Distributed Algorithms for Multiple Mutual Exclusion based on Maekawa's \(\sqrt{N}\) algorithm

Masaaki Mizuno and Ganesan Sundar
Department of Computing and Information Sciences
Kansas State University
Manhattan, KS 66506

Extended Abstract

Distributed mutual exclusion algorithms can be classified into two categories: assertion based and token based. Among the assertion based algorithms, Maekawa's \(\sqrt{N}\) algorithm requires, in most cases, the minimum number of messages. It requires \(c \times \sqrt{N}\) messages to provide mutual exclusion, where \(N\) is the number of nodes in the network, and \(c\) is a constant between 3 and 7.

We have developed algorithms which expand Maekawa's \(\sqrt{N}\) algorithm to handle multiple mutual exclusion (AND-synchronization) problems, in which each node obtains mutually exclusive access to a set of resources rather than to a single resource. It involves suspending a requesting process until it successfully locks several resources simultaneously. Unlike a sequence of requests for (separate) single mutual exclusion, this approach prevents deadlock. Distributed multiple mutual exclusion algorithms can be used in many applications such as updating a replicated database, or as part of deadlock-free two phase lock protocol in a distributed environment.

In a centralized system, two types of multiple mutual exclusion algorithms exist. One type of primitive (called a PBOTH primitive) prevents starvation, but resource utilization is low. Another type of primitive (called an SP/SV primitive) may cause starvation problems, but resource utilization is high. We have developed both types of distributed algorithms.

The nodes in the system are divided (with overlap) into \(N\) groups: \(S_1, \ldots, S_N\), where (1) \(i \in S_i\), (2) \(S_i \cap S_j = \emptyset\) for any pair of \(i\) and \(j\), and (3) \(|S_i| = \sqrt{N}\), for \(i \leq i, j \leq N\). Constructing the set of \(S_i\)'s is equivalent to finding "a finite projective plane of \(N\) points." The messages exchanged among the nodes are: REQUEST, LOCKED, ACQUIRED, INQUIRE, RELINQUISH and RELEASE messages. When a node \(i\) requests some resources, it sends REQUEST messages to all members of \(S_i\). A LOCKED message is sent by a member of the group to the requesting node. It indicates the member's permission to the requesting node to access the resources. If the node receives LOCKED messages from all of its members, it replies with ACQUIRED messages to the member nodes and is allowed to use the requested resources. Since each node gives permission to access a resource only to one node at a time, property [2] guarantees that no other node could use the same resource.

Each node maintains a queue for each resource. When a node receives a REQUEST message for a set of resources, it places copies of the request into the queues corresponding to the requested resources. In order to avoid deadlock, requests are placed in time stamp order in each queue. In a PBOTH type algorithm, the node sends a LOCKED message to a requesting node when all copies of its request reach the top of the queues. In an SP/SV type algorithm, if some copies of a request reach the top of the queues but others are still in the middle, the copies in the top are temporarily removed from the queues and placed in a special buffer. This is because the request cannot be locked immediately, and temporal removal of these copies from the queues may allow later requests to access these resources.

When the node finishes accessing the resources, it sends RELEASE messages to all of its members. INQUIRE and RELINQUISH messages are used to prevent deadlock. Even though node \(j\) has already granted permission to access resource \(R\) to node \(i\), node \(j\) forces node \(i\) to relinquish the permission if (1) node \(j\) receives another request for resource \(R\) which has a lower time stamp than that of node \(i\)'s request, and (2) node \(i\) has not received LOCKED messages from all of its member nodes. If condition (1) is met at node \(j\), node \(j\) sends an INQUIRE message to node \(i\). If condition (2) is met when node \(i\) receives the INQUIRE message, node \(i\) sends a RELINQUISH message back to node \(j\) and cancels the permission from node \(j\). Upon receipt of the RELINQUISH message, node \(j\) can give permission to the request with higher priority. This mechanism breaks circular waiting; thereby preventing deadlock.

We have proved the correctness of both algorithms with respect to guaranteed mutual exclusion and deadlock freedom. We analyzed the performance of both algorithms in terms of the number of messages required to grant one request. In the best case, four sets of messages are exchanged among \(\sqrt{N}\) nodes: REQUEST, LOCKED, ACQUIRED and RELEASE. Thus, approximately \(4 \times \sqrt{N}\) messages are required per request. In the worst case, seven sets of messages are exchanged: REQUEST, INQUIRE, RELINQUISH, LOCKED (to be relinquished), new LOCKED, ACQUIRED and RELEASE. Thus, at most \(7 \times \sqrt{N}\) messages are required per request.