The impact of clustering on token-based mutual exclusion algorithms

Julien Sopena, Luciana Arantes, Fabrice Legond-Aubry, and Pierre Sens

LIP6, Université Pierre et Marie Curie, INRIA, CNRS
{julien.sopena,luciana.arantes,fabrice.legond-aubry,pierre.sens}@lip6.fr

Abstract. We present in this article a theoretical study and performance results about the impact of the Grid architecture on token-based mutual exclusion algorithms. To this end, both the original token-based Naimi-Tréhel's algorithm and a hierarchical approach, suitable to cope with the intrinsic heterogeneity of communication latencies of Grid environments, are studied and evaluated.

1 Introduction

A Grid is usually composed of a large number of machines gathered into small groups called clusters. Nodes within a cluster are linked using local networks (LAN) whereas clusters are connected by wide area network (WAN) links. Therefore, Grids present a hierarchy of communication delays where the latency of sending a message between nodes of different clusters is much higher than sending a message between nodes within the same cluster.

As Grid resources can be shared, applications that run on top of a Grid usually require their processes to get exclusive access to one or more of these shared resources (critical section). Thus, a distributed mutual exclusion algorithm, which ensures that exactly one process can execute the critical section (CS) at any given time (*safety* property) and that all CS requests will eventually be satisfied (*liveness* property), is an important building block for Grid applications. Moreover, the performance of a mutual exclusion algorithm can have a major impact on the overall performance of these applications.

Mutual exclusion algorithms can be divided into two groups: *permission-based* (e.g. Lamport [5], Ricart-Agrawala [11], Maekawa [6], etc.) and *token-based* (Suzuki-Kazami [15], Naimi-Tréhel [8], Raymond [10], etc.). The algorithms of the first group are based on the principle that a node enters a CS only after having received a permission from all the other nodes (or the majority of them [6]). In the second group, a unique system-wide token is shared among all nodes, and its possession gives a node the exclusive right to enter the critical section. Token-based algorithms usually have an average lower message cost than permission-based ones and many of them have a logarithmic message complexity $\mathcal{O}(log(N))$ with regard to the number of nodes N. Hence, they are more suitable for controlling concurrent access to shared resources of Grids since N is often very large.

However, existing token-based algorithms do not take into account the above-mentioned hierarchy of communication latencies. To overcome this problem, we have presented in a previous article [14] a generic composition approach which enables the combination of any two token-based mutual exclusion algorithms: one at *intra-cluster* level and a second one at *inter-cluster* level. By using our composition mechanism, efficient mutual exclusion algorithms for Grids can be built where communication latency heterogeneity is not neglected. Furthermore, they can be easily deployed by just "plugging in" token-based algorithms on each levels of the hierarchy. Performance evaluation tests conducted on a Grid platform have shown that the good choice for an *inter-cluster* mutual exclusion algorithm depends on the frequency with which the distributed processes of the application request for the shared resource, i.e., the degree of parallelism of the application.

We now propose in this article to study the impact of the Grid architecture on token-based mutual exclusion algorithms with and without our composition approach, i.e., *hierarchical* and *flat* mutual exclusion algorithms respectively. Basically, we would like to know if our *hierarchical* approach is more suitable for a Grid platform than the *flat* one when the number of clusters increases, and which is the number of clusters that a Grid platform should have such that the *hierarchical* algorithm presents the highest performance gain when compared to the *flat* one.

In order to answer to the above questions, we did both a theoretical study about the probability of an algorithm's message to be sent over an inter cluster link and we conducted evaluation performance experiments on a Grid emulation cluster platform. For the experiments, we have chosen the Naimi-Trehel's [8] token-based mutual exclusion algorithm, which maintains a dynamic logical tree to transmit processes requests for the execution of the critical section. Thus, the *flat* algorithm consists of the original Naimi-Trehel's algorithm while the *hierarchical* one uses our composition approach with Naimi-Trehel's algorithm at both *intra* and *inter* levels. Our choice can be explained based on the results published in our previously mentioned article [14]: when using Naimi-Trehel's algorithm at *inter-cluster* level, we obtained the smallest delay to get access to the shared resource when compared to other token-based algorithms that use other approaches for transmitting critical section requests such as a logical ring structure or broadcasting. We should also emphasize that we considered applications with different behaviors in our experiments since we also would like to know if the degree of the parallelism of an application has an influence on our study.

The remainder of this paper is organized as follows. Section 2 briefly describes Naimi-Tréhel algorithm. In section 3, we present our compositional approach for mutual exclusion algorithms. Performance evaluation results and a theoretical study about the effect of clustering on token-based algorithms with and without our composition approach are presented in section 4. Some related work is given in section 5. Finally, the last section concludes our work.

2 Naimi-Tréhel's algorithm

Naimi-Tréhel's algorithm [8] is a token-based algorithm which keeps two data-structures: (1) A logical dynamic tree structure in which the root of the tree is always the last node that will get the token among the current requesting ones. Initially, the root is the token holder, elected among all nodes. This tree is called the *last tree*, since each node i keeps the local variable *last* which points to the *last* probable owner of the token; (2) A distributed queue which keeps critical section requests that have not been satisfied yet. This queue is called the *next queue*, since each node i keeps the variable *next* which points to the *next* node to whom the token will be granted after i leaves the critical section.

When a node i wants to enter the critical section, it sends a request to its *last*. Node i then sets its *last* to itself and waits for the token. It becomes the new root of the tree. Upon receiving i's token request message, node j can take one of the following action depending on its state: (1) j is not the root of the tree. It forwards the request to its *last* and then updates its *last* to i. (2) j is the root of the tree. It updates its *last* to i and if it holds an idle token, it sends the token to i. However, if j holds the token but is in the critical section or is waiting for the token, it just sets its *next* to i. After executing the critical section itself, j will send the token to its *next*.

3 Composition approach to mutual exclusion algorithms

In this section we present our mutual exclusion composition approach. We consider that there is one process per node, called *application* process.

Our approach consists in a hierarchy of mutual exclusion algorithms: a per cluster token-based mutual exclusion algorithm that controls critical section requests for processes within the same cluster and another algorithm that controls *inter-cluster* requests. The former is called the *intra* algorithm while the latter is called the *inter* algorithm. Each *intra* algorithm controls an *intra* token while the *inter* algorithm controls an *inter* token. An *intra* algorithm of a cluster runs independently from the other *intra* algorithms. An important advantage of our approach is that the original algorithms chosen for both layers do not need to be modified. Furthermore, it is completely transparent for *application* processes which just call the classical mutual exclusion functions *CS_Request()* and *CS_Release()*. Thus, whenever an *application* process wants to access the shared resource, it calls the *CS_Request()* (Figure 1, line 14) of the *intra* algorithm. Upon receiving the *intra* token, the process executes the CS. After executing it, the process calls the *CS_Release()* (line 17) of the same *intra* algorithm to release it.

In order to avoid that *application* processes of different clusters simultaneously access the critical section, we have introduced a special process within each cluster, called the *coordinator*. The *inter* algorithm runs on top of the *coordinators* and allows a *coordinator* to request access to the shared resource on behalf of an *application* process of its own cluster. *Coordinators* are in fact hybrid processes which participate in both the *inter* algorithm with the other *coordinators* and the *intra* algorithm with their cluster's *application* processes. Holding the *intra* token of its cluster is sufficient and necessary for an *application* process to enter the CS since the *intra* token is granted to it only if the *coordinator* of its cluster holds the *inter* token, which is unique for the whole system.

The guiding principle of our approach is described in the pseudo code of Figure 1. The *pendingRequest( )* function (line 21) informs the coordinator if there are token requests of the respective level waiting to be satisfied.

```
1   Coordinator Algorithm ()
2       /* Initially, it holds the intra-token */
3       while TRUE do
4           if ¬ intra.pendingRequest() then
5               Wait for intra.pendingRequest()
6           inter.CS_Request()
7           /* Holds inter-token. CS */
8           intra.CS_Release()
9           if ¬ inter.pendingRequest() then
10              Wait for inter.pendingRequest()
11          intra.CS_Request()
12          /* Holds intra-token CS */
13          inter.CS_Release()

14  CS_Request ()
15      ...
16      Wait for Token

17  CS_Release ()
18      ...
19      if pendingRequest() then
20          Send Token

21  pendingRequest ()
22      return $\begin{cases} TRUE & \text{if } \exists \text{ pending request} \\ FALSE & \text{otherwise} \end{cases}$
```

Fig. 1. Coordinator algorithm

Initially, every *coordinator* holds the *intra* token of its cluster and only one of them holds the *inter* token. Thus, when an *application* process wants to enter the critical section, it sends a request to its local *intra* algorithm by calling the *Intra.CSRequest()* function. The *coordinator* of the cluster, which is the current holder of the *intra* token in this case, will also receive such a request. However, before granting the *intra* token to the requesting *application* process, the *coordinator* must hold the *inter* token too. The coordinator then calls the *Inter.CSRequest()* function (line 6) in order to request the *inter* token. Upon receiving it, the coordinator gives the *intra* token to the requesting *application* process by calling the *Intra.CSRelease()* function (line 8). After executing the CS, the *application* process calls the *Intra.CSRelease()* function in order to release the *intra* token.

A *coordinator* which holds the *inter* token must also treat *inter* token requests received from the *inter* algorithm. However, it can only grant the *inter* token to another *coordinator* if it holds its local *intra* token too. Holding the token ensures that there is no *application* process within its cluster in the critical section. Thus, before releasing the *inter* token, the *coordinator* sends a request to its *intra* algorithm asking for the *intra* token by calling the *Intra.CSRequest()* function (line 11). Upon obtaining the *intra* token, the *coordinator* can grant the *inter* token to the requesting *coordinator* by calling the *Inter.CSRelease()* function (line 13).

4 Performance Evaluation

Our performance evaluation aims at studying and comparing the influence of the Grid architecture in both the original Naimi-Tréhel mutual exclusion algorithm (*flat* algorithm) and with our composition approach using Naimi-Tréhel at both levels (*hierarchical* algorithm). To this end, the number of nodes of the Grid was set to 120 but the number of clusters varied: 2, 3, 4, 6, 8, 12, 20, 30, 40, 60, and 120. The experiments were conducted on a dedicated cluster of twenty-four Bi-Xeon 2.8 Ghz with 2GB of RAM machines where a Grid environment with 120 virtual nodes was emulated. There is one process per virtual node. For those configurations where the number of virtual clusters is greater than the number of available machines, nodes of the same virtual cluster run on the same machine. This approach prevents side effects of *intra* cluster communication.

Network latencies between clusters were emulated by using the flexible tool DUMMYNET [12] which allows injection of network delay, bandwidth limitation, and packet loss. Hence, for emulating several virtual clusters, every message exchanged between two virtual clusters goes through a dedicated machine, a P4 3Ghz machine, which runs a FreeBSD DUMMYNET. *Intra* cluster communication latency is 0.5ms while *inter* cluster latency is 20ms. Machines are connected by a 140 Gbits/s Ethernet switch.

The mutual exclusion algorithms and the coordinator were written in C using UDP sockets. Each application process that runs on a single virtual node executes 100 critical sections. Each of them lasts 10ms. Every experiment was executed 10 times and the presented results are the average value.

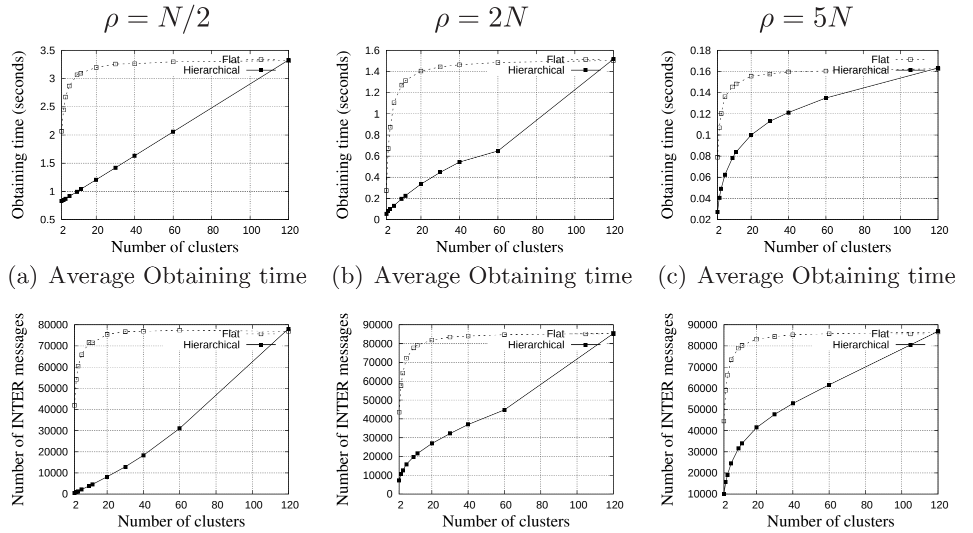
The behavior of an application can be characterized by ρ which expresses the frequency with which the CS is requested. ρ is equal to the ratio β/α, where α is the time taken by a node to execute the CS while β is the mean time interval between the release of the CS by a process and its next request.

We have developed several applications having **low**, **intermediate**, and **high** degrees of parallelism. Considering N as the total number of *application* processes, the three degrees of parallelism can be expressed respectively by:

- **Low Parallelism** ($\rho \leq N$): An application where the majority of *application* processes request the critical section. Thus, almost all *coordinators* wait for the *inter* token in the *inter* algorithm.
- **Intermediate parallelism** ($N < \rho \leq 3N$): A parallel application where some nodes compete to get the CS. Hence, only some *coordinators* wait for the *inter* token.
- **High Parallelism** ($3N \leq \rho$): A highly parallel application where concurrent requests to the CS are rare. The whole number of requesting *application* processes is small and usually distributed over the Grid.

In order to evaluate the *flat* algorithm as well as the *hierarchical* one, two metrics have been considered: (1) the *number of inter-cluster messages* and (2) the *obtaining time*, i.e., the time between the moment a node requests the critical section and the moment it gets it.

Considering $N = 120$, for each experiment, we have measured the *obtaining time* (Figures 2(a), 2(b), and 2(c)) and the number of *inter* cluster messages (Figures 2(d), 2(e), and 2(f)) for both algorithms when the number of cluster ranges from 2 to 120. Figures 2(a) and 2(d) correspond to a low parallel degree application ($\rho = N/2$); Figures 2(b) and 2(e) correspond to an intermediate parallel degree application ($\rho = 2N$); Figures 2(c) and 2(f) correspond to a high parallel degree application ($\rho = 5N$).


$\rho = N/2$ $\rho = 2N$ $\rho = 5N$

(a) Average Obtaining time (b) Average Obtaining time (c) Average Obtaining time

(d) Number of *inter* messages (e) Number of *inter* messages (f) Number of *inter* messages

Fig. 2. Impact of the number of clusters

4.1 Flat algorithm

We start by studying the impact of the number of clusters of the Grid on both the *obtaining time* and the number of *inter* cluster messages in the original *flat* Naimi-Tréhel algorithm. We can observe in Figure 2, that the curves related to this algorithm have a quite similar form. Independently of ρ, all curves present a hyperbolic form: a significant growth when the number of clusters varies from 2 to 12. This growth is then strongly reduced, becoming almost null, when the number of clusters is greater than 40.

In order to explain the form of such curves, we propose to theoretically study the frequency with which a *flat* mutual exclusion algorithm sends an *inter* cluster message, i.e., the probability $\mathcal{P}$ that the destination node of a message does not belong to the same cluster of the message's sender. To this end, we consider a Grid architecture composed of N nodes uniformly distributed over c clusters. Without loss of generality, we also suppose that a node can send a message to itself. This assumption models two successive accesses to the critical section by the same node. Then, we get the following probability $\mathcal{P}$:

$$\mathcal{P} = \frac{N - \frac{N}{c}}{N} = 1 - \frac{1}{c}$$

This equation is totally in accordance with the form of the curves of Figures 2 for the *flat* algorithm. It also shows that such a probability does not depend on the number of nodes N whenever they are uniformly distributed over the Grid, i.e., it depends only on c. A last important conclusion from this equation is that the clustering effect due to the communication latency heterogeneity of a Grid has a negligible impact on the order of CS accesses. In other words, such a heterogeneity does not change the order of priority of the requests in such a way that request from closer nodes would be satisfied before distant ones. In the above equation, any node can be chosen among N with the same probability, independently of the Grid topology. Furthermore, if theoretical curves were drawn from the equation, they would be similar to the ones of Figure 2. Thus, we can deduce that the assumption of equiprobability is reasonable and that the algorithm does not naturally adapt itself to the Grid topology.

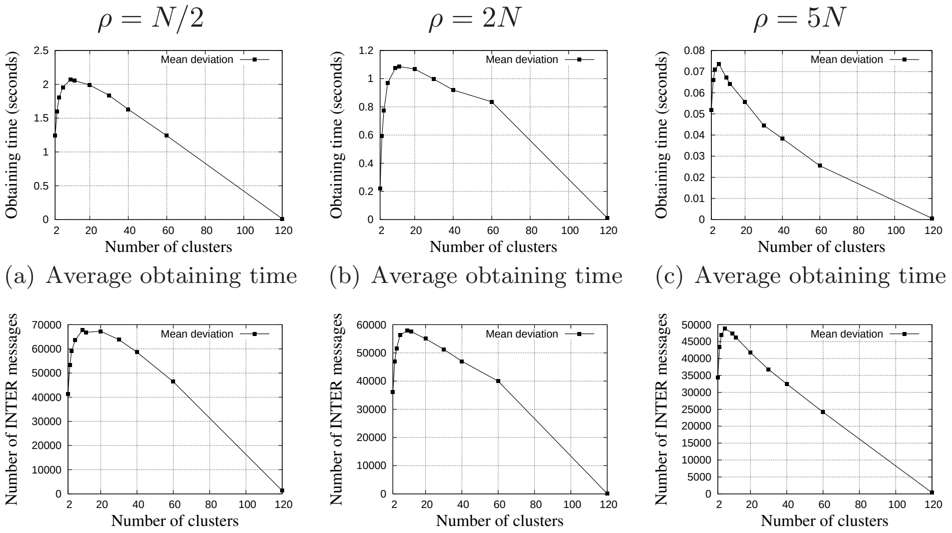
Let's come back to the curves in order to study the impact of the number of clusters with respect to the application behavior. The results of Figures 2(a), 2(b), and 2(c) show that the degree of parallelism of an application has an impact on the *obtaining time*. Furthermore, the curves of Figures 2(d), 2(e), and 2(f) show that the parallelism degree of an application has no influence on the number of *inter* cluster messages even if we observe a small reduction of this number for low parallel applications.

4.2 Hierarchical algorithm

We are now going to study the impact of the Grid architecture on our hierarchical approach. The number of clusters has an influence on the *obtaining time* as well as in the number of *inter* cluster which increase with the number of clusters. However, if we exclude the configuration with one node per cluster where there is in fact no hierarchy of communication at all, our approach always presents a smaller *obtaining time* and number of *inter* cluster messages when compared to the *flat* algorithm. Notice that the benefit of using our composition approach is considerable even for a Grid composed of 60 two-node clusters.

Since the topology of the Grid has not the same impact on our composition approach as on the *flat* algorithm, it would be interesting to study the mean deviation between the *hierarchical* curves and the *flat* ones for both the *obtaining time* and the number of *inter* cluster messages. Thus, based on the curves of Figure 2, Figure 3 shows such mean deviations.

In Figure 3, we can observe that the gain of our composition approach increases when the number of clusters ranges from 2 to 12. This is in accordance with the curves of Figures 2 where the *obtaining time* as well the number of *inter* cluster messages increase sharply for the original algorithm but smoothly for our composition approach. Such a different behavior explains why the maximum mean deviation between the two curves is reached with 12 clusters. Beyond this threshold value, the clustering effect neither has an influence on the *obtaining time* nor on the number of *inter* cluster messages since in our *hierarchical* approach the curves progressively increase while in the curves of the *flat* algorithm remain linear. Thus, the respective mean deviations inversely decrease until they become null for the configuration where each node represents a cluster (120 clusters).



(a) Average obtaining time (b) Average obtaining time (c) Average obtaining time

(d) Number of *inter* messages (e) Number of *inter* messages (f) Number of *inter* messages

Fig. 3. Mean deviation between the *hierachical* and *flat* algorithms

We would like to theoretically evaluate the above threshold in a Grid composed of N nodes uniformly divided into c clusters. Hence, similarly to section 4.1, we need to find the probability $\mathcal{P}$ that a node sends an *inter* cluster message in our own hierarchical approach on top of such a Grid. Without loss of generality, we consider the case where the cluster locality is maximum, i.e., every time a coordinator of a cluster gets the *inter* token, all the N/c nodes of this cluster execute a critical section which corresponds to a low parallel application. Thus, the probability $\mathcal{P}$ is equal to the probability of executing the last of the N/c critical section executions:

$$\mathcal{P} = \frac{1}{\frac{N}{c}} = \frac{c}{N}$$

Therefore, the mean deviation $E(c)$ between our composition approach and the *flat* algorithm in function of the number of clusters c is equal to:

$$E(c) = 1 - \frac{1}{c} - \frac{c}{N}$$

and according to the derivative of E, the mentioned threshold, $c_{threshold}$, is equal to:

$$E'(c) = \frac{1}{c^2} - \frac{1}{N} = 0 \Rightarrow c_{threshold} = \sqrt{N}$$

Such an equation shows that the maximum benefit when using our composition approach is reached for a Grid architecture composed of $\sqrt{N}$ nodes.

This result can be verified by the curves of Figure 3 since $\sqrt{120} = 10.95$. Consequently, for $\rho = N/2$ and $\rho = 2N$, the maximum mean deviation is reached between 8 and 12 clusters. It is also worth noting that for low parallel applications ($\rho = 5N$), the Grid architecture corresponding to the highest benefit is equal to 6 clusters.

Finally, contrarily to the *flat* algorithm, the parallelism degree of an application has an influence on our hierarchical approach. Indeed, we can observe in the curves of Figure 2 that it becomes less effective with higher parallel applications when the number of clusters increases, i.e., it does not present a linear behavior anymore as it does with low parallel applications.

5 Related work

Some works have proposed to adapt existing mutual exclusion algorithms to a hierarchical architecture. In [7], the author presents an extension to Naimi-Tréhel's algorithm, introducing the concept of priority. A token request is associated with a priority and the algorithm first satisfies the requests with the higher priority. Bertier and al. [1] adopt a similar strategy based on the Naimi-Tréhel's algorithm which treats intra-cluster requests before inter-cluster ones.

Several authors have propose hierarchical approaches for combining different mutual exclusion algorithms. Housni and al. [4] and Chang and al. [2] mutual exclusion algorithms gather nodes into groups. Both consider a hybrid approach where the algorithm for intra-group requests is different from the inter-group one. In Housni and al. [4], sites with the same priority are gathered at the same group. Raymond's tree-based token algorithm [10] is used inside a group, while Ricart-Agrawala [11] diffusion-based algorithm is used between groups. Chang and al. [2] algorithm applies diffusion-based algorithms at both levels: Singhal's algorithm [13] locally, and Maekawa's algorithm [6] between groups. The former uses a dynamic information structure while the latter is based on a voting approach. Similarly, Omara et al. [9]'s solution is a hybrid of Maekawa's algorithm and Singhal's modified algorithm which provides fairness. Erciyes [3] proposes an approach based on a ring of clusters where each node in the ring represents a cluster of nodes. The author then adapts Ricart-Agrawala's algorithm to this architecture.

Our work is close to these hybrid algorithms about gathering machines into groups (clusters in our case) which has in influence on the conception of the algorithm. However, none of the articles present an evaluation study of the impact of the number of groups (or clusters) on the performance of the proposed algorithms.

6 Conclusion

Our evaluation results show that clustering induces an important overhead in the *flat* algorithm but does not cause any side effects, i.e., it does not change the order of critical section accesses. Moreover, the impact of the number of clusters on the *flat* algorithm does not depend on the parallelism degree of the application.

In the case of our *hierarchical* algorithm, the number of clusters has an impact on its performance. However, our approach always presents a shorter *obtaining time* and a smaller number of *inter* cluster messages compared to the *flat* algorithm when the number of nodes per cluster is greater than one even for a Grid composed of a large number of clusters. Contrarily to the *flat* algorithm, the parallelism degree of an application has an influence on our hierarchical approach.

Finally, based both on our evaluation experiments and a theoretical study, we can conclude that the optimal number of clusters that a platform should present in order to provide the highest performance gain for the *hierarchical* algorithm is around $\sqrt{N}$, where N is the total number of nodes on the Grid.

References

1. M. Bertier, L. Arantes, and P. Sens. Distributed mutual exclusion algorithms for grid applications: A hierarchical approach. *JPDC*, 66:128–144, 2006.
2. I. Chang, M. Singhal, and M. Liu. A hybrid approach to mutual exclusion for distributed system. In *IEEE Int. Computer Software and Applications Conf.*, pages 289–294, 1990.
3. K. Erciyes. Distributed mutual exclusion algorithms on a ring of clusters. In *Int. Conf. on Computational Science and Its Applications*, volume 3045 of *LNCS*, pages 518–527, 2004.
4. A. Housni and M. Tréhel. Distributed mutual exclusion by groups based on token and permission. In *Int. Conf. on Computational Science and Its Applications*, pages 26–29, June 2001.
5. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7):558–564, 1978.
6. M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM-Transactions on Computer Systems*, 3(2):145–159, May 1985.
7. F. Mueller. Prioritized token-based mutual exclusion for distributed systems. In *Int. Parallel Processing Symp.*, pages 791–795, March 1998.
8. M. Naimi, M. Trehel, and A. Arnold. A log (N) distributed mutual exclusion algorithm based on path reversal. *JPDC*, 34(1):1–13, 1996.
9. F. Omara and M. Nabil. A new hybrid algorithm for the mutual exclusion problem in the distributed systems. *Int. Journal of Intelligent Computing and Information Sciences*, 2(2):94–105, 2002.
10. K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.
11. G. Ricart and A. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, 24, 1981.
12. L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.
13. M. Singhal. A dynamic information structure for mutual exclusion algorithm for distributed systems. *Trans. on Parallel and Distributed Systems*, 3(1):121–125, 1992.
14. J. Sopena, F. Legond-Aubry, L. Arantes, and P. Sens. A composition approach to mutual exclusion algorithms for grid applications. In *Int. Conf. on Parallel Processing*, pages 65–75, 2007.
15. I. Suzuki and T. Kasami. A distributed mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 3(4):344–349, 1985.