a CS only if it has permission from a set of nodes in the system.

"*A process wishing to enter its CS asks the other nodes to give their permission. A process enters its CS only after it had received permission from all nodes in a set.*"

Any conflicts that occur are solved by priority or by an order in which events appears. The problem of finding minimum number of nodes from which a process obtains permission has to be taken into consideration. The solution to the problem has a direct impact on the message complexity and synchronization delay. The permission-based approach has been further divided into two groups:

- Non-token based
- Quorum-based (coterie-based)

C. Quorum System

Garcia-Molina and Barbara, 1985 in their work [5] introduced the concept of a coterie for mutual exclusion in distributed systems. In simpler terms, quorum can be defined as "the minimum number of members of a group that needs to be present to make any decision for the group". For example, majority is needed for voting in a group. This is used to avoid any kind of partitioning and inconsistency that may arise during decision making process [16].

Definition. Let U is the universal set. A set of groups is a coterie C under U if and only if:

- For all $G \in C$, G should not be NULL and $G \subseteq U$.
- *Intersection property:* For all $G, H \in C$, there should be at least one common node between G and H.
- *Minimality property:* No $G, H \in C$ should be there, such that $G \supseteq H$.

It is also not important for all the nodes to appear in a coterie. The elements of coterie are also known as quorum sets or simply quorum. After the proposal of the concept of coterie by Garcia-Molina and Barbara, several different types of coterie for solving conflict had been proposed, for example [21, 22, 23]. In the paper, we review different quorum-based mutual exclusion algorithms.

II. RELATED WORK

In the section, different quorum based algorithms are discussed. Section 2.1 describes algorithms which uses quorum in traditional mutual exclusion and section 2.2 describes quorum-based group mutual exclusion algorithms. Before moving on to the background discussion, we will study *Ricart and Agrawala's* algorithm [6] which is one of the earliest known distributed mutual exclusion algorithm. The algorithm needs special mention because of its importance in mutual exclusion handling. Here, when a node *i* wants to entry CS it sends a request message to all the remaining nodes. On receiving a request, a node *j* sends a reply if it is neither requesting nor executing CS else a comparison between the priority of the incoming request and *j*'s own priority is made. A process with the higher priority gets the chance to enter its CS. The drawback of the algorithm is that a node on sending reply do not locks itself for that node which causes self-conflict. It has a message complexity of 2(*N*-1) and the synchronization delay is T. The two main issues Ricart and Agrawala face are node failure that leads to starvation and a lot of message traffic. These issues are solved by quorum based algorithms.

A. Quorum-based algorithms

The quorum-based mutual exclusion algorithms are different from others in the following two ways:

- A site does not require permission from all the sites, but only from a subset of the sites (quorum).
- A site can send out only one REPLY message at any time, that also after it has received a RELEASE message from the previous REPLY.

Maekawa, 1985 [4] gave the first quorum-based mutual exclusion algorithm. It states that "Considering all other sites in my 'status set', it is ok for you to enter CS" [10]. We can consider the algorithm as a stepping-stone in the distributed mutual exclusion algorithm development due to the reasons below:

- It assumes that the nodes are arranged in a logical manner.
- Self-conflict is avoided.
- The logical grid used in Maekawa's algorithm and its modifications, have been used in many other algorithms later.

For mutual exclusion to hold, the conditions that are to be satisfied are:

M1: ($\forall i\ \forall j$: $i \neq j$, $1 \leq$ i, j $\leq N$::the intersection between *Ri* and *Rj* should not be NULL)

M2: ($\forall i$: $1 \leq i \leq N$::$Si \in Ri$)

M3: ($\forall i$: $1 \leq i \leq N$ :: $|Ri|$ = K)

M4: The site *Sj* is contained in K number of *Ri*'s, $1 \leq i, j \leq N$.

Maekawa in his work established a logical structure. Each node is associated with a set, i.e., quorum, consisting of nodes. The set has a nonempty intersection with every set associated with each node. A node *i* who wants to enter CS must obtain permission from all other nodes in its set *Si*. As the set intersects with other set, mutual exclusion is guaranteed here. Each node *k* in *Si* is associated with another set *Sk* in the system. Node *k* acts as an arbitrator for requests received from *i* and also from members of set *Sk*. Hence, when node *k* in *Si* gives permission to node *i*, then *i* has a right to enter CS. Each arbitrator node grants only a single permission to a single node. Maekawa's algorithm achieves mutual exclusion.

The message complexity is $3\sqrt{N}$ and the synchronization delay is $2T$. The serious issue with which the algorithm suffers from is deadlock, as a site is completely locked by other sites and the requests are not prioritized by their timestamps (TS). Maekawa's deadlock-free algorithm overcomes the issues.

Maekawa's earlier algorithm has the problem of deadlock so to get by that issue another deadlock-free algorithm was introduced. Deadlock occurs at a site whenever a request with higher priority arrives and waits, because the site has already sent a REPLY message to a request with lower priority. The algorithm handles deadlocks by allowing a site to release a lock only if its request's timestamp is larger than that of other request waiting for the same lock. Additional messages are required to handle deadlocks (INQUIRE, FAILED and YIELD) and may exchange these messages even though there is no deadlock. As a result, the maximum number of messages is increased from $3\sqrt{N}$ to $5\sqrt{N}$ per CS [17].

Singhal, 1991 [7] after observing Maekawa's algorithm, presented a class of Maekawa-type mutual exclusion algorithm. He found that the deadlock can be removed without additional message exchange, i.e., without using INQUIRE, FAIL, and YIELD. To avoid deadlocks the locks on sites must be made mutable and the timestamp should be used more cleverly. Here, if a request with a higher priority arrives at a site after the site has been locked by a lower priority request, it converts the exclusive lock to a shared lock. A request with a higher priority than the priority of all the requests succeeds in locking a site. The request having lower priority preempts the lock if it has not been able to lock all the sites in its request set yet. As a result, a deadlock-free algorithm is created; however, the message complexity increases from $\sqrt{N}$ to N. The synchronization delay is T. The condition of optimality is maintained in the algorithm as in the Maekawa's deadlock-free algorithm.

Agrawal and *Abbadi*, 1991 [18] gave a tree quorum where nodes are arranged in a binary tree structure. A quorum is constructed by selecting a path that starts from a root to any of the leaves. In case of any node failure, that node is substituted by the paths starting with its children and ending with leaves. The algorithm exhibits the property of graceful-degradation in case of failure-free environment.

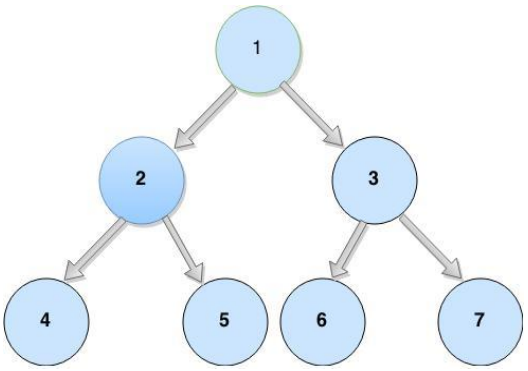

Fig. 2 A tree of 7 nodes

The algorithm works as follow: When a node *i* want to enter CS, it sends a time-stamped request to all the nodes available in the tree quorum. Then it waits for the permission from all members of the quorum. Each node maintains a queue of pending requests in an ordered form. When a request message is at the top of the queue, the node sends its permission to the node from where the request originated. When node *j* receives *i's* request, it checks for a smaller timestamp request at the head of the queue. The request from *i* is placed in the queue, otherwise an inquire message is sent to check whether the node has collected permission from all other nodes in its quorum. If it succeeds, it ignores the inquire message. In case it not, it cedes the permission by sending a yield message to the inquiring node. When *j* receives the yield message it sends the permission to *i*. A node when releases the CS, sends a relinquish message to all nodes in the quorum. Reception of which removes the node whose request has been served from the queue and sends its permission to the next request at the head of the queue.

In the best case, the algorithm takes O(log *n*) sites to form a tree quorum and the worst case tree quorum size is O((*n*+1)/2). It can tolerate the failure up to *n*−O(log *n*) sites.

Luk and Wong, 1997 [22] presented two algorithms. In the first method, he suggested a triangular grid of nodes where any two quorums will meet at only one node. The quorum constructed using the triangular grid has the size of $\sqrt{2}\sqrt{N}$ approximately. The quorums can be row-based, column-based, or both can be used alternately for each new request. The second scheme is based on cyclic difference set and cyclic block design in combinatorial theory. Here, the quorum is strictly symmetric and deals with arbitrary *N*.

Cao et al., 1998 [20] came out with the new idea; he proposed a delay-optimal quorum-based mutual exclusion algorithm. The algorithm reduces the synchronization delay by still maintaining the low message complexity. Instead of first sending a RELEASE to unlock the site which sends a REPLY to the next site to enter CS, a site exiting the CS directly sends a REPLY to the site which will enter CS next. Therefore, as a site exits the CS only one message delay occur before the next site enters the CS. The result leads to reduction in the synchronization delay from 2*T* to *T*. It is a coterie-based algorithm and is independent of the coterie (quorum) being used. The message complexity is *ck*, where *c* ranges between 3 and 6 and *k* is the average size of the quorum used. The value of *k* can be as low as log *N*. Starvation and deadlock is impossible in this algorithm.

Ousmane Thiare, 2009 [13, 15] proposed a permission based mutual algorithm which is an enhancement of the algorithm developed by Maekawa, 1985 [4]. The messages required by the algorithm are in the range of 3*M* to 5*M* per CS where *M* is number of intersecting nodes. The scheme was that the Maekawa performs better by restricting the communication related to the algorithm only to the intersecting nodes in the quorum. The basic logic of the algorithm is kept same. The number of intersecting nodes should be less than the number of nodes in the quorum. Some conditions were proposed to construct the quorum, which are:

- $\forall i \forall j$, intersection of *Si* and *Sj* should not be NULL, $i \neq j$, $1 \leq i, j \leq y$ where y = number of quorums, $y \leq N$.
- Node *i* should belong to at least one of the quorums.
- Number of nodes in the quorum need not to be equal.

The main aim of considering only intersecting nodes was that it plays an important role is maintaining mutual exclusion condition throughout the entire network. The algorithm ensures mutual exclusion, and is deadlock and starvation free.

Md. Ashiqur Rahman et al., 2010 [14] solved the problem of large participants by proposing a cluster-based two-layer hierarchical algorithm for DME. Here, the nodes in a network are partitioned into several nonintersecting groups known as cluster. Clusters being free of coordinator, there is no chance of failure. The lower layer is formed by the nodes inside the clusters. From each of the clusters in the lower layer an arbitrary node is taken which forms the upper layer. When a node *x* wants to enter the CS, it requires to have the permission from the nodes of the lower layer cluster firstly. When *x* gets permission at the lower layer, it needs to get permission from the upper layer. To do this, *x* selects an arbitrary node *y* from the upper layer cluster members as representing node. Node *x* is identified as the represented node to y. Node *y* executes the mutual exclusion in the upper layer on behalf of *x*. If *y* gets permission from all the members in the upper layer, it informs *x* about the authority and gets the permissions from both layers to enters CS. A simple quorum based ME algorithm is used inside a cluster for collecting consensus. The fault tolerance of the algorithm is high and acquires less communication cost particularly in case of node failure.

In the figure below, the nodes a, b and c of the lower layer act as the arbitrator and forms the upper layer. Each node has the information about the members of the upper as well as lower layer.

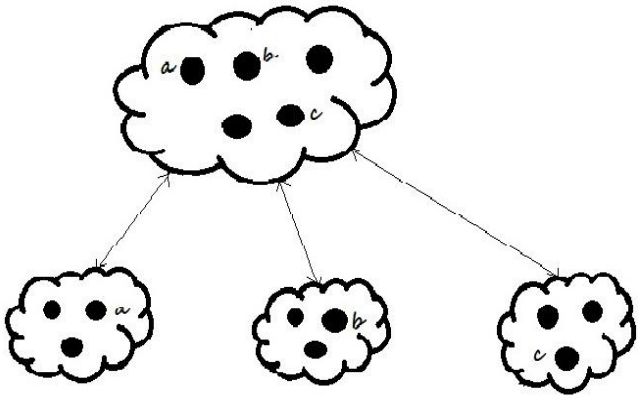

Fig. 3 Architecture of Hierarchical algorithm

Singhal, 1992 [25], presented a dynamic mutual exclusion algorithm. The information structure is assigned to the nodes that change with time as it learns about the system through messages. Each node *i* maintains an information structure consisting of two sets: *Request set Ri*, from which *i* must obtain permission to enter into its CS, and *Inform set Ii* that specifies nodes to which *i* must send its permission after it releases the CS. For all requesting nodes *i* and *j*, $Ri \cap Rj \neq \phi$, that guarantees the mutual exclusion.

Initially, *Ri* = {1, 2, 3, 4,. . ., i} is the request set of the node *i*. *Ri*'s cardinality decreases in the stepwise manner from left to right. A node *i* acquire permission from all nodes in *Ri* to enter CS. Set *Ri* keeps only those nodes to which permission is sent, and remove the nodes from which permission is to be received. Whereas, set *Ii* keeps only those nodes from which a request is received. Nodes to which *i* send its permission after releasing the CS are deleted from *Ii*. The staircase pattern is preserved in the system even though the CS has been executed several times. The message complexity of the algorithm under low load is *N-1*, and *3(N-1)/2* under heavy load.

Sung-Hoon Park et al., 2014 [25] provided the fault-tolerant quorum-based mutual exclusion problem in an asynchronous system. To solve the problem, a failure detector was determined. The paper considered that the problem is solvable by *M*∗, which is a modal failure detector. It is based on the concept of Chandra and Toueg, 1996 [26, 27], where perfect failure detectors *P*, weak failure detectors *W* and eventually weak failure detectors ♦*W* were studied.

A perfect failure detector *P* starts by trusting every node, whereas *M* ∗ initiates its working by suspecting every node. *M*∗ stop suspecting only when it receives a confirmation message at least once from every node. Implementation using *M** has advantages compared to that of *P*. Here, the assumption is made that at least one quorum is available. Deadlock situation is not considered.

The working is as follow: The process that wants to enter CS waits for a quorum whose all members are active. After it finds out the quorum, it sets its status to *try*. Then it sends a message to all members of the quorum for token. The process asking for a token does not proceed until the all tokens are received from the quorum. It then enters CS. On exiting CS, the process sends a *return* message to every member of the quorum from which it received tokens. When the failure detector history is received by the node from *M* ∗ and it is known that a node holding token is dead then it regenerates the token again. The message complexity is moderate here and the synchronization delay is *T*.

Mohamed Naimi et.al, 2013 [28], presented a deadlock free quorum-based DME algorithm. Here, each group is arranged in a logical ring of $\sqrt{n}$ processes. A process that wishes to enter its CS sends the request to its successor on the ring. It enters CS only when it receives its own request after a complete round. The request message visits all the processes in the list according to the order. The message complexity of the algorithm is $2\sqrt{n}$.

B. Quorum-Based Group Mutual Exclusion Algorithms

Y. J Joung, 2003 [2] presents the surficial quorum system for group mutual exclusion. The system is easy to construct and helps in minimizing load. Here, quora that belongs to the same group need not intersect. The surficial quorum system when used with Maekawa's algorithm can allows up to $\sqrt{(2n/m(m-1))}$ processes to share resource simultaneously, where n= total processes and m=total groups. It is in contradiction to the ordinary quorum system where only one process is allowed to access the resource at a time.

The modifications Maekawa_M and Maekawa_S were introduced so that number of processes that access a resource is not limited. Maekawa_M allows a node to assign multiple locks to the requests of the same group. The drawback of Maekawa_M is that if any conflict arises between the requests of different groups then the multiple locks are to be taken back to avoid deadlock which may cause high complexity. To overcome the problem, Joung proposed a Maekawa_S which avoids deadlock by locking quorum members in particular order. Both of these algorithms require a prior knowledge of number of groups to construct group quorum.

Ranganath Atreya et al., 2007 [8] described the algorithm, independent of the quorum system used and handle dynamic changes that may occur at runtime. It does not consider the group quorum system and solve the GME problem using surrogate-quorum. We take the leader-follower approach along with the surrogate quorum. Processes requesting to enter CS try to lock their respective quorums. A process can gain entry either by locking all its members inside the

quorum or by receiving an invitation from some another process already present in the forum. The former process is called a leader and latter as follower. To inform the leader about the other requests, the members of the quorum on sending lock also sends compatible request that is currently present in the queue. The leader does not release its quorum till all of its followers' leaves the forum. Here we use the leader's quorum as a surrogate for its followers; that is why it is known as surrogate-quorum. Its performance is better than that of Maekawa_M. The algorithm can be extended to satisfy properties like concurrent entry and unnecessary blocking freedom.

Hirotsugu Kakugawa et al., 2008 [9**]** had taken two types of tokens in their algorithm: the main token and sub tokens. The main token is unique to the particular system. A sub token is generated by the main token, and number of sub tokens may vary. A process having the main token can enter the CS only if there is no process present in the CS. In case other process in the same group is in CS, a process enters CS only by receiving a sub token. The basic idea of the algorithm is as follows: A process having the main token notifies other processes in a quorum that it holds the main token. While making a request, the process sends a request messages to processes in a quorum. A request message received is forwarded to the main token holder and is received by it. It is then enqueued in the main token. Lastly, the request is granted, and the token is sent to the requesting process.

The algorithm contributes to less message complexity and flexible concurrency control mechanism. In order to reduce message complexity, it uses a coterie as a communication structure.

III. DISCUSSION AND OBSERVATION

The table below shows the performance of the algorithm described above. The comparison between message complexities and synchronization delay is made.

TABLE I COMPARISON BETWEEN DIFFERENT ALGORITHMS

S.no	Algorithm	Message Complexity	Synchronization Delay	Observation
1.	Ricart and Agrawala's	2(*N*-1)	*T*	Earliest known DME
2.	Maekawa's	3$\sqrt{N}$	2*T*	Uses quorums
3.	Maekawa's deadlock-free algorithm	5$\sqrt{N}$	2*T*	Uses additional messages to remove deadlock
4.	Singhal's	*N*	2*T*	Removes Maekawa's deadlock problem
5.	Agrawal-el Abbadi	log *N*	2*T*	Uses tree quorum
6.	Maekawa_M	*qN/d*	2*T*	Fixed group set
7.	Maekawa_S	*q*	*q*+1	Fixed group set
8.	Surrogate	*q*	*bqr*	Leader-follower approach used
9.	Token GME	2*N*-1	2*T*	Main token and sub token used
10.	Cao's	*ck*	*T*	Send Reply message directly to the next node instead of sending Release message
11.	Ousmane Thiare's	3*M* to 5*M*	2*T*	Intersecting nodes in the quorum are taken
12.	Luk and Wong	$\sqrt{2}\sqrt{N}$		Two methods to construct a quorum proposed
13.	Singhal's dynamic information structure	*N-1*, under low load *3(N-1)/2*, under high load	*T*	Dynamic Information structure used
14.	Sung-Hoon Park and Seon-Hyong Lee	Moderate	*T*	Uses fault detectors to provide a fault tolerant quorum based mutual exclusion algorithm
15.	Naimi and Thiare's	$\sqrt{2N}$		Uses the concept of logical ring to execute CS

where,
N = number of processes in the system,
d = degree of group quorum,
b = maximum number of processes from which a node receives a request for CS,
q= maximum number of nodes in a quorum,
k= average size of quorum used,
M = intersecting nodes
and r = maximum size of a request.

IV. CONCLUSIONS

In the paper, we have analyzed and compared the characteristics of different quorum-based mutual exclusion algorithms in terms of message complexity, synchronization delay and other features. It has been observed that quorum-based mutual exclusion algorithms have excellent load balancing capability. The message complexity is also lower than other mutual exclusion techniques like token-based and message-based. The choice of particular algorithm depends upon the factors like topology, system requirements, delay, message complexity, etc. Furthermore, group mutual exclusion using quorums can be used in applications that require concurrency, allowing more than one similar processes to be in critical section at a time. Therefore, permission based mutual exclusion algorithms using quorums are better than other mutual exclusion algorithms in many ways and gives better performance.

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my supervisor Dr. Poonam Saini (Asstt Professor), for her continuous support and encouragement during the tenure of my work. I also thank her for the opportunity she gave me to carry on my research work.

REFERENCES

[1] Y.J. Joung, "*Asynchronous Group Mutual Exclusion*" Distributed Computing, vol. 13, no. 4, pp. 189-206, 2000.
[2] Y.J. Joung, "*Quorum-Based Algorithms for Group Mutual Exclusion*" IEEE Trans. Parallel and Distributed Systems, vol. 14, no. 5, pp. 463-475, May 2003.
[3] Y.J. Joung, "*The Congenial Talking Philosophers Problem in Computer Networks*" Distributed Computing, pp. 155-175, 2002.
[4] M. Maekawa, "*A√N Algorithm for Mutual Exclusion in Decentralized Systems*", ACM Trans. Computer Systems, vol. 3, no. 2, pp. 145-159, May 1985.
[5] H.Garcia-Molina, D. Barbara, "*How to Assign Votes in a Distributed System*", J. ACM, vol. 32, no. 4, pp. 841-860, October 1985.
[6] G. Ricart, A.K. Agrawala, "*An Optimal Algorithm for Mutual Exclusion in Computer Networks*" Comm. ACM, vol. 24, no. 1, pp. 9-17, January 1981.
[7] M. Singhal, "*A Class of Deadlock-Free Maekawa-Type Algorithms for Mutual Exclusion in Distributed Systems*" Distributed Computing, vol. 4, no. 3, pp.131-138, 1991.
[8] R. Atreya, Sathya Peri, Neeraj Mittal, *"A Quorum-based Group Mutual Exclusion Algorithm for a Distributed System with Dynamic Group Set*", IEEE Trans. Parallel and Distributed Systems, vol. 18, no. 10, pp. 1345-1360, December 2007.
[9] Hirotsugu Kakugawa, Sayaka Kamei, Toshimitsu Masuzawa, *"A token-based Distributed Group Mutual Exclusion Algorithm with Quorums*", IEEE Trans. Parallel and Distributed Systems, vol. 19, no. 9, pp. 1153-1166, September 2008.
[10] M. Singhal, "*A Taxonomy of Distributed Mutual Exclusion*", Journal of Parallel and Distributed Computing 18, pp. 94-101, 1993.
[11] Martin G. Velazquez, "*A Survey of Distributed Mutual Exclusion Algorithm*", Technical Report CS-93-116, Department of Computer Science, Colorado State University, September 6, 1993.
[12] P.C.Saxena, J.Rai, "*A Survey of permission-based distributed mutual exclusion algorithm*", Computer Standards and Interfaces, pp. 159-181, May 2003.
[13] Ousmane Thiare, "*A solution to improve algorithm for Distributed mutual Exclusion by Restricting Message exchange in Quorum*", Department of Computer science, Gaston Berger University, Senegal. IEEE, pp.38-42, August 2009.
[14] Mohammad Ashiqur Rahman, Md. Mostofa Akbar, *"A Permission based Hierarchical Algorithm for Mutual Exclusion*", Journal of computers, vol 5, no.12, pp. 1789-1799, December 2010.
[15] Ousmane Thiare, Papa Alioune Fall, "*Using Maekawa's Algorithm to Perform Distributed Mutual Exclusion In Quorums*", Advances in Computing, vol.2, no.4, pp. 54-59, 2012.
[16] Soma Chaudhuri, "*Advanced topics on Networks and Distributed Algorithms*", Lecture 13, Monday, March 4, 2013.
[17] Kshemkalyani and M. Singhal, "*Distributed Computing- Principals, Algorithms and Systems*", Cambridge University press, 2008.
[18] Agrawal and El Abbadi, "*An Efficient and Fault-Tolerant Solution for Distributed mutual exclusion*", ACM Transc. On Computer Systems, vol.9, no.1, pp. 1-20, February 1991.

[19] E.W. Dijkstra, "*Co-operating sequential processes*", in: F.Genuys (Ed.), Programming Languages, Academic Press, New York, pp. 43–112, 1965.

[20] Guohong Cao, Mukesh Singhal, Yi Deng, Naphtali Rishe, Wei Sun, "*A Delay-Optimal Quorum-Based Mutual Exclusion Scheme with Fault-Tolerance Capability*", Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98), IEEE Computer Society, Amsterdam, 1998, pp. 444–451, May 1998.

[21] Marko Vukolic, "*The Origin of Quorum Systems*", IBM Research-Zurich.

[22] Wai-Shing Luk, Tien-Tsin Wong, "*Two New Quorum Based Algorithm for Distributed Mutual Exclusion*", Distributed Computing systems, 1997, Proceedings of the 17th International Conference, pp. 100-106, May 1997.

[23] Takashi Harada, Masafumi Yamashita, "*Transversal Merge Operation: A Nondominated Coterie Construction Method for Distributed Mutual Exclusion*", IEEE Transc. on Parallel and distributed Systems, vol. 16, No. 2, pp.183-192, February 2005.

[24] M. Singhal, "*A dynamic information-structure mutual exclusion algorithm for distributed systems*", IEEE Transactions on Parallel and Distributed Systems, vol. 3, No. 1, pp.121– 125, 1992.

[25] Sung-Hoon Park, Seon-Hyong Lee, "*Quorum-based mutual exclusion in asynchronous distributed systems with unreliable failure detectors*", J Supercomput (2014)67:469-484.

[26] Chandra TD, Hadzilacos V, Toueg S, "*The weakest failure detector for solving consensus*", J ACM 43(4), pp. 685-722, 1996.

[27] Chandra TD, Toueg S, "*Unreliable failure detectors for reliable distributed systems*", J ACM 43(2), pp.225-267, 1996.

[28] Mohamed Niami, Ousmane Thiare, "*A Distributed Deadlock-Free Quorum-Bases Algorithm for Mutual Exclusion*", International Journal of Computer Science and Information Security, vol. 11, No. 8, August 2013.

