Performance Comparison of Suzuki Kasami's and Raymond's Tree Token Based Mutual Exclusion Algorithm's

Sagar Panchariya,
sagar.panchariya@gmail.com,
KENT STATE UNIVERSITY, OHIO.

***Abstract*—Suzuki Kasami's and Raymond's Tree are distributed Algorithms that realize mutual exclusion among N nodes in a computer network by usage of a single token. Suzuki Kasami's Algorithm requires 0 or at most N number of messages to enter into critical section. Raymond's Tree Algorithm requires O(Log N) message under light demand and reduced number of messages exchanged per critical section to approximately 4 messages under saturated demand. Suzuki Kasami's Algorithm Operates on a fully connected network, however Raymond's uses a spanning tree of the network.**

Additional Keywords: — critical section, message exchange, delay, privilege message.

I. Introduction

The algorithms are used for N computer network nodes, communicating by message passing rather than shared memory. Message delivery is guaranteed by the communication network however neither the time (state) nor the order of message arrival can be predicted. Nodes may enter critical section out of order. The performance parameters to be measured for a mutual exclusion algorithm other than number of messages are

Synchronization delay(Sd): The time measured between when one site leaves and next one enter.
Response Time: The time interval a site waits its CS execution to be over after request has been sent.
Throughput=1/(Sd + E): Where Sd is the average synchronization delay and E is the average critical section execution time.

Suzuki Kasami's Algorithm:

A node having the token is allowed to enter into the critical section. A single node has the privilege and a node requesting critical section broadcast's a message to all the other nodes. A site sends the privilege if the token is idle with the site. The site having token can continuously enter critical section until it sends the token to some other site. The request message has the format REQUEST(j,n), which means site j is requesting its nth critical section. Each node maintains an array RN of size N for recording latest sequence number received from each of the other nodes. The PRIVILEGE message has the format PRIVILEGE (Q, LN), where Q is queue of nodes requesting critical section and LN is an array of size N where LN[j] is the latest critical section executed by a node j. If RN[j] = LN[j]+1 means a node j has sent a request for its new sequence of critical section, and the node having the privilege adds this to the queue and if token is idle sends the node sends the PRIVILEDGE(LN,Q) to the node requesting critical section. Number of message per Critical section entry is (N-1) REQUEST messages plus 1 PRIVILEGE message so N messages in all or 0 if the node having the token wants to enter critical section.

Raymond's Tree Algorithm:

In this Algorithm nodes are arranged in an un-rooted tree structure. All messages are sent along the undirected edges of the tree. Every node knows about the existence of its immediate neighbors. Again a PRIVILEGE message has to be received by a node to enter into critical section. At every node a variable HOLDER points to a node along the path to the PRIVILEGE. At node having the PRIVILEGE the HOLDER points to itself. When a non-privileged node wants to enter critical section it generates a request and adds it to its REQUESTQ, A REQUESTQ is maintained by each of the nodes. If it has not sent a message along the directed path towards the node pointed by the holder variable, it sends a message along the edge to its holder. On receiving a message the nodes sends it to its holder along path before that the node adds the request in its REQUESTQ. When the request reaches the node having the PRIVILEGE, if the token is idle with the node it sends the PRIVILEGE to the node from which it received the message. On receiving the PRIVILEGE if the nodes own id is top of the queue, it executes critical section else sends the PRIVILEGE to the node pointed by the id, and set its holder to point to that node. The number of messages required to execute critical section can be 0 or typically 2D, where D is the diameter of the tree on which the algorithm is running, however this is reduced to maximum of four messages per critical section execution under full load when the topology is proper tree and two messages when it's a chain.

Experimental Setup:

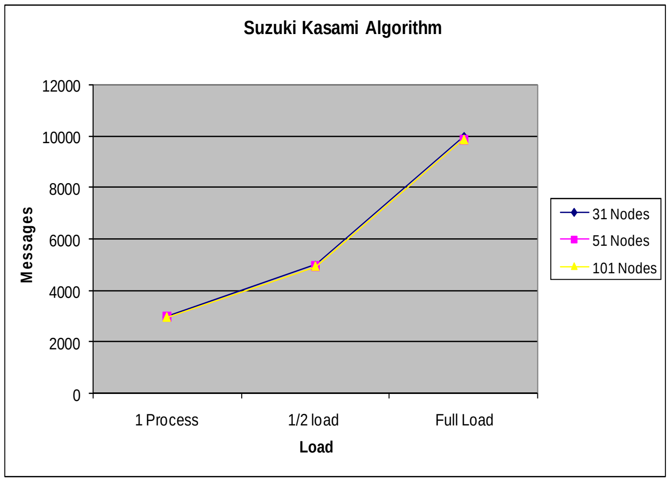
To measure performance of the algorithms as given by the two authors, a experimental setup was made to simulate the distributed nodes. The simulation takes in the number of nodes on which the algorithm has to run. The experiments were carried on number of nodes approx 30, 50 and 100 and the readings were recorded for loads as under 1 node requesting at a time (light load condition), half-load and full load condition. The numbers of messages were recorded for 100 critical executions and message delay was set between 1 to 9 and critical section execution time from 1 to 9 as chosen by the random selector.

More Insight in the Simulation engine

Simulation engine uses three random generators one for selecting node ids, the second one is used to generated state delay for REQUEST or PRIVILEGE type of message. The last one is used to generate the state delay for which a node executes critical section again which is set between 1 to 9.

The experiments starts with initializing all the nodes and giving the PRIVILEGE to one of the node, note the token is idle initially. Next a random selector picks number of requestor id's depending on the load condition this would typically be state 0 in the simulation and the state delay will be calculated between 1-9 units and set as message arrival time for each of the receivers in state queue. Next cycle of selecting the requestors will be any of the states while running the program; it may over with lap any of the message delivery state from the queue. If the ids selected by the random selector had already send request in the previous one and haven't got chance to execute their critical section then those nodes are ignored. When maximum allowed critical sections have been requested, then the random selector for selecting ids is stopped. The simulation engine eventually halts when each of the requesting node gets chance to execute critical section. Each of the nodes receiving the PRIVILEGE is allowed to keep the token busy for 1-9 states again selected by the third type of random generator.

Results for Suzuki Kasami's Algorithm:



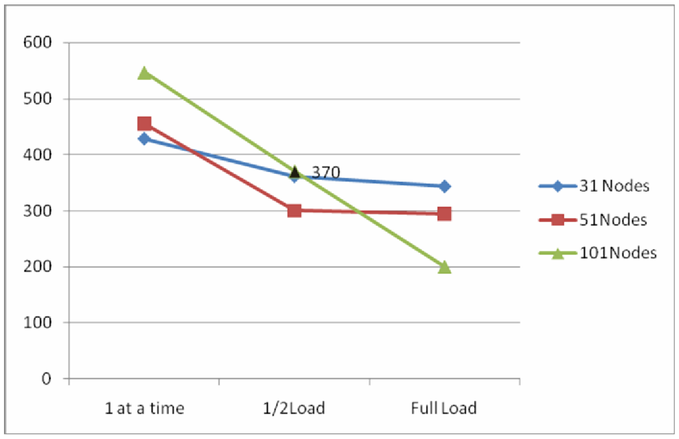
For Suzuki Kasami's algorithms under light weight, the number of messages per critical section was almost N all the time, as it was too rear case that the same node having the token idle (not executing critical section) was requesting so the number of messages per critical section were always N*100 . Only at half load or full load, the number of messages were (N*100)-N, because there was at-least one node which had token idle and was chosen by the random selector for executing critical section. After any node leaves critical section only one message is required for the next requesting node to enter critical section that is the PRIVILEGE message. This reduces the synchronization delay for this algorithm and therefore increases the throughput of the system. However the synchronization delay and throughput were not measured as they averaged to half of the limits set as bounds for random generator when considering the number of messages exchanged in the system.

Results for Raymond's Tree Algorithm:

The experiments were carried out on four different types of tree topologies, a straight line, a star or a tree with depth one, a tree with depth two with approx 30% nodes at level one and approx 70% at the next level, and the last with again 2 depth approx 70% nodes at level one and 30% of nodes at level 2.

In the graphs the focus of observation should be mainly on approx. 100 nodes line as the number of executions completed in the setup is 100.

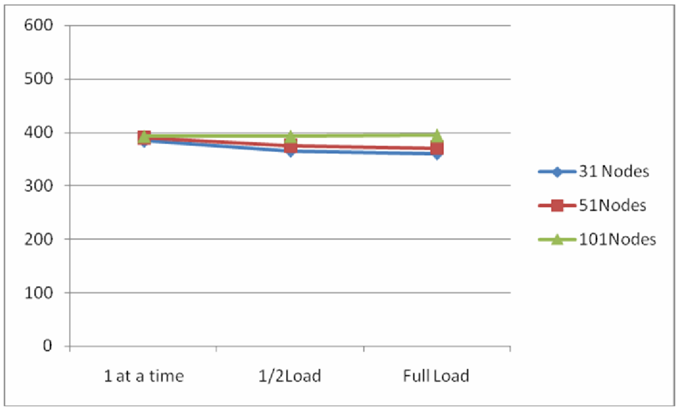
Raymond's Straight line:



For a straight line if number of critical executions is equal to the number of nodes in the chain (in this case approx 100 nodes), then under full-load the maximum number of messages required per critical section entry is 2. The maximum variation of number of messages in noted under half-load with approx 100 nodes requesting 100 critical section this because of the nature of the random selector and simulation engine which may allow fewer number of nodes for requesting critical sections at later states (some of the selected nodes already may have sent request previously), and nodes allowed may be either side of the chain. Hence the token traversal path along the chain is not predictable in this condition.

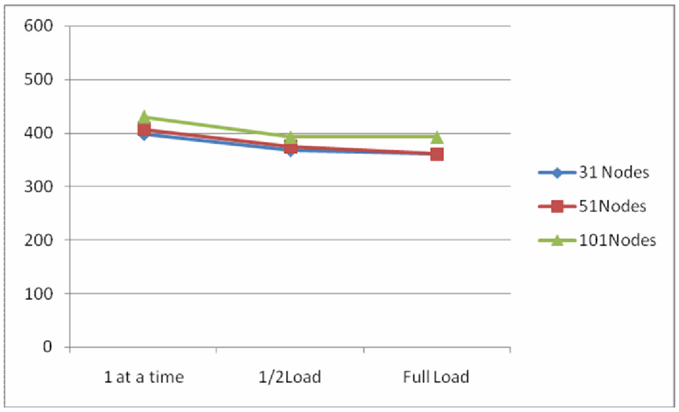
Raymonds Star or a tree with depth one:

Maximum diameter for this topology is 2. Here again focusing on load conditions for approx 100 nodes as number of critcal section is 100 in setup the number of messsge required per critical section execution is approx 4 under full load. The number of messages



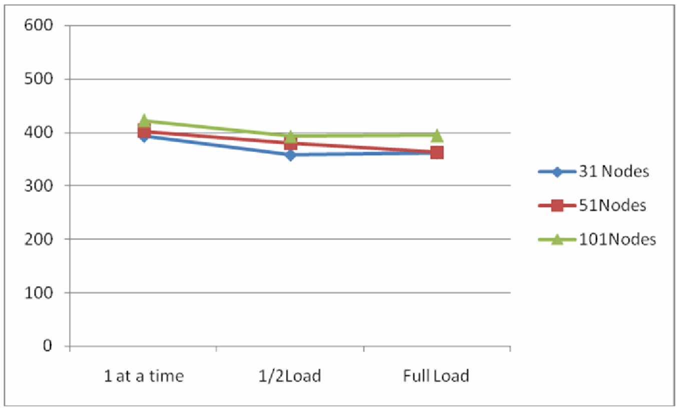
exchanged for approx 30 and 50 nodes cannot be predicted as they are not proportional to the number of executions used as parameter while taking readings and also due to the nature of the random selector and simulation engine.

Raymond's 2 Depth Tree (30%-70%):



Again focusing on full load conditions for approx 100 nodes as number of critcal section is 100 in setup the number of message required per critical section execution is approx 4.

Raymond's 2 depth tree (70%-30%):



Focusing on full load conditions for approx 100 nodes as number of critcal section is 100 in setup the number of messsge required per critical section execution is approx 4.

Conclusion:

A experimental simulation of the Suzuki Kasami' and Raymond's distributed mutual exclusion was made and verified. The results obtained were pretty much a function of the chosen fixed parameters, though they highlighted and verified all the important aspects of the algorithms. Ideally number of messages exchanged in the Suzuki Kasami's Algorithm is greater than that of Raymond, however the synchronization delay and system throughput would be lower at the cost of using a denser broadcast interconnection network among nodes. In Raymond's Algorithm the worst case number of messages exchanged per critical section is when light load and when the topology is like a chain, which increases the diameter of the network used. This kind of network increases number of messages exchanged per critical section and also increases synchronization delay. However this is improved under full load condition. While making a making a choice between either of the algorithms the topology and the load condition should be taken into consideration also tolerance factors like synchronization delay and throughput should be taken into consideration.

Future Work:

Future work will include investigating other mutual exclusion or similar network type problems by implementing more accurate simulation engines for measuring performances of those algorithms.

References

[1] A tree based algorithm for distributed mutual exclusion - Raymond – 1989

[2] Suzuki-Kasami DMX algorithm, Nov 1985

[3] Advanced Operating Systems, Mukesh Singhal chapter 6.