Corso di Laurea in Computer Science

Algoritmi di mutua esclusione per i sistemi distribuiti

Mattia Prestifilippo Colombrino

ANNO ACCADEMICO 2025/2026

Premesse

- I nodi non condividono memoria. Comunicano tramite scambio di messaggi.
- Non esiste un clock globale.
- I messaggi possono arrivare dopo un ritardo imprevedibile
- **Safety:** Solo un nodo alla volta può trovarsi nella sezione critica.
- **Liveness:** Ogni nodo che richiede di entrare nella sezione critica prima o poi potrà farlo.

Token based: Suzuki Kasami Algorithm

- L’accesso alla sezione critica è regolamentato da un **token**. Il nodo che detiene il token **puo entrare** nella sezione critica.
- I nodi in attesa vengono inseriti in una **coda di attesa**. Al termine della CS, il nodo estrae il nodo in testa e gli invia il token.
- **PRIVILEGE:** Messaggio che trasferisce il diritto di entrare nella sezione critica.
- **REQUEST(idNode, reqN):** Messaggio che richiede a tutti gli altri nodi il token.

Token based: Suzuki Kasami Algorithm

Stato del nodo

- **RequestNumber_I[]:** Array che per ogni entry j memorizza il numero di richiesta piu alto ricevuto dal nodo j.
- **HavePrivilege:** Booleano che indica se il nodo possiede il token
- **Requesting:** Booleano che indica se il nodo sta richiedendo il token.

Stato del token

- **LastRequestNumber[]:** Array che per ogni entry j memorizza l'ultimo numero di richiesta di j per cui il token è stato concesso
- **Coda Q:** Coda FIFO che contiene gli ID dei nodi in attesa del token

Token based: Suzuki Kasami Algorithm

```
PROCEDURE: REQUEST_CS()    // called when this node wants to enter critical section
```

```
Requesting ← true

if HavePrivilege == false then
    // generate a new request number for myself
    RN[id] ← RN[id] + 1

    // broadcast request to all other nodes
    for each node j in {1..N} \ {id} do
        send REQUEST(id, RN[id]) to j
    end for

    // wait until the token arrives (it carries Q and LN)
    wait until receive PRIVILEGE(Q, LN)
    HavePrivilege ← true
end if
```

Token based: Suzuki Kasami Algorithm

```
// now we have exclusive access
CRITICAL_SECTION()

// mark my request as served in the token
LN[id] ← RN[id]

// update the token queue with any pending requests we know about
for each node j in {1..N} \ {id} do
    // j is requesting iff it has exactly one more request than last granted
    if (RN[j] == LN[j] + 1) and (j not in Q) then
        Q.enqueue(j)
    end if
end for

// pass token to next waiting node, if any
if Q is not empty then
    next ← Q.dequeue()
    HavePrivilege ← false
    send PRIVILEGE(Q, LN) to next
end if

Requesting ← false
```

Token based: Suzuki Kasami Algorithm

```
HANDLER: ON_RECEIVE_REQUEST(j, n)   // atomic / indivisible
```

```
// remember the most recent request number from j
RN[j] ← max(RN[j], n)

// if I hold the token, I can immediately transfer the token to j
if HavePrivilege == true
    and Requesting == false
    and (RN[j] == LN[j] + 1) then

     HavePrivilege ← false
     send PRIVILEGE(Q, LN) to j
end if
```

Token based: Suzuki Kasami Algorithm

- **Mutua esclusione**
- **Starvation free**
- **Deadlock free**
- **Complessità: N-1** REQUEST + **1** PRIVILEGE = **N messaggi totali**
- **Problemi**: Se il token viene perso, nessuno può piu entrare nella sezione critica.
- Il token dopo un certo timeout deve essere generato in modo sicuro, evitando che due token girino contemporaneamente (ritardo).

No Token Approach: Ricart Agrawala Algorithm

- Se un nodo vuole **entrare** nella CS, invia a tutti gli altri nodi una **REQUEST.**
- **Entra** nella **CS** quando riceve da tutti una **REPLY**.
- Un nodo può **ritardare** una REPLY verso un altro nodo se richiedendo la CS con **sequence number** minore, o se **uguale** ma ha ID nodo minore.

No Token Approach: Ricart Agrawala Algorithm

- **OurSequenceNumber**: Numero di richiesta generata dal nodo
- **Outstanding:** Numero di **REPLY** attese per entrare nella CS.
- **RequestingCS:** Booleano che indica se il nodo sta richiedendo la CS.
- **ReplyDeferred[]:** Array di booleani che per ogni entry j indica se il nodo ha deferito la REPLY al nodo j.

No Token Approach: Ricart Agrawala Algorithm

```
PROCEDURE: ENTER_CS()     // invoked by this node when it wants the critical section
```

```
lock(SharedVars)

  // I start requesting the critical section
  RequestingCS ← true

  // choose a new timestamp (sequence number), strictly greater than any seen
  OurSeq ← HighestSeq + 1

unlock(SharedVars)

// I need permission from every other node
Outstanding ← N - 1

// broadcast REQUEST to all other nodes
for each node j in {1..N} where j ≠ me do
    send REQUEST(OurSeq, me) to j
end for
```

No Token Approach: Ricart Agrawala Algorithm

```
// wait until every other node has granted permission
wait until Outstanding == 0

// ── Critical Section starts here ──
CRITICAL_SECTION()
// ── Critical Section ends here ──

// leaving CS: stop requesting and flush all deferred replies
RequestingCS ← false

for each node j in {1..N} where j ≠ me do
    if ReplyDeferred[j] == true then
        ReplyDeferred[j] ← false
        send REPLY to j
    end if
end for
```

No Token Approach: Ricart Agrawala Algorithm

```
HANDLER: ON_RECEIVE_REQUEST(k, j)    // request from node j with sequence number k
```

```
// update global max sequence number seen at this node
HighestSeq ← max(HighestSeq, k)

// decide if I must defer reply to j
lock(SharedVars)
    boolean defer =
        RequestingCS AND ( (k > OurSeq) OR (k == OurSeq AND j > me) )

unlock(SharedVars)
if defer then
    // remember to reply later (after I exit CS)
    ReplyDeferred[j] ← true
else
    // grant permission immediately
    send REPLY to j
end if
```

No Token Approach: Ricart Agrawala Algorithm

```
HANDLER: ON_RECEIVE_REPLY()          // a REPLY arrived from some node
```

```
// one fewer permission needed to enter CS
Outstanding ← Outstanding - 1
```

No Token Approach: Ricart Agrawala Algorithm

- **Mutua Esclusione Garantita**
- **No Deadlock**
- **No Starvation**
- **Complessità:** (**N-1)** REQUEST + (**N-1)** REPLY = **2**(**N-1)** TOTALI
- **Problemi:** Se un messaggio viene perso, il nodo richiedente rimane in attesa per sempre.
- Dopo un certo timeout un nodo che non risponde viene contattato. Se non risponde, viene dato per guasto e si considera la reply arrivata.

Quorum approach: Maekawa Algorithm

- **Quorum:** Sottoinsieme di nodi a cui un nodo deve chiedere il permesso per entrare nella CS.
- Qualsiasi coppia di quorum ha almeno un membro in **comune.** Tale nodo può dare nello stesso momento il permesso **solo ad uno** dei due quorum.
- **Tutti** i nodi del quorum devono dare il permesso per entrare nella CS

Es: Sistema con 5 processi: P1, P2, P3, P4, P5.
Quorum(P1) = {**P1**, **P2**, P3}
Quorum(P2) = {**P2**, **P4**, P5}
Quorum(P3) = {**P1**, P3, **P4**}
Se P2 e P3 vogliono accedere insieme alla CS, e hanno un processo in comune (P4), P4 darà il permesso solo a uno dei due.

Quorum approach: Maekawa Algorithm

- Per richiedere l'accesso alla CS, il nodo i invia una **REQUEST** al proprio quorum Si.
- Quando un nodo membro del quorum Si riceve una REQUEST, se non è già bloccato per un'altra richiesta, si marca come LOCKED.
- Restituisce **LOCKED** al nodo richiedente.

Quorum approach: Maekawa Algorithm

- Se gia bloccato, il nodo verifica se la richiesta arrivata preceda la richiesta gia bloccante o qualsiasi altra in coda.
- Se no, viene inviato FAILED al nodo i.
- Se è la più precedente, il nodo invia al nodo bloccante **INQUIRE.**
- **INQUIRE**: Chiede al nodo bloccante se sia riuscito a bloccare tutti i membri del proprio quorum.

Quorum approach: Maekawa Algorithm

- Se il nodo bloccante ha ricevuto qualche FAILED e sa che non riuscirà a bloccare tutti i membri del proprio quorum, risponde **RELINQUISH.**
- In tal caso, il nodo bloccato si libera, lo inserisce nella waiting queue e si blocca verso il nodo i.
- Quando il nodo i riesce a bloccare tutti i nodi del suo quorum entra nella CS. Restituisce poi RELEASE.
- I nodi del quorum si liberano dal blocco e si bloccano per il nodo con più priorità nella propria coda.

Quorum approach: Maekawa Algorithm

- **Mutua Esclusione**
- **No deadlock:** Deadlock possibile, ma gestito e annullato grazie al meccanismo INQUIRE-RELINQUISH.
- **No starvation**

Quorum approach: Maekawa Algorithm

- Bassa domanda: (dimQuorum - 1) REQUEST + (dimQuorum - 1) LOCKED + (dimQuorum - 1) RELEASE
- Alta domanda: (dimQuorum - 1) REQUEST + (dimQuorum - 1) FAILED + (dimQuorum - 1) LOCKED + (dimQuorum - 1) RELEASE
- Richiesta con SN inferiore: (dimQuorum - 1) REQUEST + (dimQuorum - 1) INQUIRE + (dimQuorum - 1) RELIQUISH + (dimQuorum - 1) LOCKED + (dimQuorum - 1) RELEASE
- Si ha dimQuorum = $\sqrt{N}$
- **Complessità:** 3 to 5($\sqrt{N} - 1$) messaggi

Quorum approach: Maekawa Algorithm

- Bassa domanda: (dimQuorum - 1) REQUEST + (dimQuorum - 1) LOCKED + (dimQuorum - 1) RELEASE
- Alta domanda: (dimQuorum - 1) REQUEST + (dimQuorum - 1) FAILED + (dimQuorum - 1) LOCKED + (dimQuorum - 1) RELEASE
- Richiesta con SN inferiore: (dimQuorum - 1) REQUEST + (dimQuorum - 1) INQUIRE + (dimQuorum - 1) RELIQUISH + (dimQuorum - 1) LOCKED + (dimQuorum - 1) RELEASE
- Si ha dimQuorum = $\sqrt{N}$
- **Complessità:** 3 to 5($\sqrt{N} - 1$) messaggi