Quorum-Based Mutual Exclusion Algorithms

Quorum-based mutual exclusion algorithms are different in the following two ways:

1. A site does not request permission from all other sites, but only from a subset of the sites. The request set of sites are chosen such that $\forall i \; \forall j : 1 \leq i, j \leq N :: R_i \cap R_j \neq \Phi$. Consequently, every pair of sites has a site which mediates conflicts between that pair.
2. A site can send out only one REPLY message at any time. A site can send a REPLY message only after it has received a RELEASE message for the previous REPLY message.

continuation..

Since these algorithms are based on the notion of 'Coteries' and 'Quorums', we next describe the idea of coteries and quorums.
A coterie C is defined as a set of sets, where each set $g \in C$ is called a quorum.
The following properties hold for quorums in a coterie:

- **Intersection property:** For every quorum $g, h \in C$, $g \cap h \neq \emptyset$.
  For example, sets {1,2,3}, {2,5,7} and {5,7,9} cannot be quorums in a coterie because the first and third sets do not have a common element.
- **Minimality property:** There should be no quorums g, h in coterie C such that $g \supseteq h$. For example, sets {1,2,3} and {1,3} cannot be quorums in a coterie because the first set is a superset of the second.

continuation..

Coteries and quorums can be used to develop algorithms to ensure mutual exclusion in a distributed environment. A simple protocol works as follows:

- Let 'a' is a site in quorum 'A'. If 'a' wants to invoke mutual exclusion, it requests permission from all sites in its quorum 'A'.
- Every site does the same to invoke mutual exclusion. Due to the Intersection Property, quorum 'A' contains at least one site that is common to the quorum of every other site.
- These common sites send permission to only one site at any time. Thus, mutual exclusion is guaranteed.

Note that the Minimality property ensures efficiency rather than correctness.

Maekawa's Algorithm

Maekawa's algorithm was the first quorum-based mutual exclusion algorithm. The request sets for sites (i.e., quorums) in Maekawa's algorithm are constructed to satisfy the following conditions:

M1: $(\forall i\ \forall j : i \neq j,\ 1 \leq i, j \leq N :: \mathrm{R}_i \cap \mathrm{R}_j \neq \phi)$

M2: $(\forall i : 1 \leq i \leq N :: S_i \in \mathrm{R}_i)$

M3: $(\forall i : 1 \leq i \leq N :: |R_i| = K)$

M4: Any site S_j is contained in K number of R_is, $1 \leq i, j \leq N$.

Maekawa used the theory of projective planes and showed that $N = K(K-1) + 1$. This relation gives $|R_i| = \sqrt{N}$.

continuation..

- Conditions M1 and M2 are necessary for correctness; whereas conditions M3 and M4 provide other desirable features to the algorithm.
- Condition M3 states that the size of the requests sets of all sites must be equal implying that all sites should have to do equal amount of work to invoke mutual exclusion.
- Condition M4 enforces that exactly the same number of sites should request permission from any site implying that all sites have "equal responsibility" in granting permission to other sites.

The Algorithm

A site S_i executes the following steps to execute the CS.

Requesting the critical section

(a) A site S_i requests access to the CS by sending REQUEST(i) messages to all sites in its request set R_i.

(b) When a site S_j receives the REQUEST(i) message, it sends a REPLY(j) message to S_i provided it hasn't sent a REPLY message to a site since its receipt of the last RELEASE message. Otherwise, it queues up the REQUEST(i) for later consideration.

Executing the critical section

(c) Site S_i executes the CS only after it has received a REPLY message from every site in R_i.

The Algorithm

Releasing the critical section

(d) After the execution of the CS is over, site S_i sends a RELEASE(i) message to every site in R_i.

(e) When a site S_j receives a RELEASE(i) message from site S_i, it sends a REPLY message to the next site waiting in the queue and deletes that entry from the queue. If the queue is empty, then the site updates its state to reflect that it has not sent out any REPLY message since the receipt of the last RELEASE message.

Correctness

Theorem: *Maekawa's algorithm achieves mutual exclusion.*
Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j are concurrently executing the CS.
- This means site S_i received a REPLY message from all sites in R_i and concurrently site S_j was able to receive a REPLY message from all sites in R_j.
- If $R_i \cap R_j = \{S_k\}$, then site S_k must have sent REPLY messages to both S_i and S_j concurrently, which is a contradiction. $\square$

Performance

- Since the size of a request set is $\sqrt{N}$, an execution of the CS requires $\sqrt{N}$ REQUEST, $\sqrt{N}$ REPLY, and $\sqrt{N}$ RELEASE messages, resulting in $3\sqrt{N}$ messages per CS execution.
- Synchronization delay in this algorithm is $2T$. This is because after a site S_i exits the CS, it first releases all the sites in R_i and then one of those sites sends a REPLY message to the next site that executes the CS.

Problem of Deadlocks

- Maekawa's algorithm can deadlock because a site is exclusively locked by other sites and requests are not prioritized by their timestamps.
- Assume three sites S_i, S_j, and S_k simultaneously invoke mutual exclusion.
- Suppose $R_i \cap R_j = \{S_{ij}\}$, $R_j \cap R_k = \{S_{jk}\}$, and $R_k \cap R_i = \{S_{ki}\}$.
- Consider the following scenario:
  - S_{ij} has been locked by S_i (forcing S_j to wait at S_{ij}).
  - S_{jk} has been locked by S_j(forcing S_k to wait at S_{jk}).
  - S_{ki} has been locked by S_k(forcing S_i to wait at S_{ki}).
- This state represents a deadlock involving sites S_i, S_j, and S_k.

Handling Deadlocks

- Maekawa's algorithm handles deadlocks by requiring a site to yield a lock if the timestamp of its request is larger than the timestamp of some other request waiting for the same lock.
- A site suspects a deadlock (and initiates message exchanges to resolve it) whenever a higher priority request arrives and waits at a site because the site has sent a REPLY message to a lower priority request.

Deadlock handling requires three types of messages:

FAILED: A FAILED message from site S_i to site S_j indicates that S_i can not grant S_j's request because it has currently granted permission to a site with a higher priority request.

INQUIRE: An INQUIRE message from S_i to S_j indicates that S_i would like to find out from S_j if it has succeeded in locking all the sites in its request set.

YIELD: A YIELD message from site S_i to S_j indicates that S_i is returning the permission to S_j (to yield to a higher priority request at S_j).

Handling Deadlocks

Maekawa's algorithm handles deadlocks as follows:

- When a REQUEST(ts, i) from site S_i blocks at site S_j because S_j has currently granted permission to site S_k, then S_j sends a FAILED(j) message to S_i if S_i's request has lower priority. Otherwise, S_j sends an INQUIRE(j) message to site S_k.
- In response to an INQUIRE(j) message from site S_j, site S_k sends a YIELD(k) message to S_j provided S_k has received a FAILED message from a site in its request set or if it sent a YIELD to any of these sites, but has not received a new GRANT from it.
- In response to a YIELD(k) message from site S_k, site S_j assumes as if it has been released by S_k, places the request of S_k at appropriate location in the request queue, and sends a GRANT(j) to the top request's site in the queue. Maekawa's algorithm requires extra messages to handle deadlocks
- Maximum number of messages required per CS execution in this case is $5\sqrt{N}$.