Algoritmi di Sincronizzazione Distribuiti

La sincronizzazione nei sistemi distribuiti

Il modello a scambio di messaggi è la naturale astrazione di un sistema distribuito, nel quale processi distinti eseguono su nodi fisicamente separati, collegati tra di loro attraverso una rete.

Caratteristiche dei sistemi distribuiti:

- concorrenza/parallelismo delle attività nei nodi
- assenza di risorse condivise tra nodi
- assenza di un clock globale
- possibilità di malfunzionamenti indipendenti:
  - nei nodi (crash, attacchi, …);
  - nella rete di comunicazione (ritardi, perdite di messaggi, ecc.).

Proprietà desiderate nei sistemi distribuiti

Nelle applicazioni distribuite è importante poter soddisfare alcune proprietà:

- **scalabilità**: nell'applicazione distribuita le prestazioni dovrebbero aumentare al crescere del numero di nodi utilizzati.

- **tolleranza ai guasti**: l'applicazione è in grado di funzionare anche in presenza di guasti (es. crash di un nodo, problemi sulla rete, ecc.)

Prestazioni: speedup

Oltre al tempo di calcolo, un indicatore per misurare le prestazioni di un sistema parallelo/distribuito è dato dallo **Speedup**.

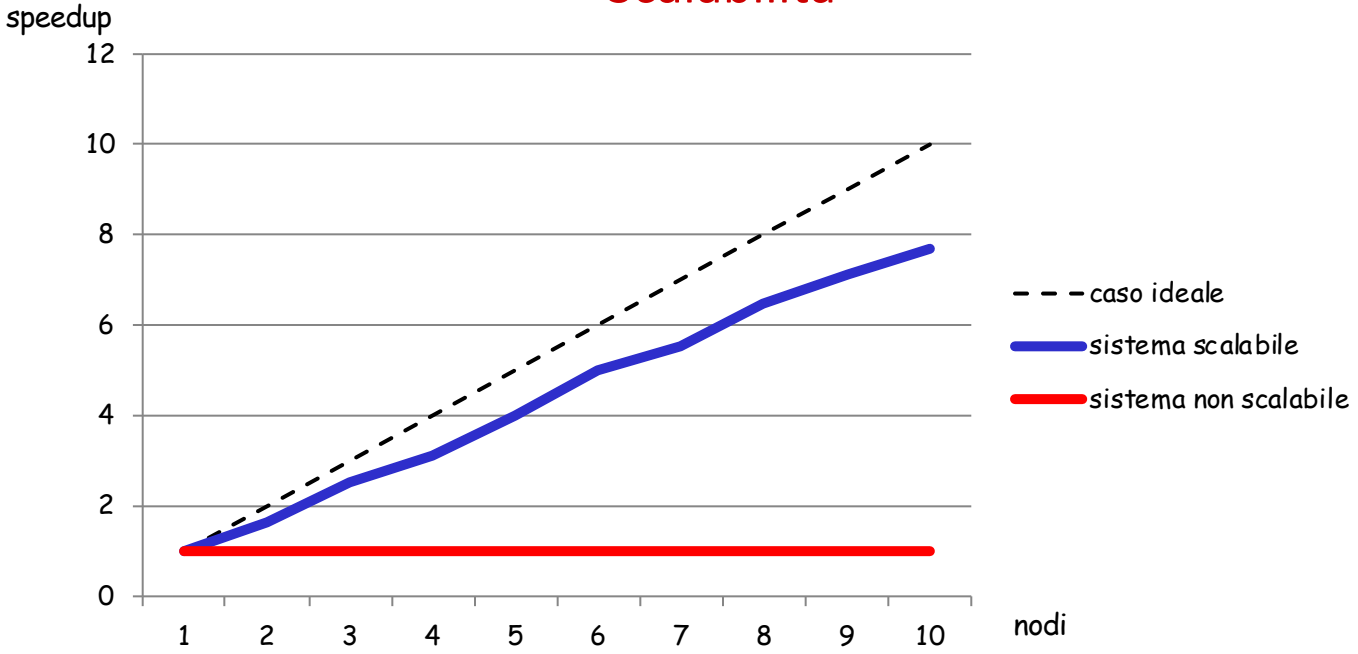
Ipotizzando che la stessa applicazione possa essere eseguita su un numero variabile di nodi:

lo speedup per n nodi è il rapporto tra il tempo di esecuzione dell'applicazione ottenuto con 1 solo nodo e quello ottenuto con n nodi:

$$Speedup(n)=Tempo(1)/Tempo(n)$$

-> Lo speedup è una funzione del numero di nodi.

Scalabilità



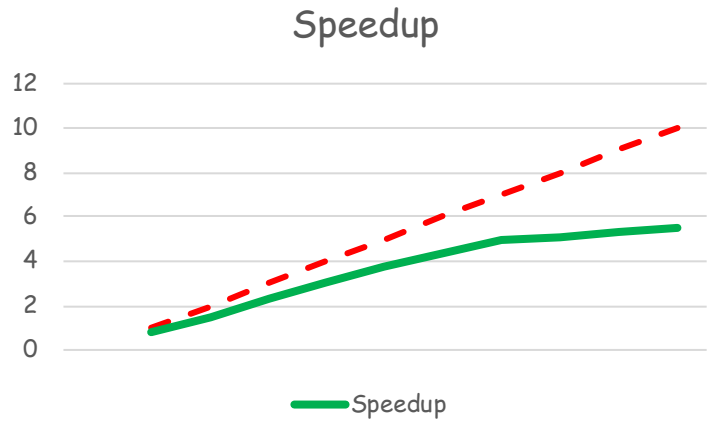
Il grafico traccia la relazione tra speedup e numero di nodi n.

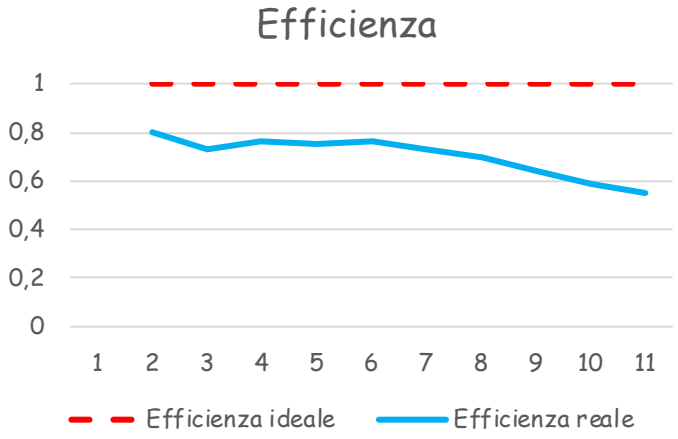
Lo speedup ideale è: Speedup(n)=Tempo(1)/Tempo(n)=n

Scalabilità e Efficienza

Oltre allo speedup, per valutare le prestazioni di un sistema distribuito/parallelo si può misurarne l'efficienza E(n):

E(n)=Speedup(n)/n





Tolleranza ai guasti

Il sistema distribuito tollerante ai guasti riesce a erogare i propri servizi anche in presenza di guasti in uno o più nodi.

Tipi di guasto:

- Transiente
- Intermittente
- Persistente

Tolleranza ai guasti

Un sistema tollerante ai guasti è in grado di funzionare correttamente anche in presenza di guasti in uno o più componenti.

La Tolleranza ai guasti può essere conseguita, ad esempio, con tecniche di **ridondanza**:

- vengono mantenute più istanze degli stessi componenti, in modo da poter rimpiazzare l'elemento guasto con un elemento equivalente.

È necessario prevedere meccanismi:

- per la **rilevazione** dei guasti (fault **detection**)
- per il **ripristino** del sistema dopo ogni guasto rilevato (**recovery**)

Algoritmi di Sincronizzazione

Come nel modello a memoria comune, anche nel modello a scambio di messaggi è importante poter disporre di algoritmi di sincronizzazione tra i processi concorrenti, che consentano di risolvere alcune problematiche comuni attraverso un opportuno coordinamento i vari processi.

Ad esempio:

- **timing**: sincronizzazione dei clock e tempo logico
- **mutua esclusione distribuita**
- **elezione** di coordinatori in gruppi di processi

E' desiderabile che gli algoritmi distribuiti godano delle proprietà di scalabilità e di tolleranza ai guasti.

Algoritmi per la gestione del tempo

Tempo nei sistemi distribuiti

In un sistema distribuito, ogni nodo è dotato di un proprio orologio.

Se gli orologi locali di due nodi non sono sincronizzati, è possibile che se un evento e2 accade nel nodo N2 dopo un altro evento e1 nel nodo N1, ad e2 sia associato un istante temporale precedente quello di e1.

👉Possibili problemi in applicazioni distribuite.

Ad es.: compilazione con make, utility basata sulla data dei file

Tempo nei sistemi distribuiti

In applicazioni distribuite può essere necessario avere **un unico riferimento temporale**, condiviso da tutti i partecipanti.

Si può realizzare con:

-un **orologio fisico universale**: a questo scopo vengono utilizzati degli algoritmi di sincronizzazione dei nodi del sistema (es. algoritmo di Berkeley, alg. di Cristian) che garantiscono che il tempo misurato da ogni clock su ogni nodo sia lo stesso. Utile, ad esempio , se c'è bisogno di utilizzare l'ora esatta.

-un "**orologio logico"**, che permetta di associare ad ogni evento un istante logico (timestamp) la cui relazione con i timestamp di altri eventi sia coerente con l'ordine in cui essi si verificano.

Orologi logici (Lamport, 1979)

In un'applicazione distribuita, gli eventi sono legati da vincoli di precedenza che danno origine ad una relazione d'ordine parziale (v. algoritmi non sequenziali)

Relazione di precedenza tra eventi (*Happened-Before*, ->):

1.se a e b sono eventi in uno stesso processo ed **a si verifica prima di b**: **a->b**

2.se **a** è l'evento di **invio di un messaggio** e **b** è l'evento di **ricezione** dello stesso messaggio: **a->b**

3.se **a->b** e **b->c** allora a->c

Data una coppia di eventi (a,b) sono possibili 3 casi:

1.a->b , cioè a avviene prima di b

2.b->a , cioè b avviene prima di a

3.a e b non sono legati dalla relazione **HB** => a e b sono concorrenti

Orologi logici

Assumiamo che ad ogni evento e venga associato un **timestamp C(e)**.

Si vuole definire un modo per misurare il concetto di tempo tale per cui ad ogni evento a possiamo associare un timestamp C(a) sul quale tutti i processi siano d'accordo.

I timestamp devono soddisfare la seguente proprietà:

se **a -> b allora C(a) < C(b)** [*]

Quindi:

1. Se all'interno di un processo **a precede b**, allora **C(a)<C(b)**
2. Se a è l'evento di invio (in un processo Ps) e b l'evento di ricezione (in un processo Pr) dello stesso messaggio m allora **C(a) < C(b)**.

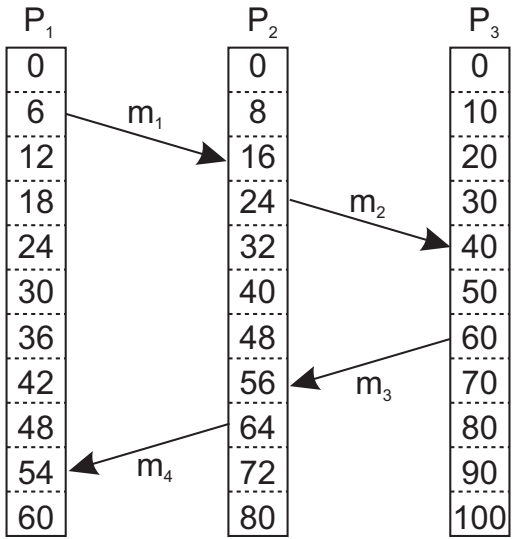
Algoritmo di Lamport

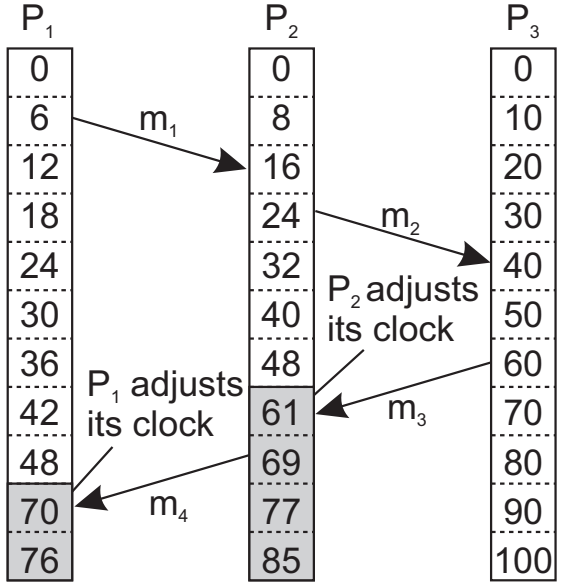
Per garantire il rispetto della proprietà [*] ogni processo Pi mantiene localmente un **contatore** del tempo logico Ci, che viene gestito nel modo seguente:

1. ogni nuovo evento all'interno di Pi provoca **un incremento** del valore di Ci: Ci=Ci+1
2. ogni volta che Pi **vuole inviare un messaggio** m, il contatore Ci viene incrementato: Ci=Ci+1 e successivamente il messaggio viene inviato, insieme al timestamp Ci: ts(m)=Ci.
3. Quando un processo Pj riceve un messaggio m (con timestamp ts(m)), Pj assegna al proprio contatore Cj un valore uguale al **massimo tra Cj e ts(m):** Cj=max{Cj, ts(m)} e successivamente lo incrementa di 1: Cj=Cj+1

Orologio logico di lamport: esempio

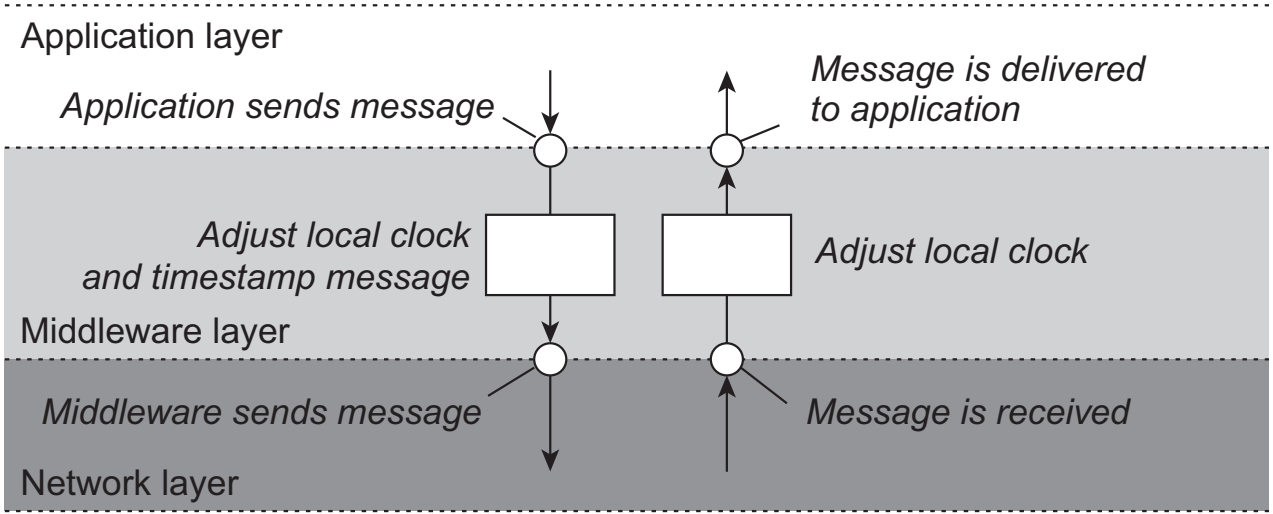
Consideriamo 3 processi in esecuzione su 3 nodi, ognuno con il proprio clock (frequenze diverse):





Lamport: aggiornamenti degli orologi

Nei sistemi distribuiti l'algoritmo di Lamport viene generalmente eseguito da uno strato software (middleware) che interfaccia i processi alla rete: i processi vedono il tempo logico.



La Mutua Esclusione in sistemi distribuiti

Mutua esclusione distribuita

Obiettivo: garantire che due o più processi non possano eseguire contemporaneamente certe attività.

Ad esempio: accesso a risorse "condivise" : file, stampanti, ecc.

Soluzioni:

- **centralizzata**: la risorsa è gestita da un processo dedicato (coordinatore), al quale tutti i processi si rivolgono per accedervi.

- **decentralizzata**: non è previsto un coordinatore, quindi i processi in competizione si sincronizzano tra loro tramite opportuni algoritmi, la cui logica è distribuita tra tutti i processi.

Una soluzione decentralizzata è, in generale, più scalabile di soluzioni centralizzate, nelle quali il processo gestore della risorsa rappresenta un "collo di bottiglia".

Soluzioni distribuite al problema della mutua esclusione

In generale, possiamo suddividere le soluzioni in due categorie:

1.Permission-based: ogni processo che vuole eseguire la sua sezione critica, richiede un permesso ad uno o più altri processi.

2.Token-based: un oggetto ("testimone", o token) viene passato tra i vari processi in competizione: il processo che possiede il token può:

- eseguire la sua sezione critica,

oppure

- passare il token a un altro processo, se non intenzionato ad entrare nella sezione critica.

=> Algoritmi token-based sono sempre decentralizzati, mentre algoritmi permission-based possono essere sia centralizzati che decentralizzati.

Mutua esclusione: soluzione centralizzata

- L'algoritmo si basa su **permessi** (permission-based):
  - la risorsa viene gestita da un **processo coordinatore** al quale ogni processo che vuole eseguire la sua sezione critica si rivolge per ottenere il permesso.
  - per eseguire la propria sezione critica ogni processo Pi:
    1. **Richiesta**: Pi invia una richiesta di autorizzazione al coordinatore; quando verrà accolta, il processo Pi otterrà il permesso.
    2. <esegue la sezione critica>
    3. **Rilascio**: Pi comunica al coordinatore il termine della sezione critica

- **Il coordinatore concede un permesso alla volta:** ogni Richiesta ricevuta da un processo Pi mentre l'autorizzazione è concessa al processo Pj viene messa in attesa.
- Il coordinatore mantiene una coda delle Richieste in attesa.

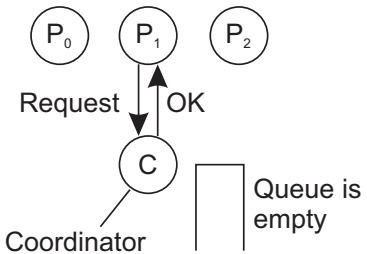
Mutua esclusione: soluzione centralizzata

Esempio:

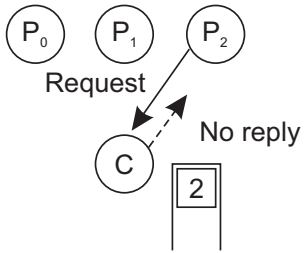
1) P1 chiede il permesso al coordinatore, che autorizza l'accesso.

2) P2 chiede il permesso al coordinatore, che non può concederlo perchè P1 sta eseguendo la sua sezione critica; il Coordinatore non risponde e inserisce la richiesta di P2 nella coda.

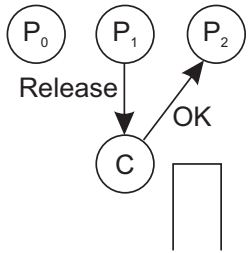
3) quando P1 termina la sezione critica, invia un messaggio di Rilascio al coordinatore, che quindi può successivamente autorizzare P2.



(1)



(2)



(3)

Commenti sulla soluzione centralizzata

Vantaggi:

L'algoritmo è equo (non c'è starvation).

Prevede solo 3 messaggi (richiesta-autorizzazione-rilascio) per ogni sezione critica.

Svantaggi:

Scalabilità: la soluzione è poco scalabile perchè al crescere del numero dei processi il coordinatore può diventare un collo di bottiglia.

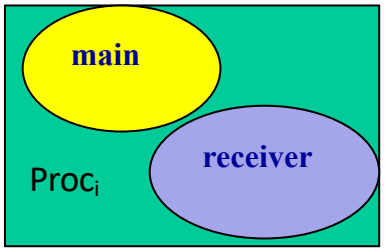
Tolleranza ai guasti:

- il coordinatore può essere soggetto a guasto e, in questo caso, l'intero sistema si blocca (Single Point of Failure).
- Inoltre, se un processo P che ha fatto una richiesta non ottiene una risposta, P non può distinguere le due possibili cause:
  - autorizzazione non concessa
  - coordinatore guasto.

Mutua esclusione: Algoritmo Ricart-Agrawala

Algoritmo decentralizzato basato su permessi (permission-based).

- Il sistema è costituito da un insieme di processi in competizione.
- Ad ogni processo sono associate 2 attività concorrenti (es. thread):
  - **main**: il thread che esegue la sezione critica
  - **receiver**: il thread dedicato alla ricezione delle autorizzazioni



REQUISITO: Si assume che vi sia un temporizzatore globale per tutti i nodi (es: orologio logico): ogni messaggio è corredato da un timestamp che ne indica l'istante di invio.

Algoritmo Ricart - Agrawala

Struttura del main:

Quando un processo **vuole entrare nella sezione critica**:

1. invia (n-1) richieste di autorizzazione ai receiver degli altri (n-1) nodi : **Request(Pid, timestamp)**

2. attende le **(n-1) autorizzazioni**

3. esegue la **sezione critica**

4. invia **OK** a tutte le richieste in attesa

Algoritmo Ricart - Agrawala

Struttura del receiver:

Quando riceve una richiesta, il receiver può trovarsi in uno di tre possibili stati:

1.RELEASED. Il processo non è interessato ad entrare nella sezione critica: risponde OK.

2.WANTED. Il processo è in procinto di entrare nella sezione critica (attende autorizzazione): confronta il timestamp Tr della richiesta ricevuta con quello della richiesta inviata (Ts):

- se Tr < Ts risponde OK
- altrimenti non risponde e mette la richiesta ricevuta in coda

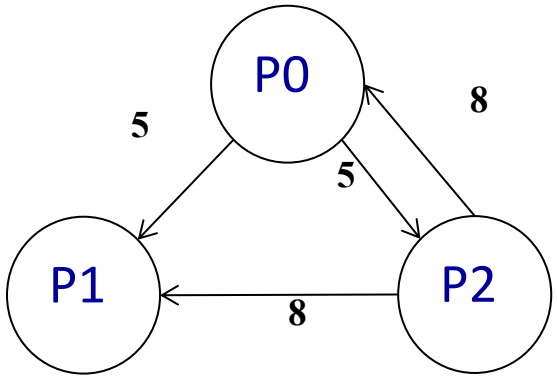
3.HELD. Il processo sta eseguendo la sezione critica: la richiesta viene messa in coda.

Ricart – Agrawala: Esempio

Consideriamo 3 processi: P0, P1, P2.

Se P0 e P2 vogliono eseguire una sezione critica (cioè sono entrambi nello stato W):

- P0 invia a P1 e P2 una Richiesta (timestamp=5)
- P2 invia a P0 e P1 una richiesta (timestamp=8)

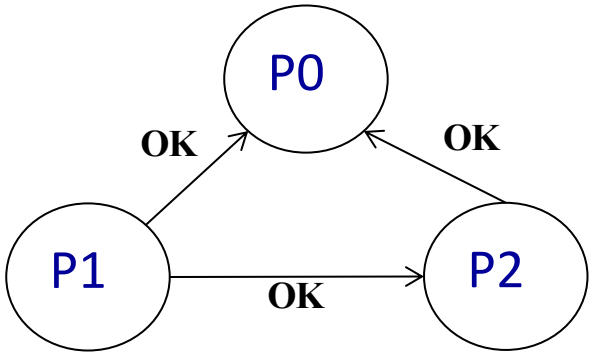


Ricart – Agrawala: Esempio

P0 è in stato WANTED: non risponde a P2 (perckè 5<8) e mette in coda la sua richiesta.

P1 è in stato RELEASED: risponde ad entrambi OK.

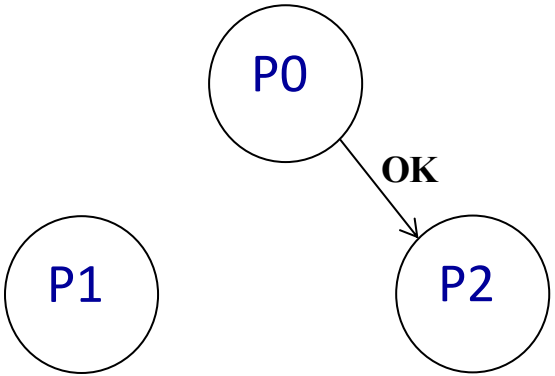
P2 è in stato WANTED: risponde OK a P0 (perckè 5<8).



=> P0 esegue la sezione critica.

Ricart – Agrawala: Esempio

Terminata la sezione critica, P0 invia OK a P2 (la cui richiesta era stata accodata).



=> P2 esegue la sezione critica.

Commenti sull'algoritmo Ricart-Agrawala

Vantaggi:

Distribuzione -> **scalabilità**.

Svantaggi:

- Maggior costo di comunicazione per il singolo partecipante: 2*(N-1) messaggi per ogni sezione critica.

Non c'è più un single point of failure, ma N points of failure: se uno dei nodi va in crash, non risponderà alle richieste degli altri.

Impossibilità di rilevare il guasto: l'attesa è dovuta a guasto o a sezioni critiche in esecuzione?

Guasti nell'algoritmo Ricart-Agrawala

Rilevazione dei guasti: si può modificare il protocollo prevedendo che ad ogni richiesta venga **sempre** fornita una risposta:

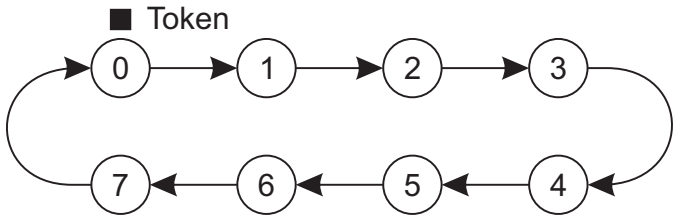
- –Ok
- –Attendi (autorizzazione negata): in questo caso il richiedente si mette in attesa di un OK.

$\Rightarrow$ Il richiedente può impostare timeout per rilevare guasti del destinatario.

N points of failure: ogni processo ha un ruolo equivalente agli altri; se è guasto, può essere semplicemente escluso dal gruppo.

Mutua esclusione: Algoritmo Token ring

- Algoritmo distribuito basato su token.
- Il sistema è costituito da un insieme di processi in competizione, collegati tra di loro secondo una topologia logica ad anello.
- Ogni processo conosce i suoi vicini nell'anello.
- Un messaggio (detto token), circola attraverso l'anello, nel verso relativo all'ordine dei processi nella topologia.



Mutua esclusione: Algoritmo Token ring

Il token rappresenta il permesso unico di eseguire sezioni critiche: chi detiene il token può eseguire la sua sezione critica.

Il token viene inizialmente inviato da P0 a P1:

- P1 può essere nello stato WANTED: in questo caso trattiene il token ed esegue la sezione critica; al termine passa il token al processo successivo P2.
- P1 può essere nello stato RELEASED: in questo caso passa il token al processo successivo P2.

P2 si comporta allo stesso modo di P1, come tutti i processi successivi nell'anello.

In questo modo il token circola nell'anello nel verso stabilito dall'ordine dei processi: P0->P1 ->P2->..->PN->P0...

Commenti sull'algoritmo Token Ring

Vantaggi:

Distribuzione -> scalabilità.

Svantaggi:

- Il numero messaggi per ogni sezione critica non è limitato: [1,..∞].

- Non c'è più un single point of failure, ma N points of failure: se uno dei nodi va in crash, interrompe la catena. Per fare fronte a questo problema, si può modificare il protocollo prevedendo che ad ogni invio del token (da P_i a P_{i+1}), venga sempre restituita una risposta; se la risposta non arriva (allo scadere di un timeout), P_{i+1} viene escluso dall'anello ed il token viene inviato a P_{i+2} .

- Possibilità di perdere il token: ad es. se va in crash il processo che lo detiene, il token va perduto. → rilevazione del problema e rigenerazione del token

Algoritmo	num messaggi per sezione critica		Problemi
Centralizzato	3		scarsa scalabilità crash coordinatore
Ricart-Agrawala	2*(n-1)		crash qualunque processo
Token Ring	1..∞		crash qualunque processo perdita token

Algoritmi di elezione

In alcuni algoritmi è previsto che un processo rivesta un ruolo speciale nella sincronizzazione tra tutti i nodi. Tale processo viene detto **coordinatore**. (es. il coordinatore nell'algoritmo di mutua esclusione centralizzato)

La **designazione del coordinatore** può essere:

- **statica**: il coordinatore viene deciso in modo arbitrario dal programmatore/amministratore prima dell'esecuzione.
- **dinamica**: il coordinatore viene designato a runtime dai processi del sistema con un **algoritmo di elezione**.

Ad esempio: Se il coordinatore di un gruppo di processi subisce un crash, per ripristinare l'operatività del sistema, è necessario individuare a runtime un altro processo del gruppo a cui attribuire il ruolo di nuovo coordinatore.

Algoritmi di Elezione

Assunzioni di base:

- ogni processo è identificato da un unico ID. Il dominio degli Id è totalmente ordinato (ad esempio, insieme degli interi).
- ogni processo conosce gli ID di tutti gli altri (ma non il loro stato: se è attivo o terminato)

Obiettivo: Viene designato vincitore il processo attivo che ha l'ID più alto.

Algoritmo Bully

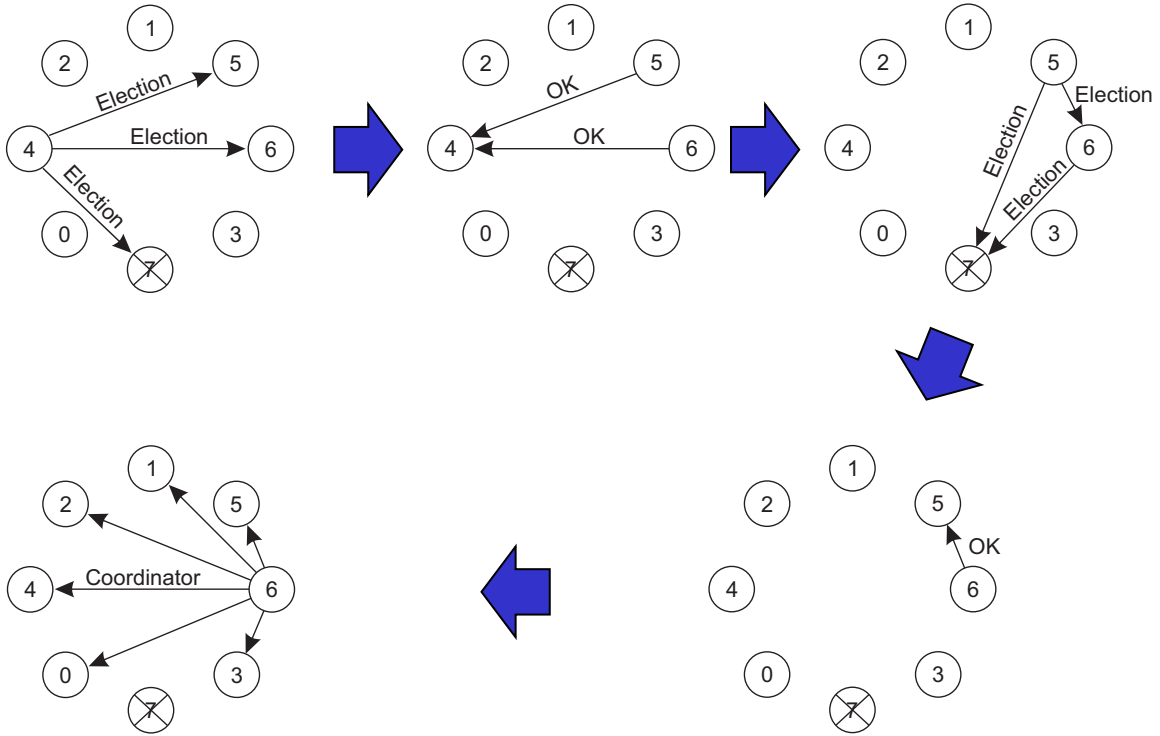
Consideriamo un sistema composto da N processi {P0 , . . . , PN −1}; sia Id(Pi)=i.

Quando un processo Pk rileva che il coordinatore non è più attivo, organizza un'elezione:

1. Pk invia un messaggio "ELEZIONE" a tutti i processi con ID più alto: Pk+1,Pk+2,...,PN−1.
2. Se nessun processo risponde, Pk vince l'elezione e diventa il nuovo coordinatore; comunica quindi a tutti gli altri processi il nuovo ruolo tramite il messaggio "COORDINATOR"
3. Se un processo Pj (j>k) risponde, Pj prende il controllo e Pk rinuncia ed esce dall'elezione.

Ogni processo attivo risponde ad ogni messaggio di ELEZIONE ricevuto.

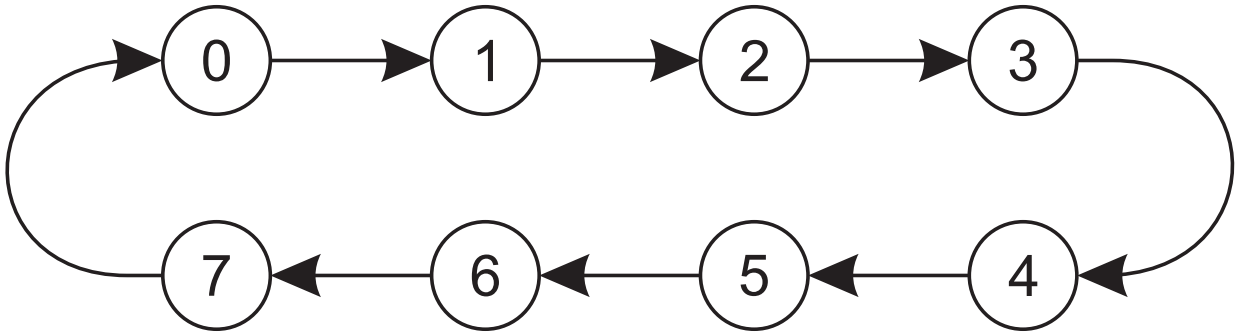
Bully: esempio



Algoritmo di elezione ad Anello

I processi del gruppo sono collegati tramite una topologia logica ad anello. La posizione (ID) che ogni processo occupa all'interno dell'anello rappresenta la sua priorità.

Il processo attivo con la massima priorità viene eletto **coordinatore**.



Algoritmo ad anello

Quando un qualunque processo P_i si rende conto che il coordinatore non risponde più, inizia un'elezione:

1. P_i invia un messaggio ELEZIONE contenente il suo ID al suo successore P_{i+1}. Se il successore P_{i+1} è in crash, il messaggio viene spedito al successore di P_i ., ecc.
2. Quando un processo $P_j$riceve un messaggio ELEZIONE:
   - se il messggio non contiene l'ID di Pj, aggiunge il suo ID al messaggio e lo spedisce al successivo ecc..
   - se il messaggio contiene l'ID di Pj, significa che è stato compiuto un giro completo dell'anello: Pj designa come coordinatore il processo corrispondente all'ID più alto nel messaggio ricevuto e invia al successivo processo un messaggio COORDINATOR contenente l'ID del processo designato come nuovo coordinatore.
3. Quando un processo riceve un messaggio COORDINATOR, ne prende atto e inoltra lo stesso messaggio al successivo.

Algoritmo ad anello: esempio

P3 e P6 avviano entrambi un'elezione: entrambe le elezioni designeranno P6 come nuovo coordinatore. Ogni messaggio COORDINATOR dopo aver percorso l'anello verrà eliminato.