Distributed mutual exclusion

- DMX definitions
- DMX vs. single-computer MX
- DMX taxonomy
- measuring performance of a DMX algorithm
- trivial DMX algorithm – central coordinator
- non-token-based DMX algorithms
  - Lamport’s
  - Ricart-Agrawala’s
  - Maekawa’s
  - Raymond’s extension to K-exclusion

MX in single-computer vs. DMX

- Single computer MX (also called shared memory DMX)
  - similar to DMX problem - multiple processes compete for CS execution
  - processes have access to shared variables/clock (presumed to be executed on the same machine)
  - solutions usually based on shared-memory based synchronization primitives (locks/condition variables, semaphores)
  - in DMX do not have access to shared memory/clock
  - synchronization has to be done without shared memory
  - delays in propagation of information are unpredictable

Measuring performance

- Metrics to measure performance of DMX algorithms
  - message complexity - number of messages per CS entry
  - synchronization delay - amount of time required after one process leaves CS and another process enters CS - measured in the number of causally related messages
- the measures are considered
- low and high load - the number of processes in the system simultaneously requesting CS
- worst and average case

Distributed mutual exclusion (DMX)

- N processes share a single resource, and require mutually-exclusive access
- conditions to satisfy:
  - safety - only one process can access the shared resource at a time
  - liveness - if a process requests to access the shared resource it should eventually be given a chance to do so
  - the process accessing the shared resource is said to be in critical section (CS), process wishing to access the resource is said to be requesting CS
  - each process may or may not request CS during a computation
  - the CS execution is always finite
- Assumptions made:
  - Messages between two processes are received in the order they are sent (channels are FIFO)
  - Every message is eventually received
  - Each process can send a message to any other process (fully connected network)

DMX algorithms taxonomy

- Lock-based (aka permission based, non-token based) - to enter CS a process needs to obtain permission from other processes in the system.
  - Lamport
  - Ricart-Agrawala
  - Maekawa
- Token-based - unique token (privilege) circulated in the system. A process possessing the token can enter CS
  - Suzuki-Kasami
  - LeLann
  - Raymond

Central Coordinator

- one processor is coordinator – maintains queue of requests
- to enter the critical section, a processor sends a request message to the central coordinator
- when the coordinator receives a request:
  - If no other processor is in the critical section, it sends back a reply message
  - If another processor is in the critical section, the coordinator adds the request to the tail of its queue, and does not respond
- When the requesting processor receives the reply message from the coordinator, it enters the critical section
- When it leaves the critical section, it sends a release message to coordinator
- When the coordinator receives a release message, it removes the request from the head of the queue, and sends a reply message to that processor
- evaluation
  - message complexity 3
  - synchronization delay 2T
### Lamport’s algorithm (1978)
- Each processor maintains a request queue, ordered by timestamp value
- Requesting the critical section (CS):
  - When a processor wants to enter the CS, it:
    - Adds the request to its own request queue - requests are ordered by timestamps
    - Sends a timestamped request to all processors
  - When a processor receives a request message, it:
    - Adds the request to its own request queue
    - Returns a reply message
- Executing the CS:
  - A process enters the CS when both:
    - Its own request is at the top of its own request queue (its request is earliest)
    - variant 1 – it has received a reply with greater timestamp form every process in the system
  - variant 2 – it has received any message with greater timestamp

### Lamport’s algorithm (cont.)
- Releasing the CS:
  - When a processor leaves the CS, it:
    - Removes its own (satisfied) request from the top of its own request queue
    - Sends a timestamped release message to all processors in the system
  - When a processor receives a release message, it:
    - Removes the (satisfied) request from its own request queue
    - (Perhaps raising its own message to the top of the queue, enabling it to finally enter the CS)
- Evaluation:
  - message complexity - $3(N-1)$
  - $(N-1)$ release, $(N-1)$ request, $(N-1)$ reply
  - synchronization delay - $T$

---

### Ricart and Agrawala’s algorithm (1981)
- optimization of Lamport’s – no releases (merged with replies)
- Requesting the critical section (CS):
  - When a processor wants to enter the CS, it:
    - Sends a timestamped request to all OTHER processors
  - When a processor receives a request:
    - If it is neither requesting nor executing the CS, it returns a reply (not timestamped)
    - If it is requesting the CS, but the timestamp on the incoming request is smaller than the timestamp on its own request, it returns a reply
    - Means the other processor requested first
    - Otherwise, it defers answering the request
- Executing the CS:
  - A process enters the CS when:
    - It has received a reply from all other processors in the system

### Ricart and Agrawala’s algorithm (cont.)
- Releasing the CS:
  - When a process leaves the CS, it:
    - Sends a reply message to all the deferred requests
    - (process with next earliest request will now received its last reply message and enter the CS)
- Evaluation:
  - message complexity - $2(N-1)$
  - $(N-1)$ reply, $(N-1)$ request
  - synchronization delay – $T$

---

### Maekawa’s algorithm
- Lamport’s and Ricart-Agrawala’s have message complexity proportional to the number of processes in the system
- observation - a process does not have to send message to all other processes to lock them
- every process $Pi$ is assigned a request set $Ri$, (quorum) of processes
  - $P_i$ is in $R_i$
  - for any two processes $P_i$ and $P_j$, $R_i \cap R_j = \emptyset$
- Maekawa showed that minimum quorum size is $\lceil \sqrt{N}\rceil$
- example quorums:
  - for 3 processes: $R_0=\{P_1,P_2\}, R_1=\{P_2,P_3\}, R_2=\{P_3,P_1\}$
  - for 7 processes: $R_0=\{P_1,P_2,P_3\}, R_1=\{P_2,P_3,P_4\}, R_2=\{P_3,P_4,P_5\}, R_3=\{P_4,P_5,P_6\}, R_4=\{P_5,P_6,P_7\}$

### Maekawa’s algorithm, Basic operation
- Requesting CS
  - process requests CS by sending timestamped request message to processes in its quorum
  - a process has just one permission to give, if a process receives a request it sends back reply unless it granted permission to other process; in which case the request is queued
- Entering CS
  - process may enter CS when it receives replys from all processes in its quorum
- Releasing CS
  - after exiting CS process sends release to every process in its quorum
  - when a process gets release it sends reply to the lowest timestamped request in its queue
Maekawa's algorithm, deadlock possibility

- Since processes do not communicate with all other processes in the system, CS requests may be granted out of timestamp order
- example:
  - suppose there are processes $P_i$, $P_j$, and $P_k$ such that:
    - $P_j \in R_i$ and $P_j \in R_k$ but $P_j \notin R_i$ and $P_j \notin R_k$.
    - $P_i$ and $P_j$ request CS such that $t_i < t_j$.
    - if request $P_i$ from reaches $P_j$ first, then $P_j$ sends reply to $P_i$, and $P_i$ has to wait for $P_j$ out of timestamp order.
  - a wait-for cycle (hence a deadlock) may be formed.

Maekawa's algorithm, deadlock avoidance

- To avoid deadlock process recalls permission if it is granted out of timestamp order.
  - if $P_i$ receives a request from $P_j$ with higher timestamp than the request granted permission, $P_i$ sends failed to $P_j$.
  - if $P_j$ receives a request from $P_i$ with lower timestamp than the request granted permission (deadlock possibility), $P_j$ sends inquire to $P_i$.
  - when $P_j$ receives inquire it replies with yield if it did not succeed getting permissions from other processes.
    - either got failed or sent a yield and did not get reply.

Raymond's extension for sharing K identical resources (1987)

- $K$ identical resources, which must be shared among $N$ processes.
- Raymond's extension to Ricart-Agrawala's algorithm:
  - A process can enter the CS as soon as it has received $N-K$ reply messages.
  - Algorithm is generally the same as R&A, with one difference:
    - R&A — reply messages arrive only when process is waiting to enter CS
    - Raymond —
      - $N-K$ reply messages arrive when process is waiting to enter CS
      - Remaining $K-1$ reply messages can arrive when process is in the CS, after it leaves the CS, or when it's waiting to enter the CS again.
      - Must keep a count of number of outstanding reply messages, and not count those toward next set of replies.
- how would you modify Maekawa's to share $K$ resources?