A FAULT TOLERANT ALGORITHME FOR MULTIPLE RESOURCES SHARING IN DISTRIBUTED SYSTEMS

[1]TAHAR ALLAOUI, [2]MOHAMED BACHIR YAGOUBI
Laboratory of mathematics and computer science, Computer science department, University of Laghouat, Algeria
E-mail: [1]t.allaoui@mail.lagh-univ.dz, [2]m.yagoubi@mail.lagh-univ.dz

ABSTRACT

In this paper we present a fault tolerant algorithm to resolve the K mutual exclusion problem in distributed systems. Our algorithm, which is a token based, ensures K mutual exclusion with an interesting message complexity where the number of exchanged messages to satisfy each request is between 0 message in the best case and K+2 messages in the worst case, where K denotes the system' resources number. In this algorithm, we introduce a fault tolerance mechanism that tolerates the likelihood crash of several nodes at the same time and ensures the system well-functioning after messages loss without relying on complex election algorithms to generate new tokens.

Key words: *Mutual exclusion, K-mutual exclusion, Critical section, Distributed systems, Fault tolerance, Tokens.*

1. INTRODUCTION

A distributed system is a set of autonomous sites interconnected by a communication network. In such systems, there is neither global clock, nor common memories, and the communication between sites is ensured only by messages exchange.

The mutual exclusion (or simply ME) problem is one of the most known problems in distributed systems. ME ensures the access of the system sites to a single shared resource in a mutually exclusive manner, which means that at any given time only one site at most can execute a part of code named the critical section (CS) which manipulates the resource. The proposed solutions to this problem can be classified into two categories of algorithms; Permission-Based [1-4] and Token-Based [5, 6] as an example. In the first category, the requesting site must ask for authorization from other sites, then waits for their permissions to access the CS. In the second category, a particular message called token is used to ensure ME, only the possession of the token allows a site to use the shared resource.

The K-Mutual Exclusion problem (KME for short) is a generalization of the ME problem, where K identical copies of the same resource are available in the system, thus K sites at most can access simultaneously their CS. The algorithms resolving this problem can be also classified as permission based [7, 8] and token based algorithms [9].

The sites of a distributed system may likely be affected by an adverse event with a negative impact on the well-functioning of the system; this event is the failure of one or several sites simultaneously. Usually, the cause of the failure is unknown in advance. In the KME problem, the failure of the token-holding site leads to the token loss which affects the system's coherence, and calls for complex election algorithms to designate a site that generates the lost token. For this reason fault tolerant KME algorithms were proposed to ensure smooth operation continuity even in the presence of case failures. In these algorithms, two main requirements have to be satisfied. First, reducing exchanged messages for each entry to the CS to avoid network overload. Second, avoiding complex election algorithms to generate new tokens in case of token loss.

In this paper, we propose a fault tolerant KME algorithm in distributed systems, this algorithm is token based, and ensures KME by an interesting message complexity that varies between 0 message -favorable case- and K+2 -worst case-. The fault tolerance method that we use in our algorithm overcomes the failure of several sites simultaneously. It also permits the generation of the lost tokens without the use of complex election algorithms.

The rest of the paper is organized as follows. Section 2 briefly presents related work and the most known algorithms in the field. Section 3 describes our initial algorithm. Section 4 explains fault tolerant algorithm's principle. Section 5 shows correctness proof. Section 6 considers algorithm's performance. Section 7 discusses simulation results. Section 8 concludes the paper and gives some perspectives.

2. RELATED WORKS

The mutual exclusion problem has been well treated by many algorithms [1-6], that can be considered as the corner stone of the field. Since the KME is an extension of the ME problem, it was also treated by several algorithms, the first proposed algorithms [7, 8] can be considered as improvements of the ME algorithms. Raymond's algorithm [7] was the first attempt to solve the KME problem. The algorithm is permission-based extending Ricart and Agrawala's principle [2]. The requesting in [7] site sends request messages to the (n-1) other sites and waits for at least (n-k) permissions, 2n-k-1 messages are sufficient to access the CS. Srimani and Reddy proposed a Token-Based Algorithm [8] based on Suzuki and Kasami's solution [5], the message complexity is between 0 and n+k-1. The use of k-coterie in Kakugawa et al. algorithm [10] ensures the KME with a complexity of O (n log n). The algorithm of Naimi [9] is a Token-Based and uses the directed graphs, in which the number of messages exchanged per CS is between 0 and 2(n-1). Baldoni et al proposed in [11] a KME algorithm in prioritized systems.

The research was not limited to the ME and the KME problems, but it was extended to treat other problems such as the Group Mutual Exclusion [12], the ME in grid applications [13], and the KME in Mobile Ad hoc networks (Manet)[14-19].

The fault tolerant ME and KME algorithms are reported in the literature [20-25], in these algorithms not only the consistency of the resources use is ensured, but also the proper functioning in presence of failures. The algorithm of Ming et al [20] uses a distributed queue strategy and maintains alternative paths at each site to provide a high degree of fault tolerance. Hélary and Mostefaoui have proposed a hybrid algorithm [21] based on the use of the open-cube structure which offers symmetric proprieties that facilitate the implementation of the fault tolerance. The algorithm of Agrawal and El Abbadi [22] uses a dynamic tree which gets updated after the sending of a token; its fault tolerance mechanism is based on the use of Lamport's logical clocks [26]. Jiang [23] uses a mechanism that allows more nodes to be in the CS concurrently, and reduces the message overhead. In [24], Loallemi et al. used clusters. They consider that there is at least one node that does not crash in the cluster, this node has the token and the token requests are transmitted through broadcasts between clusters. Bouillageut et al. proposed in [25] a method that detects and tolerates f number of failures in a constant grid.

We proposed in [27] an initial algorithm that ensures KME with an interesting message complexity; however, it was vulnerable to nodes failures and tokens loss. Therefore, we propose a fault tolerant KME algorithm (FT algorithm) as an improvement of the initial algorithm (NFT algorithm). In the next section we will introduce the principle of the initial algorithm, then we will present the mechanism of the fault tolerance used in the improvement.

3. THE NFT ALGORITHM

Since the number of exchanged messages is a major concern for the KME algorithms, we used in our algorithm a logical structure that ensures the KME through the use of reduced number of messages for each entry to the CS.

3.1 The logical structure

In the system, there are N sites numbered from 1 to N, and K available resources. The logical structure divides the N sites into K groups. The belonged sites of any group address their requests to a leader which is a particular site in the group designated to play the role of a coordinator. Initially, each leader holds a free token. The leaders are interconnected with each other via a bidirectional ring that allows the tokens passing between them.

The creation of this structure involves two steps

1. **Creating groups**: which is based on the identity of the sites, for example, if K = 3, and the sites are numbered from 1 to 14, sites 1, 4, 7, 10 and 13 will be in the same group, sites 2, 5, 8, 11 and 14 will be in the second group, and finally, 3, 6, 9 and 12 will be in the third group. The leaders

of the groups will be the sites 1, 2 and 3 respectively.

2. **Connecting the groups**: Only the K leaders will be connected to each other via a bidirectional ring as for a leader J, J-1 is its left neighbor and J+1 is its right neighbor.

3.2 Principle of the NFT algorithm

In order to ensure KME, we should use as many tokens as the available resources in the system. Initially, each leader holds a free token, this token is used to serve the requests in the group, if an additional token is needed, it will be asked from a neighbor leader, hence, a leader can hold up to K free tokens at a time. A requesting site in a group sends its demand to the group leader, this latter will grant the request by sending a free token if it is available, otherwise the request will be saved in a queue, and the leader will ask for an additional token from its neighbors via the ring. The leader that receives the request from a neighbor will respond positively to this request by sending a token if it holds free ones. Otherwise it will propagate the request to another neighbor.

Receiving a token by a leader permits to serve the first request in the queue, therefore, the requesting site that receives the token can access the CS. After releasing CS, the token is sent back to the leader.

A judicious method has been created to avoid costly message exchange between leaders through the use of a mechanism that allows sending tokens via the shortest path between sender and receiver. As a result of such method the number of messages used decreases in each entry to the CS.

3.3 Local variables

In our NFT algorithm, we have two types of sites, the leaders and the simple sites.

For a simple site
Status: Refers to the status of the site, it belongs to {Out, Requesting, In_CS}. Initially set to Out.
Leader: The identity of this site's group leader.

For a leader
Status: Refers to the status of the site, it belongs to {Out, Requesting, In_CS}. Initially set to Out.
Leader: at the beginning is initialized to Nil.
Right_neighbor: the identity of right neighbor in the ring.
Left_neighbor: the identity of left neighbor in the ring.
Free_Tokens: an integer variable indicates the number of free tokens held by the leader, initialized to 1.
Present_Tokens: an integer variable which specifies the number of tokens used in the group, which is initialized to 1.
Requesting: a queue used to store requests it contains the identity of the requesting sites, initially it is empty.

4. THE FAULT TOLERANT ALGORITHM

NFT algorithm was vulnerable to nodes failure and tokens loss. So we present a fault tolerant version of the algorithm aiming to solve node failure, especially the failure of nodes holding tokens which causes the tokens loss.

The fault cases that may cause serious problems in our algorithm are as follows.

- The failure of a leader: The local variables held in the leader such as the requesting sites queue, the number of free tokens and used tokens in the group are necessary for the proper functioning of the algorithm, when a leader fails, all its variables will be lost, hence, tokens will be lost and the requesting sites can never access their CS.
- The failure of a site holding a token: In this case, the token is lost, so the number of tokens in the system is decreased and will not reflect the correct number of resources, and as a result there will be lesser sites in CS than the available resources in the system. It is obvious that this case affects the proper functioning of the algorithm.
- The failure of a requesting site: The identity of a requesting site is stored in the requesting sites queue. When a free token is available it will be sent to the first site in the queue, and if this site is already crashed, then the token will be lost, and

consequently we face the same problem as the failure of a site holding a token.

The token loss is the common feature and the main disadvantage of the 2nd and the 3rd case. So in order to ensure well-functioning of the system we must use complex algorithms of election to designate a site that will generate lost tokens. We aim in our algorithm to avoid the use of such algorithms.

4.1 Basic idea

The basic idea of the fault tolerance mechanism used in our FT algorithm is inspired by far from the algorithms [28,29] that use the principle of supervisor, which is a particular site designated to play a specific role whenever a failure occurs.

Since leaders are very important to ensure the well-functioning of the algorithm, we have appointed a supervisor to each leader which will be the site with the smallest identity in the group. Each supervisor will keep a copy of all variables of its leader and when the leader's variables are updated, the supervisor's variables must be updated as well. In the case of a leader failure, its supervisor would take its place, so the variables would not be lost and the system continue to function properly.

Depending on the nature of the site, we can distinguish four cases of failure, in which every case the algorithm must apply specific actions.

- **The failure of a requesting site**: The leader removes the identity of the requesting site from the queue and informs the supervisor.
- **The failure of a site holding a token**: The leader generates a new token, updates its local variables and informs the supervisor.
- **The failure of a supervisor**: The leader must choose a new supervisor; this new supervisor must be informed by all the values of the leader's local variables.
- **The failure of a leader**: In this case the supervisor must act as a leader, first it must inform the neighbors of the old leader and the members of the group of its identity by sending a particular message, second it chooses a new supervisor, and then updates its local variables and informs the new supervisor.

In the FT algorithm, we assume:

- Message delays are bounded.
- Communication links are reliable and do not fail.
- When a site fails, its failure can be detected by the other nodes in the system.
- Failed sites may eventually recover.
- A leader and its supervisor don't fail in the same time.

4.2 Local variables

The likelihood failure of some leaders or supervisors may set any site belongs to the group as a candidate to play the role of a supervisor or a leader, and for this reason the site needs the same variables of the lost leader, but the difference lies in the initialization. The variables used at each site will be the same variables of the leader that have been mentioned in section 3.3. In addition to these variables another two more variables are required:

Supervisor: for a leader, this variable contains the identity of the supervisor, for the other sites the value is nil.

IN_CS: a list containing the identities of sites holding tokens in the same group (the sites that are executing their CS). The usefulness of this variable appears in the cases of failure, when a site in the list fails it gets detected by the leader and a new token gets generated, and the identity of the failed site gets removed from the list.

4.3 Messages used by the algorithm

In order to update the supervisor's local variables, the leader must send the new value of each variable whenever it changes, so we can use a single message containing all the variables: the left neighbor, the right neighbor, the number of free tokens, the present tokens in the group, the requesting sites queue, and the sites' list in CS.

Two problems may appear with the use of such a message. First, the size of the message could be very large, especially when there is a large number of requesting sites, so the exchange of this message would be difficult. Second, the frequency change of the different values in the message is not the same, for example, the requesting queue must be updated with every new request and the supervisor must be informed, however the value of the variable left neighbor for example could remain unchanged, thus it would be unnecessary to send it each time.

For this reason we will use a distinct message for each variable.

Update_Left_neighbor (x): x is the identity of the left neighbor.
Update_right_neighbor (x): x is the identity of the right neighboring
Update_free_tokens (x): replaces the old value of free token by x
Update_present_tokens (x): replaces the old value of present tokens by x
Update_requesting (Q): replaces the old requesting queue by Q
Update_In-CS (Q): replaces the old list by Q.
In addition to those messages, another message will be used when a supervisor takes the place of its leader:
New_Leader (i, j): sent by the supervisor **j** in the case of the failure of the leader **i**. By this message, the members of a group and the neighbors of the old leader are informed that the old leader **i** has failed, and a new leader **j** takes its place.

5. CORRECTNESS PROOF

Theorem 1. The algorithm guarantees K Mutual Exclusion.

Proof: To ensure KME we must ensure that at any given time K sites at most simultaneously execute the CS. Since our algorithm is a token based, thus only the possession of a token allows a site to access the CS, and because we use K tokens in the algorithm the KME is granted. However, a problem may arise if a token holding site fails, the token will be lost as a consequence, and we end with at most K-1 sites in CS. To avoid this problem, in our algorithm when a token is lost the leader will generate a new one and serves the next request, therefore, the KME is ensured by the algorithm.

Theorem 2. The algorithm is starvation free.

Proof: Starvation may occur when a requesting site cannot access the CS in the presence of free resources, while other sites would do. This situation may happen with the failure of a leader, in such a case all the requesting sites belonging to the failed leader group will wait indefinitely because of the loss of their queue, even in the existence of free resources in the system. The fault tolerance mechanism used in our algorithm ensures that this situation would never take place, because the supervisor keeps a copy of the queue, and whenever the leader fails, the supervisor takes its place and serves all the requests in a finite time.

Theorem 3. The algorithm is deadlock free.

Proof: The deadlock appears if there is a circular waiting chain between sites, which may be caused by the failure of a token holding site. In our algorithm, the leader generates a token whenever it detects the failure of a token holding site in the belonged group, so the next request in the queue will be served, and the circular waiting chain between sites may never occur. Thus, the algorithm is deadlock free.

6. PERFORMANCE OF FT ALGORITHM

The fault tolerance mechanism of our algorithm overcomes the simultaneous failure of several sites. It can tolerate:

- The fail of K sites holding the tokens simultaneously, in this case, the leaders can detect the failure of these sites, so each leader will generate new tokens according to the number of failing sites in its group, the identity of the failing sites will be removed from the queue, and the supervisors will be informed.
- The fail of K supervisors in the same time, in this case, each leader will choose a new supervisor among the sites in its group, the new supervisors will receive the values of the local variables.
- The fail of K/2 non-adjacent leaders in the same time, the supervisors of the failing leaders will take place; the failing leaders mustn't be adjacent in order to avoid the conflicts that may occur when the supervisors take place of the failing leaders.

In all these cases, our algorithm ensures that the system does not cease to function smoothly, moreover, the lost tokens will be generated by the leaders, avoiding the use of complex election algorithms to identify the site that will generate the

Journal of Theoretical and Applied Information Technology
20th May 2015. Vol.75. No.2
© 2005 - 2015 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

lost tokens, our principle makes easier this task and minimizes the number of exchanged messages while ensuring KME and fault tolerance.

7. SIMULATION RESULTS

The performance of the NFT algorithm was shown in [27], the behavior of the algorithm compared to some well known algorithms [7, 8] in the field is very interesting with a better message complexity.

In this section we present the simulation results of the proposed FT algorithm. We compared its behavior with that of the NFT algorithm by causing intentionally some faults in the system. In order to create different scenarios, we varied one of the metrics each time: the number of resources the number and the number requesting sites. The simulation compares the message complexity and the waiting delay between the proposed algorithm and the NFT algorithm.

In Figure 1 we notice that the number of messages for each entrance to CS decreases with the increase of the number of resources for both algorithms, however, the FT algorithm requires more messages comparing with the NFT algorithm due to the exchange of messages between new leaders and other sites when a fault occurs.

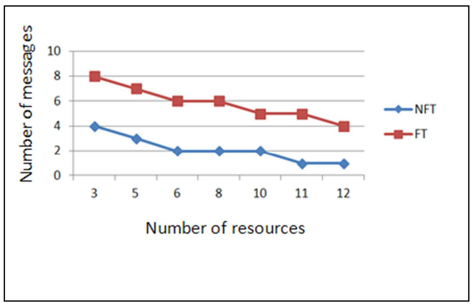

Figure 1. Number Of Messages Against Number Of resources

In Figure 2 the waiting time curve has the same direction of change with that of Figure 1 because the increase of the number of resources minimizes the load on leaders and thus requests will be satisfied in shorter time. We notice that the difference between the NFT curve and the FT curve is not considerable because the extra exchanged messages haven't an influence on the waiting time.

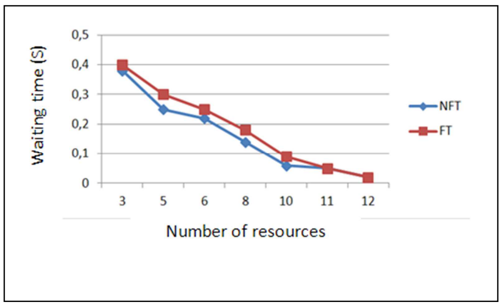

Figure 2. Waiting Time Against Number Of Resources

In Figure 3 we notice a light increase of the number of message with increasing of the requesting nodes, for the same reason of the precedent curves, the FT algorithm requires a little more exchanged messages. Curves in Figure 4 are practically identical.

In all figures we notice that the FT algorithm has the same behavior with the NFT algorithm in spite of a light inevitable difference of the messages number because of the necessary exchange of information between the new leaders and the other sites in case of fault. we conclude that the fault tolerance mechanism that we use in our algorithm keeps the same behavior and performance of the initial algorithm, the waiting time remains practically the same, and the difference in the number of exchanged messages has no influence on the interesting performance of the algorithm.

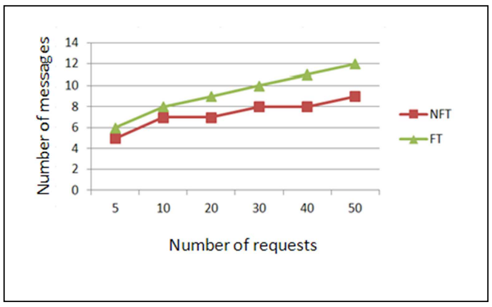

Figure 3. Number Of Messages Against Number Of Requests

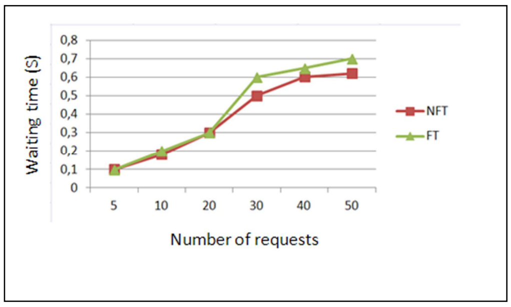

Figure 4. Waiting Time Against Number Of Requests

8. CONCLUSION

In this paper we have presented an efficient fault tolerant K mutual exclusion algorithm in distributed systems, the algorithm ensures the K mutual exclusion by using K tokens, and each access to the critical section is guaranteed by the exchange of 0 to K+2 messages.

In this algorithm we used the supervisor mechanism to make the algorithm fault tolerant, this mechanism resists several nodes crashes, and resists the loss of the tokens as well, without using the complex election algorithms.

Compared with the initial algorithm, our proposed algorithm ensures KME by the same number of exchanged messages per CS, 0 messages is sufficient in the best case, and K+2 messages in the worst case. However, a light increase of messages for the FT algorithm comparing with the NFT algorithm was noticed in our study, this increase is inevitable due to the necessity of extra messages in case of faults, but this increase didn't affect the behavior of the algorithm and has no considerable influence on the FT algorithm performance.

The main contribution of this paper is the development of a fault tolerant algorithm that ensures the KME and overcomes the likelihood of token loss. Simulation results show that message complexity between FT and NFT algorithms has no significant difference. However, our study was only limited to token loss, thus the algorithm needs to be more improved to deal with other fault types such as messages loss and communication link failure. These two problems can be considered in future improvements. Despite these shortcomings, the future work may involve the implementation of the algorithm's principle and fault tolerance mechanism in other systems such as mobile ad hoc networks (Manet), and vehicular ad hoc networks (Vanet).

REFERENCES

[1]. Saxena, P.C.; Rai, I. : "A survey of permission-based distributed mutual exclusion algorithms" , Computer Standards & Interfaces. 25, 2003, pp. 159-181.

[2]. Ricart, G.; Agrawala, A.K.: "An Optimal Algorithm for Mutual Exclusion in Computer Networks", Comm. ACM, Vol. 24, N°.1, 1981, pp. 9-17.

[3]. Carvalho, O.S.F.; Roucairol, G.: "On mutual exclusion in computer networks", Comm. ACM, Vol. 26, N°. 2, 1983, pp. 146-147.

[4]. Maekawa, M.: "An √n Algorithm for Mutual Exclusion in Decentralized Systems" , ACM Trans. Computer Systems,Vol. 3, N°. 2, 1985, pp. 145-159.

[5]. Suzuki, I.; Kasami, T.: "A distributed mutual exclusion algorithm", ACM Trans. Computer Systems, Vol. 3, N°. 4,1985, pp. 344-349.

[6]. Naimi, M. ; Trehel, M.: "A log(n) distributed mutual exclusion algorithm based on the path reversal", Journal of Parallel and Distributed Computing, Vol. 34, 1996, pp. 1–13.

[7]. Raymond, K.: "A Distributed Algorithm For Multiple Entries to a Critical Section", Information Processing Letters 30, 1989, pp. 189-193.

[8]. Srimani, P.K.; Reddy, R.L.N.: "Another distributed algorithm for multiple entries to a critical section", Information Processing Letters 41, 1989, pp. 51-57.

[9]. Naimi, M: "Distributed algorithm for K-entries to a critical section based on the directed graphs", ACM, Op. Systems Review, (Oct 1993).

[10]. Kakugawa, H. ; Fujita, S.; Yamashita, M.; Ae, T.: "A distributed k-mutual exclusion algorithm using k-coterie", Information Processing letters, Vol. 49, 1994, pp. 213-218.

[11]. Baldoni, R.; Ciciani, B.: "Distriuted algorithms for multiple entries to a critical section with priority", Information processing Letters, Vol. 50, 1994, pp.165-172.

[12]. Atreya, R.; Mittal, N: "A quorum-based group mutual exclusion algorithm for distributed system with dynamic group set", IEEE transaction on parallel and distributed systems, 18, 2007, pp. 1345-1360.

[13]. Bertier, M.; Arantes, L.; Sens, P: "Distributed mutual exclusion algorithms for grid applications : A hierarchical approach", Journal of Parallel and Distributed Computing, 66, 2006, pp. 128-144.

[14]. Walter, J.; Welch, J.; Vaidya, N.: "A mutual exclusion algorithm for ad hoc mobile networks", Wireless Networks, 9, 2001, pp. 585–600

[15]. Walter, J.; Cao, G.; Mohanty, M.: "A k-mutual exclusion algorithm for wireless ad-hoc networks", in : Proc of the first annual

Workshop on Principles of Mobile Computing (POMC 2001, pp. 171-180

[16]. Baldoni, R.; Virgillito, A.; Petrassi, R.: "A distributed mutual exclusion algorithm for mobile ad-hoc networks", in : Proceedings of the Seventh IEEE International Symposium Computers and Communications, 2002, pp. 539-544.

[17]. Chen, Y.; Welch, J.: "Self-stabilizing mutual exclusion using tokens in ad hoc networks", in : Proc. Of IALM'02, ACM, 2002, pp. 34-42.

[18]. Derhab, A.; Badache, N.: "A distributed mutual exclusion algorithm over multi-routing protocol for mobile ad hoc networks", International journal of parallel, emergent and distributed systems. 23, 2008, pp. 197-218

[19]. Masum, S.M.; Akbar, M.M.; Ali, A.A.; Rahman, M. A.: "A consensus-based ℓ-Exclusion algorithm for mobile ad hoc networks", Ad Hoc Networks, Vol. 8, 2010, pp. 30-45.

[20]. Liu, M. T.; Chang, Y. I.; Singhal, M.: "An Improved O (log N) Mutual Exclusion Algorithm for Distributed Systems", ICPP (3), 1990, pp. 295-302.

[21]. Hélary, J. M.; Mostefaoui, A.: "A O(log 2 n) fault-tolerant distributed mutual exclusion algorithm based on open-cube structure", 14th International Conference on Distributed Computing Systems (ICDCS), 1994, pp. 89-96 (1994)

[22]. Agrawal, D.; El Abbadi, A.: "A token-based fault-tolerant distributed mutual exclusion algorithm", Journal of Parallel and Distributed Computing. 24, 1995, pp. 164-176.

[23]. Jiang, J.: "A fault tolerant h-out-of-k mutual exclusion algorithm using cohorts coteries for distributed systems", in : proceedings of 5th international conference on parallel and distriuted computing, application and technologies PDCAT , 2004, pp. 267-273.

[24]. Loallemi, M.; Mansouri, Y.; Rasoulifard, A.; Naghibzadeh, M.: "Fault tolerant hierarchical token-based mutual exclusion algorithm", In International Symposium in Communications and Information Technologies , 2006, pp.171-176.

[25]. Bouillageut, M.; Arantes, L.; P. Sens: "A Timer-Free Fault Tolerant K-Mutual Exclusion Algorithm", Fourth Latin-American Symposium on Dependable Computing LADC 09, 2009, pp. 41-48 .

[26]. Lamport, L.: "Time, clocks , and the ordering of events in distributed system", Communications of the ACM, Vol. 21, 1978, pp. 558-565

[27]. Allaoui, T.; Yagoubi, M.B; Djoudi, M.; Ouinten, Y: "A fast token based algorithm for multiple resources sharing in distributed systems", in proceedings of The International Conference on Innovations in Information Technology, IIT 2008, pp. 24-28.

[28]. Bouabdallah, A.; König, J. C.; Yagoubi, M.B.: "A fault tolerant algorithm for the mutual exclusion in real time distributed systems", Journal of computing and Information, Vol. 1, 1995, pp. 438-454

[29]. Bouabdallah, A.; König, J. C.; Yagoubi, M.B.: "An improvement of O(log N) mutual exclusion algorithm to make it fault tolerant", 10th International Conference On parallel and distributed computing systems, 1997.