ROUCAIROL AND CARVALHO OPTIMIZATION IS PRONE TO DEADLOCK

Rajeev Ranjan Kumar Tripathi[1], Ankan Srivastava[2], Shivam Kumar Pandey[3]

[1] Assistant Professor, Department of CSE, Buddha Institute of Technology, GIDA, Gorakhpur (India)

[2,3] Student, Department of CSE, Buddha Institute of Technology, GIDA, Gorakhpur (India)

ABSTRACT

Critical section problem is a well known problem in Computer Science. It arises when multiple processes or threads simultaneously try to access shared resources like physical devices or logical objects. In Conventional Operating System Design, we use semaphores to solve this one. In Distributed System due to absence of shared memory we cannot implement the same solution. Various approaches are given to solve the critical section problem in Distributed System. An algorithm that solves critical section problem should have following properties like fairness along with deadlock freedom, freedom from starvation and fault tolerant. Ricart and Agrawala suggested message based approach to get mutual exclusion in Distributed System. This approach fairly deals with the critical section problem. Roucairol and Carvalho suggested an optimization for the given approach. This paper is shedding lights on the proposed optimization. This paper finally concludes that though the optimization has a better performance over the original one yet this one is compromising with the fairness.

Keywords— Critical Section, Distributed System, Fairness, Deadlock, Starvation, Fault Tolerance.

I. INTRODUCTION

Distributed System is a collection of inter connected nodes that work together to get a specific goal. Each node has some pre defined roles. Nodes communicate with each other only by message passing. Distributed System has inherent limitations of 1) Shared Memory & 2) Global Clock [1, 2, 3]. In an application, nodes may be in different time zones. It is almost impossible to synchronize the clocks associated with the individual nodes due to clock drift rate. Again if processes are running at different machines we cannot terminate them in the same manner as we do in a standalone machine. A resource may be shared in two different ways 1) Read Only Mode & 2) Exclusive Mode. Note that if sharing is in read only mode, simultaneously many processes, threads and users can access this one. This sharing does not cause the critical section. Critical section arises when sharing is in exclusive mode. Algorithms designated for the mutual algorithms basically generates schedule when multiple requests reach to access the critical section [1, 2, 3]. We can broadly categorise the algorithms given to get mutual exclusion in Distributed System in two different ways [1]:

A. Message Based

B. Token Based

In message based algorithms, site interested to execute critical section sends REQUEST messages to all other participants. On reception of this message, participants cannot send the REPLY message to the sender if they have already sent REPLY message to a site which is still executing the critical section. When a site exists from the critical section it sends RELEASE message. The RELEASE message works as an acknowledgement. If participants have received the RELEASE message from the site to which REPLY message was sent, REPLY message to the new REQUEST is sent. Lamport's Algorithm, Ricart-Agrawala Algorithm and Maekawa Algorithm come under this category. In token based algorithm, a TOKEN is available in the system. A site can enter into the critical section if it has the TOKEN. If a site wants to execute the critical section and it has not TOKEN, the site broadcasts REQUESTS to the other sites. If the site having the TOKEN is not currently executing the critical section, sends back the TOKEN. If it is already executing the critical section, received REQUEST is kept in waiting state in a queue. Suzuki-Kasami Broadcast Algorithm, Raymond Tree Based Algorithm and Singhal Heuristic Algorithm come under this category. Generally we measure the algorithm's performance on two parameters: time and space. In this scenario algorithm's efficiency is measured in terms of messages required to invoke critical section at a time. In next section this paper is describing the essential properties required by the mutual exclusion algorithms.

II. REQUIREMENTS

Primary requirement for a mutual exclusion algorithm is that only one site can only execute the critical section. No two sites can simultaneously execute the critical section at any cost. Besides this followings are some other requirements [1]:

A. Freedom from Deadlock: Mutual exclusion algorithm should be free from deadlock. In message based algorithm, a site should not wait infinitely for the REPLY message. In token based algorithm, a site should not wait infinitely for the TOKEN.

B. Free from Starvation: A site should not be forced to wait infinitely whether on the other there is a site which is frequently executing the critical section. That is every site should get a chance to execute the critical section.

C. Fairness: Permission for executing the critical section should be always granted in the same manner in which the REQUESTS appear. That is disposal of REQUESTS should be in First Come First Serve (FCFS) basis. When we have to introduce fairness in a system, queue is the most appropriate data structure to use. That's why all the algorithms which are based either on message or on token usage the queue.

D. Fault Tolerance: A mutual exclusion algorithm should work even in presence of any failure.

Lamport was the first who had proposed an algorithm to achieve mutual exclusion in Distributed System. In next section, this paper is exploring the Lamport's approach to get mutual exclusion in Distributed System.

III. LAMPORT'S APRROACH OF MUTUAL EXCLUSION

In 1978, Lamport proposed this concept. Every site maintains a request set. Let there are n sites in a system. The request set of site S_i will contain all other sites. So the size of request set (R_i) of S_i is (n-1). That is in R_i, S_i is absent. Site S_i has a request queue (request_queue$_i$). In request_queue, incoming requests are placed in the same order in which they arrive. Every REQUEST has a timestamp associated with it. For timestamp we have a clock. Every site S_i has a clock C_i. Every REQUEST has two tupples: timestamp and the site identifier. Let two REQUESTS are coming from S_j and S_k as (t_{sj}, j) and (t_{sk}, k). Where t_{sj} is indicating the timestamp associated with this REQUEST and j denotes that this REQUEST is coming from site S_j and the similar interpretation for the second REQUEST. Let these REQUESTS arrive at S_i. In request_queue$_i$, REQUEST of S_j will be at the top if and only if $t_{sj}<t_{sk}$ else REQUEST of S_k will be at the top in request_queue$_i$.

3.1 Algorithm

1. Requesting the critical section

- When a site S_i wants to execute the critical section it sends a REQUEST (t_{si}, i) to all the sites of its request_queue$_{i.}$ The site Si places this REQUEST in its request_queue$_i$. Note that (n-1) REQUESTS are sent by S_i.
- On reception of this REQUEST, the site S_j sends a REPLY message to S_i. The REPLY message is also equipped with the timestamp and the site identifier. The site identifier attached with the REPLY informs the receiver about the originator of this REPLY. Happened before (->) relationship is used to identify that the received REPLY is for in response to the REQUEST made or not.

2. **Executing critical section**

Site S_i enters into the critical section when two conditions meet together:

- S_i has received REPLY messages with timestamp larger than t_{si} from all other sites.
- S_i's request is at the top of request_queue$_i$.

3. Releasing the critical section

- The site S_i when exits from the critical section, it removes its REQUEST from its request_queue$_i$ and sends RELEASE messages to all the sites. Note that release message is also equipped with the timestamp and identifier. This timestamp is again used by the receivers to determine that it is coming in response of the REPLYS that they have earlier sent to the S_i.
- When a site S_j receives the RELEASE message, it deletes the REQUEST of Si from its request_queue$_j$ and sends the REPLY to the site whose REQUEST is in its request_queue$_j$ after the deleted request.

A. **Performance**

For each critical section execution, we have to spend 3(n-1) messages. First (n-1) REQUEST messages are sent, then (n-1) REPLY messages are received and finally (n-1) RELEASE messages are sent again.

B. **Proof of Correctness**

This proof is based on contradiction. Let two sites S_i& S_k are simultaneously executing the critical section.

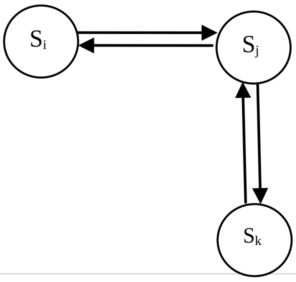

Fig.2 Sites requesting from the permission

The site S_j receives two REQUESTS from S_i and S_k as: (t_{si} ,i) and(t_{sk} ,k). Let S_j sends REPLY to the site S_i, this will be if and only if t_{si}<t_{sk} (as per the algorithm). Let S_j sends REPLY to the site S_k, this will be if and only if t_{sk}<t_{si} (as per the algorithm).These two conditions cannot occur simultaneously. Finally we can conclude that only one site is executing the critical section [6].

To reduce the number of messages required in critical section invocation, we have only two approaches:

- Override a message so that a single message can play role of more than one message.
- Reduce the size of request set.

Above mentioned approaches are optimizations. Ricart-Agrawala approach follows the first criteria while Maekawa approach follows the second. Next section of this paper is describing the Ricart-Agrawala algorithm

IV. RICART-AGRAWALA ALGORITHM

In Ricart-Agrawala algorithm all the assumptions are same except that REPLY message works as both REPLY and RELEASE.

- *The Algorithm*

1. Requesting the critical section:

- When a site S_i wants to enter the critical section, it sends a timestamped REQUEST message to all the sites in its request set.
- When site S_j receives a REQUEST message from site S_i, it sends a REPLY message to site S_i if site S_j is neither requesting nor executing the critical section or if site S_j is requesting and S_i's request's timestamp is smaller than site S_j's own request's timestamp. The request is deferred otherwise.

2. Executing the critical section

- Site S_i enters the critical section after it has received REPLY message from all the sites in its request set.

3. Releasing the critical section

- When site S_i exits the critical section, it sends REPLY message to all the deferred requests.

A site's REPLY messages are blocked only by sites that are requesting the critical section with higher priority (i.e., a smaller timestamp). Thus, when a site sends out REPLY messages to all the deferred requests, the site with the next highest priority request receives the last needed REPLY message and enters the critical section. The execution of critical section requests in this algorithm is always in the order of their timestamp.

4. **Performance**

The Ricart-Agrawala algorithm requires 2(N-1) messages per critical section execution: (N-1) REQUEST and (N-1) REPLY messages [5].

Its correctness proof is same as Lamport's approach.

V. ROUCAIROL AND CARVALHO OPTIMIZATION

Roucairol and Carvalho [4] proposed an improvement to the Ricart-Agrawala algorithm by observing that once a site S_i has received a REPLY message from a site S_j, the authorization implicit in this message remains valid until Si sends a REPLY message to S_j (which happens only after the reception of a REQUEST message from S_j). Therefore, after site S_i has received a REPLY message from site S_j, site S_i can enter its critical section any number of times without requesting permission from site S_j until S_i sends a REPLY message to S_j. With this change, a site in the Ricart-Agrawala algorithm requests permission from a dynamically varying set of sites and requires 0 to 2(N-1) messages per CS execution.

VI. ANALYSIS OF ROUCAIROL AND CARVALHO OPTIMIZATION

Let a site S_i gets permission from all the sites and it enters into the critical section. During execution some more REQUESTS are coming to S_i for the permission. If site S_i is repeatedly executing the critical section all the REQUESTS will be treated as deferred REQUESTS. Requesting sites have to wait infinitely. Though this optimization optimizes the execution of critical section by consuming zero messages in next subsequent invocation, starvation occurs for the other sites.

VII. CONCLUSION

Optimization approach suggested by Roucairol and Carvalho is not fairly good. Once a site gets permission to execute the critical section it remains always in critical section. Ricart-Agrawala approach uses request_queue to introduce fairness. REQUESTS are always processed in the same ways as they arrive. But the proposed optimization tends the Ricart-Agrawala approach to starvation.

REFERENCES

[1] SINGHAL AND SHIVARATRI, "ADVANCED CONCEPTS IN OPERATING SYSTEM," 18TH REPRINT EDITION,TMH.

[2]Coulouris, Dollimore and Kindberg, "Distributed System Concepts and Design," 4th Edition, Pearson Education.

[3] Gerard Tel, "Introduction to Distributed Algorithms," 2nd Edition, Cambridge University Press, 2000.

[4]Carvalho O.S.F. and G. Roucairol, "On Mutual Exclusion in Computer Science, Technical Correspondance," Journal of ACM, 1985.

[5] Ricart , G. and A.K. Agrawal, " An optimal Algorithm for Mutual Exclusion in Computer Networks," Communication of the ACM, Jan. 1981.

[6]Lamport, L, "Time, Clocks and Ordering of Events in Distributed Systems," Communication of the ACM, Jan. 1978.