Simulation Implementation of Centralized Algorithm Mutual Exclusion in Distributed Systems

Qian Li
Changchun Guanghua University, Changchun, Jilin, China, 130033

ABSTRACT: In this paper, the author analyses and studies the problems of distributed synchronization and mutual exclusion, and improves a token-based algorithm—Goscinski algorithm after evaluating a number of existing distributed mutual exclusion algorithm. In order to prove that the improved algorithm has better performance, the author in this paper carries out a detailed analysis and calculation of the performance of the improved algorithm, and compares it with Goscinski algorithm and other algorithms.

1 INTRODUCTION

Mutual exclusion aims to solve the control of each concurrent process’s access to critical resources in the single-machine system. With the development of computer technology, mutual exclusion problem extends from the single-machine system to the distributed systems. For example, in Web applications, multiple Web servers in response to different requests may need to modify the same file on the same server at the same time, and each Web server must synchronize the mutual exclusion modification of the file to ensure the accuracy and completeness of the file content; another example, in large-scale distributed parallel computing, concurrent processes distributed in different machines inevitably needs constant interaction of intermediate result data, and if there is no mutual exclusion mechanism, the correct modification of the intermediate results cannot be guaranteed, which may leads to fatal error in the entire calculation.

In order to achieve distributed mutual exclusion, people have proposed a lot of distributed mutual exclusion algorithms, including license-based algorithm and token-based algorithm. License-based algorithm and token-based algorithm use a completely different way to achieve mutual exclusion, each with distinct characteristics. Although license-based algorithm was modified and improved for a lot of times, in general, the average number of messages that need to be changed in license-based algorithm is larger than that in token-based algorithm. Therefore, this paper focuses on the research and improvement of token-based algorithm.

2 TOKEN-BASED ALGORITHM

The concept of token is introduced to the token-based algorithm. A token actually represents a control point and is transmitted in the form of messages among the system processes in accordance with a certain order, and only the process that receives the message has access to the critical section. A token is unique for each critical resource, i.e. the system assigns only one token for each critical resource. The uniqueness of the token determines that at any time, only one process has access to the critical resource, which ensures mutual exclusion access of each process to the critical resource.

2.1 TokenRing Algorithm

In TokenRing algorithm, all the processes form a ring structure logically among which the token circulates. Each process in the ring has the only predecessor process and the only successor process. When the token moves in the ring to a certain process and is received by this process, if this process wishes to enter the critical section, it will keep the token and enters the critical section. Once it exits the critical section, it will transfer the token to the successor process; if the process that receives the token does not wish to enter the critical section, it will directly transfer the token to its successor process. Since there is only one token in the logic ring, TokenRing algorithm achieves mutual exclusion.

2.2 Ramond Algorithm

In Ramond algorithm, all nodes in the system are organized into a logical tree structure in which the connection line between the parent node and the child node is directed, and is always directed from the child node to the parent node.
2.3 Goscinski Algorithm

Goscinski algorithm is an algorithm that has no requirement on the system’s logical structure and support the system with a priority. Its core idea is as follows: 1) in the system there is only one token created by a process at the initialization time of the system; 2) when the process requires access to the critical resource, it broadcasts the request message to all other processes and at the same time informs them of its priority; 3) each process maintains a request queue P, and the process holding the token puts the request message it receives into the request queue P in descending order by priority, and the other processes do nothing when they receive the request message; 4) a request queue Q is maintained in the token, and the process holding the token adds the request in request queue P to the request queue Q in the token at the initialization time of the system or when it exits the critical section and sends the token to the sender of the first request in Q, and finally clears request queue P and eliminates the first request in the request queue Q.

2.4 Comparison of These Three Algorithms

In TokenRing algorithm, the process will not send any request message when it wishes to enter the critical section, but just waits for the arrival of the token; in centralized algorithm, the process will take the initiative send a request message to the management process when it wishes to enter the critical section, and then waits for the arrival of the token the transfer of which is determined by the management process; in Goscinski algorithm, the process will initiatively sends a request message to the holder of the token when it wishes to enter the critical zone, and then waits for the arrival of the token and the token movement is determined by the current token holder. In Ramond algorithm, the process initiatively sends a request message to the parent node in the tree structure when it wishes to enter the critical zone, and then waits for the arrival of the token, and the movement of the token is finally determined by the holder of the token, while the parent node is indirectly involved in the determination of the token’s movement in the process of this algorithm.

Among these three algorithms, the advantage of Foscinski algorithm is the short synchronization delay, but because the request message needs to be received and processed by the token holder who is always changing, in order to ensure that the request message can be sent to the token holder, larger number of messages must be exchanged. Therefore, reducing the number of messages in Goscinski algorithm can better achieve synchronization.

3 IMPROVED GOSCINSKI ALGORITHM

3.1 Improving Ideas and Improvement Program

When analyzing Goscinski algorithm, the author first understands that it is a multi-request-reception point algorithm in which the request message of each process is received by all processes but is processed only by the token holder which is always changing, so in order to ensure that the process that wishes to enter the critical section can send the request message to the token holder, this algorithm requires the process to send in the form of broadcast (N-1) messages among which (N-2) messages are not sent to the token holder and become additional messages whose existence reduces the performance of this algorithm. Reducing the number of additional messages can effectively improve the performance of this algorithm.

When re-examine Goscinski algorithm, the author finds that the reason for broadcasting the message is that the request message can only be processed by the token holder that is always changing, and this kind of change is not known by the other processes. Therefore, Goscinski algorithm adopts the method of broadcast of request messages to ensure that the request message can be always received by the token holder.

If the token holder can inform other processes of the broadcast message at each change, other processes will be sure to send the request message to the new token holder instead of sending the request message in the form of broadcast, so the additional messages discussed above do not exist. However, after the above conversions are done, the total number of messages in this algorithm remains the same but not reduced, because each transfer of the token requires a broadcast of message, while the number of transfer of the token is the same as that of generation times of request messages. In this case, in order to reduce the total number of messages, the number of broadcast of messages must be reduced, in other words, the broadcast of message does not occur every time the token holder changes.

Based on the above considerations, the following modifications are made to Goscinski algorithm:

(1) In the system there is an auxiliary process which is fixed. Whenever there is the case of delayed request, a non-management process is bound to receive the request message and after it receives the request message, it forwards it to the auxiliary process:

(2) After receiving the request message, the auxiliary process will store it in the local queue P, and when the token is transferred to the auxiliary process, the auxiliary process will sort all the delayed request messages that it receives and add them to the token’s request queue Q:
(3) In order to ensure that the delayed requests can be added to the request queue Q asap, each time after the management exits the critical section, it will transfer the token to the auxiliary process. After the auxiliary process adds the delayed request message to the token, it will transfer the token to the sender of the first request message in the token’s request queue Q.

3.2 Performance Analysis of the Improved Algorithm

After the above modifications of Gosinski algorithm, the management process and auxiliary process are introduced, so the improved algorithm is more complex than Gosinski algorithm in how to process request messages, which may cause some loss of the performance. Since the main object of this paper is to improve the performance of Gosinski algorithm in three aspects of the average number of messages, synchronization delay, and the response time, this paper will only focus on the average number of messages, synchronization delay, and the response time and will not consider the influence of the complexity of processing request messages on the performance of the algorithm.

3.3 The Average Number of Messages

Suppose the system has N processes.

In the improved algorithm, each request message may need to be forwarded but at most for once, and in the best case, each request message can be received by the management process, while in the worst case, only one request message is received by the management process and the other request messages need to be forwarded.

(1) Consider the average number of messages of the algorithm in the best case.

In a cycle, suppose that when the management process exits the critical section, the number of request message it has collected is R, so from sending the request to entering the critical section, these R processes send a total of R request messages, (N-1) notification messages, and (R+1) token messages, so in n cycles, the average number of messages is:

\[
\bar{M} = \frac{\sum_{i=1}^{n}(R_i + R_i + 1 + N - 1)}{n} = 2 + \frac{\sum_{i=1}^{n} N}{\sum_{i=1}^{n} R_i}
\]

According to (1), \(\bar{M}\) decreases with the increase of \(R_i\).

Because \(1 \leq R_i \leq N - 1\), So

\[
2 + \frac{\sum_{i=1}^{n} N}{\sum_{i=1}^{n} (N - 1)} \leq \bar{M} \leq 2 + \frac{\sum_{i=1}^{n} N}{\sum_{i=1}^{n} 1}
\]

That is

When \(1 \leq R_i \leq N - 1\),

And

\[
3 + \frac{1}{N - 1} \leq \bar{M} \leq 2 + N
\]

When \(1 \leq R_i \leq N - 1\),

\[
2 + \frac{\sum_{i=1}^{n} N}{\sum_{i=1}^{n} (N - 1)} \leq \bar{M} \leq 2 + \frac{\sum_{i=1}^{n} N}{\sum_{i=1}^{n} 1}
\]

And

\[
3 + \frac{1}{N - 1} \leq \bar{M} \leq 2 + N
\]

(2) Consider the average number of messages of the algorithm in the worst case.

Suppose the number of request message in any given cycle is R, (R-1) request messages are forwarded, and from R processes sending request messages to entering the critical section, a total number of R request messages, (N-1) notification messages, and (R+1) token messages, as well as (R-1) forwarded messages are sent, so in n cycles, the average number of messages is:

1) The average number of message integrating both the best case and the worst case: in the best case, the average number of messages is

\[
\frac{1}{N - 1} \leq \bar{M} \leq 2 + N
\]

and in the worst case, the average number of message is

\[
4 \leq \bar{M} \leq 2 + N
\]

Therefore, the above cases integrated, the average number of messages in the algorithm is

\[
3 + \frac{1}{N - 1} \leq \bar{M} \leq 2 + N
\]

In the case of low request load, namely the request load is 2, \(R_i = 1\), the maximum of \(\bar{M}\) is 2+N, and when the request load is gradually increased, \(\bar{M}\) decreases gradually and approaches 3, and the higher the request load, the smaller \(\bar{M}\) is.

Compared with Gosinski algorithm, when N>4, and the request load is greater than 2, with or without request messages requiring to be forwarded, the average number of messages of the improved algorithm is smaller than that of Gosinski algorithm, and the gap increases with the increase of the request load.

3.3.1 Synchronization Delay

Since synchronization delay time is determined by the length of the connection between nodes and the size of the network transmission rate which depend on the actual network circumstances, the analysis of synchronization delay time should be based on the final experimental data collection. The author should have conducted experiments and collect the experimental data related to synchronization delay time and compare it of the improved algorithm with that of Gosinski algorithm, but due to environmental constraints, the author can only rely on a multi-
threaded simulation of distributed mutual exclusion on the computer.

In Token-based algorithm, during the time from one process exiting the critical section to the next process entering the critical section, the number of token transfer determines the length of synchronization delay time, so in this section, synchronization delay time is measured by the number of token transfer times during the time from one process exiting the critical section to the next process entering the critical section.

In the improved algorithm, every time a process exits the critical section, the token is directly transferred to the sender of the first request in token queue Q, and when the sender of the first request in token queue Q receives the token, it enters the critical section, so synchronization delay time in the improved algorithm is the transfer time of the token, which is the same as Gosciniski algorithm.

3.3.2 Response Time
Since the response time is determined by a variety of factors such as the network status, the length of time that processes remain in the critical section, etc., it is difficult to be analyzed and calculated and needs statistical analysis through experiment under specific conditions. In the experiments, data related to the response time should be collected through the comparison of which the response time by the improved algorithm is compared with that by Gosciniski algorithm.

3.3.3 Comprehensive Comparison
The performance comparison among the improved algorithm, Gosciniski algorithm and other algorithms is shown in Table 1. According to the comparison of performance data in Table 1, the performance of improved algorithm is better.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>the average number of messages</th>
<th>the number of synchronization delay messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>the improved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>algorithm</td>
<td>(3+ ( \frac{1}{\sqrt{N-1}} ))((2+N) )</td>
<td>1</td>
</tr>
<tr>
<td>Gosciniski</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TokenRin</td>
<td>( \frac{1}{2} )</td>
<td>0 \sim (N-1)</td>
</tr>
<tr>
<td>algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ramond</td>
<td>O(logN)</td>
<td>0 \sim (N-1)</td>
</tr>
</tbody>
</table>

3.4 Simulation Implementation of Improved Gosciniski Algorithm
The improved Gosciniski algorithm simulates the multi-thread implementation of distributed mutual exclusion simulation experiments main interface on which test data are input, as shown in Fig.1, and the results after running are shown in Fig. 2.

Figure 1. Improved Gosciniski algorithm experimental procedure figure.

Figure 2. Improved Gosciniski algorithm experimental results figure.

4 CONCLUSION
In this paper, the author conducts in-depth studies and discussion of several distributed mutual exclusion algorithms, after analysis and comparison of the advantages and disadvantages of these algorithms, focuses on and improves a token-based algorithm --- Gosciniski algorithm. The improved algorithm can not only synchronize access to the critical section of each node in the computer network without requirements of logical structure through sending messages and transferring the token to ensure the consistency and integrity of critical resources, but also support access to critical resources of each node in priority order. The average number of the improved algorithm is maintained in the range of \( 3+ \frac{1}{\sqrt{N-1}} \) \sim 2+N, and is better than Gosciniski algorithm in most cases. Meanwhile, unlike other algorithms such as Remond algorithm and other ones, the improved algorithm has no special restraints and requirements on the network logical structure, and is more flexible and applicable.

This paper not only provides a detailed description of the improved algorithm, but gives a detailed description of the specific improvement ideas and processes, making it easier to understand. After the detailed description of the improve algorithm, the paper provides the correctness proof, detailed analysis and evaluation of the performance of the improved algorithm. In order to verify the correctness
of the performance analysis of the improved algorithm, comparative experiments are conducted to prove that the improved algorithm has better performance, and the performance deriving from the experiments is consistent with that from the analysis and evaluation.

REFERENCES