Algoritmi avanzati

a.a. 2010/2011

Prof.ssa Rossella Petreschi

DIPARTIMENTO
DI INFORMATICA

SAPIENZA
UNIVERSITÀ DI ROMA

Broadcast Distribuito

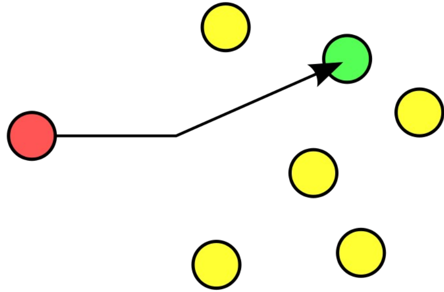
a cura di Diego Cammarano

Contenuto

- Broadcast
- Qualità del servizio broadcast:
  - ***ordinamento*** : sorgente-singola FIFO, totalmente ordinato, causalmente ordinato
  - ***affidabilità***
- Implementazione di un servizio broadcast
- Algoritmo broadcast totalmente ordinato
- Algoritmo broadcast causalmente ordinato

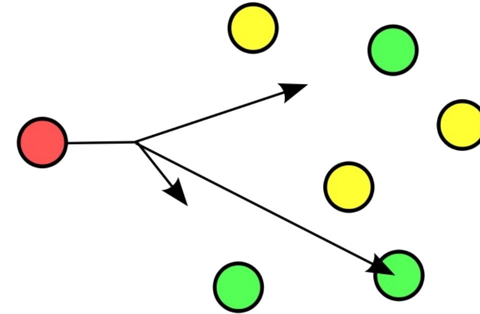
Tipologie invio di messaggi

Unicast



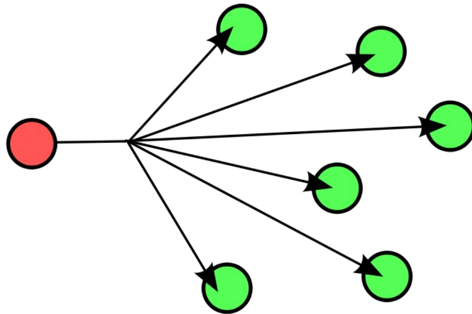
Connessione punto-punto

Anycast



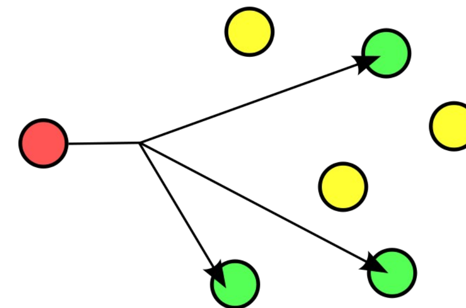
Servizio di DNS

Broadcast



Radio Network

Multicast



Streaming multimediale

Sistema Broadcast

Sistema che permette lo scambio asincrono di messaggi tra processori caratterizzato da interfacce diverse a seconda delle funzionalità o servizi che devono essere implementati

L'interfaccia di un servizio generico di broadcast prevede due tipi di eventi:

bc-send$_i$(*m*) : evento di input generato dal processore p_i che invia un messaggio *m* ad un sottoinsieme di processori

bc-recv$_i$(*m, j*) : evento di output generato dal processore p_i che riceve il messaggio *m* inviato da p_j

Proprietà interfaccia broadcast

Esiste un mapping fra ogni evento bc-recv$_i$(*m*, *j*) e il suo corrispondente evento bc-send$_i$(*m*) con le seguenti proprietà

- **Integrità:** ogni messaggio ricevuto è stato precedentemente inviato, nessun messaggio è ricevuto “dal nulla”
- **Nessun duplicato:** nessun messaggio è ricevuto più di una volta da un qualsiasi processore
- **Liveness:** ogni messaggio inviato è ricevuto da ogni processore

Specifiche servizio broadcast

Broadcast può essere facilmente simulato ma presenta dei problemi che devono essere affrontati per avere un servizio e risultati corretti

- non c'è nessuna garanzia sull'***ordine*** con cui sono ricevuti dai processori

- non c'è nessuna garanzia sull'***affidabilità*** del sistema e dei possibili processori che possono non essere più in grado di inviare/ricevere messaggi

Qualità del servizio broadcast

Ordinamento

- I processori vedono tutti i messaggi nello stesso ordine ?
- L'ordine in cui sono ricevuti i messaggi corrisponde a quello con cui sono stati inviati ?

Affidabilità

- Tutti i processori vedono lo stesso insieme di messaggi anche in presenza di guasti ?
- Tutti i processori vedono tutti i messaggi inviati in broadcast da un processore attivo ?

Requisiti ordinamento

Sorgente Singola FIFO (ssf): per tutti i messaggi m_1 e m_2 e tutti i processori p_i e p_j, se p_i invia m_1 prima di inviare m_2 allora m_2 non è ricevuto da p_j prima di m_1

Totalmente ordinato (to): per tutti i messaggi m_1 e m_2 e tutti i processori p_i e p_j, se m_1 è ricevuto da p_i prima dell'invio di m_2 allora m_2 non è ricevuto da p_j prima di m_1

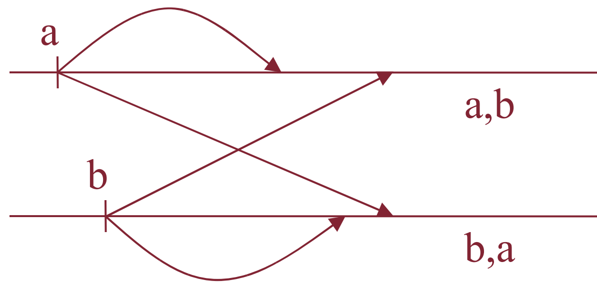
Causalmente ordinato (co): per tutti i messaggi m_1 e m_2 e ogni processore p_i, se m_1 "accade prima" di m_2 allora m_2 non è ricevuto da p_i prima di m_1

Nota: il messaggio m_1 "accade prima" di m_2 se o:

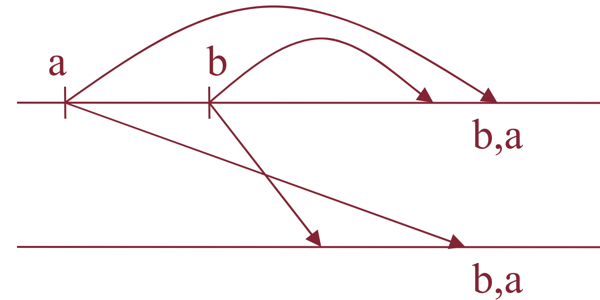
- l'evento bc-recv per m_1 si verifica prima del bc-send per m_2 oppure
- m_1 e m_2 sono inviati dallo stesso processore e m_1 è inviato prima di m_2

Relazioni tra i requisiti di ordinamento

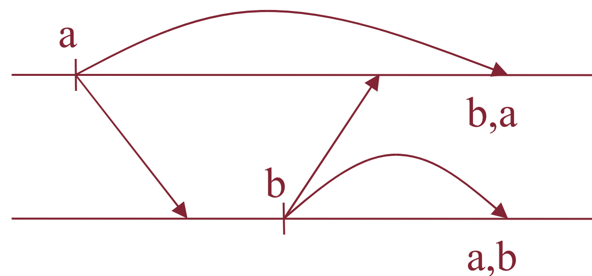
- Solo il **Causalmente ordinato (co)** implica **Sorgente-Singola FIFO (ssf)** perchè la relazione sui messaggi “accade prima” rispetta l’ordine con cui ogni processore invia i messaggi



a. Casualmente ordinato (**co** non implica **to**)



b.Totalmente ordinato (non implica **co** oppure **ssf**)



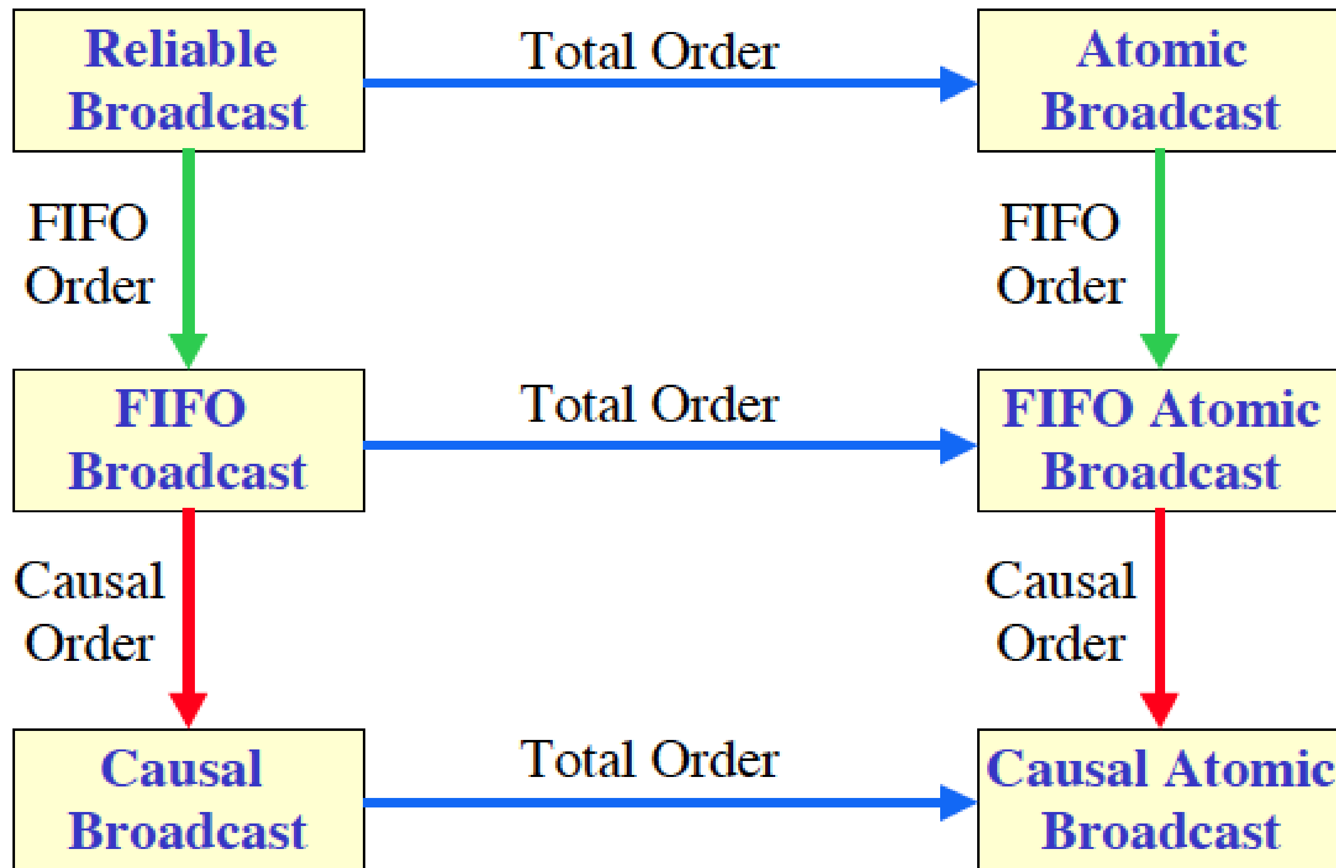
c. Sorgente-singola FIFO (non implica **to** oppure **co**)

Combinazioni di servizi broadcast

In funzione dei tipi di ordinamento si ha :

- **FIFO broadcast** è un broadcast affidabile con ordinamento a singola-sorgente FIFO
- **Total broadcast** è un broadcast affidabile totalmente ordinato
- **Causal broadcast** è un broadcast che garantisce la causalità e quindi anche il FIFO broadcast

Costruzione modulare di servizi broadcast



Affidabilità (1)

Nel caso di processori soggetti a guasti la proprietà di ***Liveness*** deve essere indebolita.

In presenza di guasti i processori vengono partizionati in due gruppi ed il mapping fra eventi bc-recv(m) e bc-send(m) deve soddisfare le seguenti caratteristiche:

- ***Integrità:*** ogni messaggio ricevuto è stato precedentemente inviato, nessun messaggio è ricevuto "dal nulla"

- ***Nessun duplicato:*** nessun messaggio è ricevuto più di una volta da un qualsiasi processore

Affidabilità (2)

- ***Liveness non soggetta a guasti:*** ogni messaggio inviato da un processore attivo è ricevuto da tutti i processori attivi

- ***Liveness soggetta a guasti:*** ogni messaggio inviato da un processore non più attivo o è ricevuto da ogni processore attivo o da nessuno di essi

Queste condizioni non garantiscono comunque la ricezione dei messaggi dai processori non attivi

Servizio base (broadcast)

- **Servizio base broadcast** : ogni evento bc-recv$_i$(m,j,qos) corrisponde ad un precedente evento bc-send$_i$(m,qos); ogni messaggio ricevuto è stato precedentemente inviato (**Integrità**), e ogni messaggio inviato è ricevuto una volta (**Liveness**) e solo una volta (**Nessun Duplicato**) da ogni processore
- **bc-send**$_i$(*m*,*qos*): evento di input del processore p_i, che invia un messaggio *m* a tutti i processori, contenente un indicazione del mittente; *qos* è un parametro che descrive la *qualità* del servizio richiesto.
- **bc-recv**$_i$(*m*,*j*,*qos*): evento di output nel quale il processore p_i riceve il messaggio *m* inviato da p_j; *qos*, descrive la *qualità* del servizio prestato.

Implementazione servizio broadcast

Servizio base di broadcast

- quando si verifica una ***bc-send*** per il messaggio *m* sul processore $\boldsymbol{p_i}$, $\boldsymbol{p_i}$ usa il sistema asincrono di messaggistica punto-punto sottostante per inviare una copia di m a tutti i processori
- quando un processore riceve il messaggio da $\boldsymbol{p_i}$ esegue la bc-recv per ***m*** ed ***i***

Ordinamento a sorgente-singola FIFO

- Ogni processore assegna un numero di sequenza ai messaggi che invia in broadcast

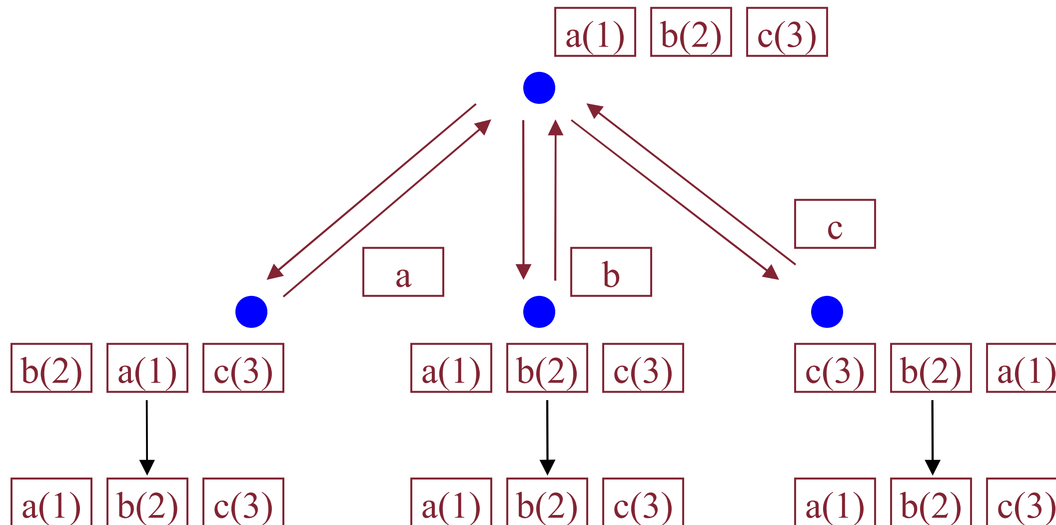
- Incrementa tale numero di uno ad ogni nuovo invio

- Il processore p_i con numero di sequenza T aspetta di ricevere il messaggio fino a quando ha ricevuto tutti i precedenti messaggi con numero di sequenza minore di T

Broadcast totalmente ordinato

Algoritmo asimmetrico

1. Il processore p_i invia m usando il servizio base verso un unico processore p_c coordinatore centrale (identificato da un token)
2. Il processore p_c assegna un numero progressivo ad ogni messaggio e lo invia a tutti i processori tramite il servizio base
3. Il messaggio(k) è ricevuto se tutti i messaggi(i) ($0 <= i < k$) sono ricevuti



Broadcast totalmente ordinato

Algoritmo simmetrico

I processori decidono insieme l'ordine di broadcast dei messaggi tenendo conto del timestamp ad essi assegnato. Quindi

- ogni processore mantiene un contatore (timestamp), che viene allegato ai messaggi
- si assume che il sistema di comunicazione sottostante fornisca il broadcast a sorgente-singola FIFO
- il processore p_i aggiorna la cella relativa a p_j usando i tags sui messaggi ricevuti da p_j e usando uno speciale messaggio di 'timestamp update' inviato da p_j.

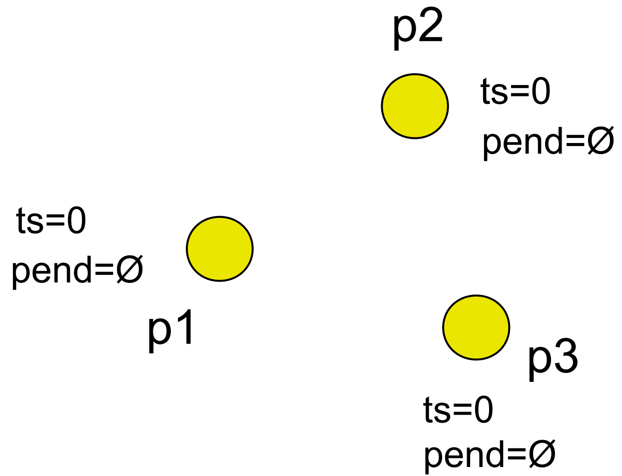
Broadcast totalmente ordinato

Initially $ts[j] = 0, 0 \leq j \leq n-1$, and *pending* is empty

```
1:  when bc-send_i(m,to) occurs:      // quality of service to means totally ordered
2:      ts[i] := ts[i] + 1
3:      add ⟨m,ts[i],i⟩ to pending
4:      enable bc-send_i(⟨m,ts[i]⟩,ssf)
                                // quality of service ssf means single-source FIFO

5:  when bc-recv_i(⟨m,T⟩,j,ssf), j ≠ i, occurs:
                                // j indicates sender; ignore messages from self
6:      ts[j] := T
7:      add ⟨m,T,j⟩ to pending
8:      if T > ts[i] then
9:          ts[i] := T
10:         enable bc-send_i(⟨ts-up,T⟩,ssf)     // bcast timestamp update message

11: when bc-recv_i(⟨ts-up,T⟩,j,ssf), j ≠ i, occurs:    // ignore messages from self
12:     ts[j] := T

13: enable bc-recv_i(m,j,to) when
14:     ⟨m,T,j⟩ is the entry in pending with the smallest (T, j)
15:     T ≤ ts[k] for all k
16: result: remove ⟨m,T,j⟩ from pending
```

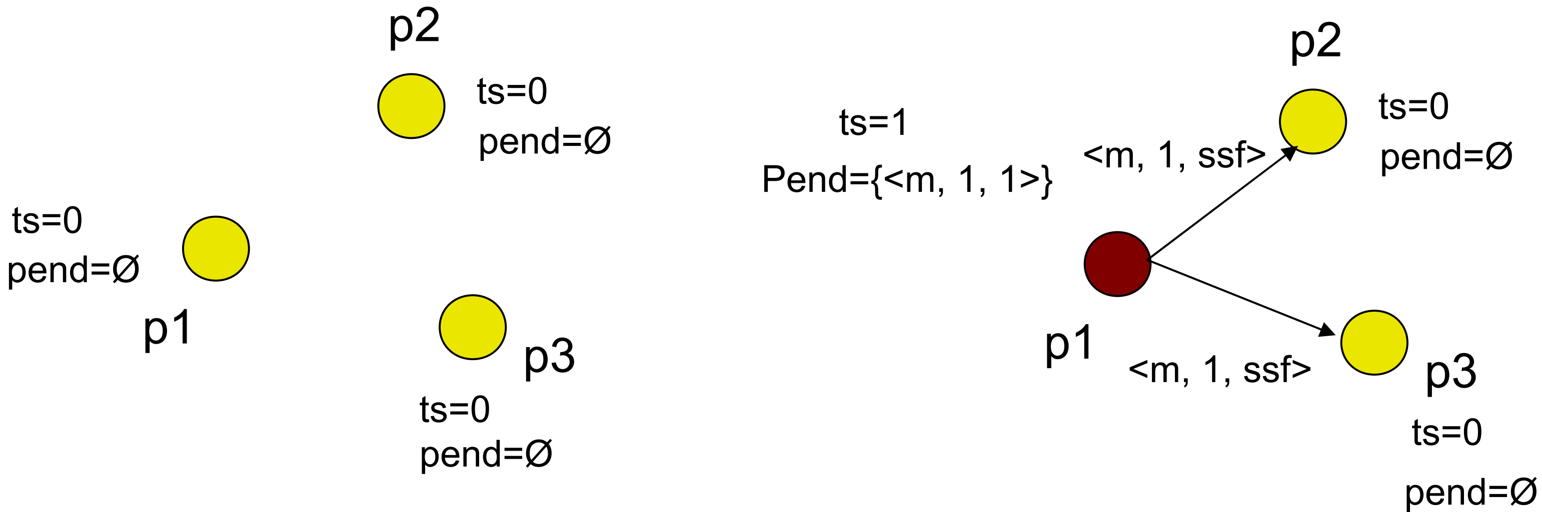
Esempio

bc-send$_i$(m,to)



Esempio

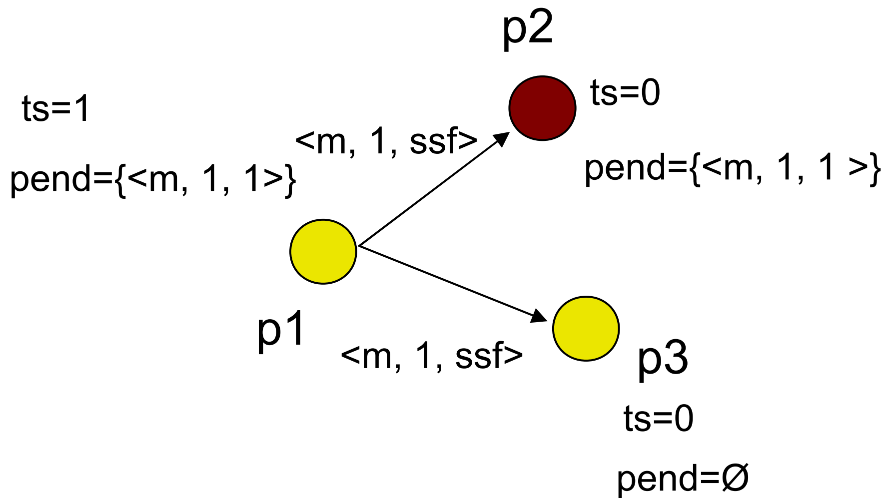
bc-send$_i$(m,to)



1: when bc-send$_i(m$,to) occurs: // quality of service to means totally ordered
2: $ts[i] := ts[i] + 1$
3: add $\langle m, ts[i], i\rangle$ to *pending*
4: enable bc-send$_i(\langle m, ts[i]\rangle$,ssf)
// quality of service ssf means single-source FIFO

Esempio

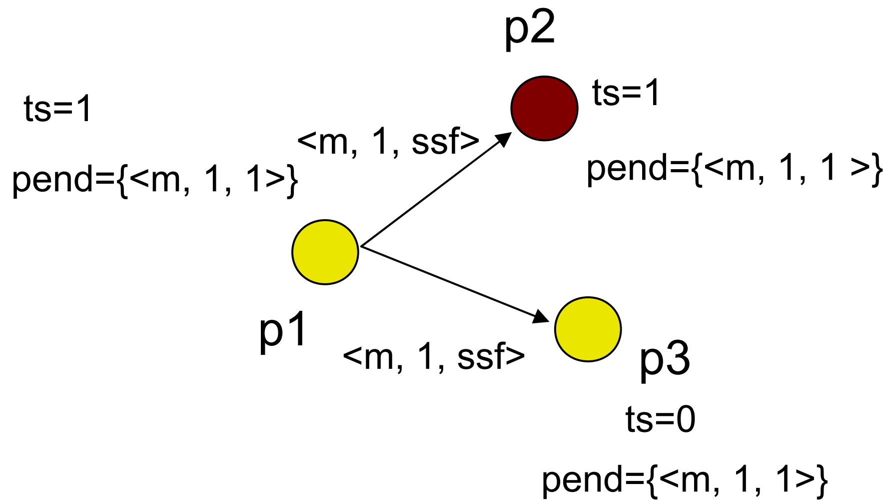
bc-recv$_i$(<m,T>,j,ssf)

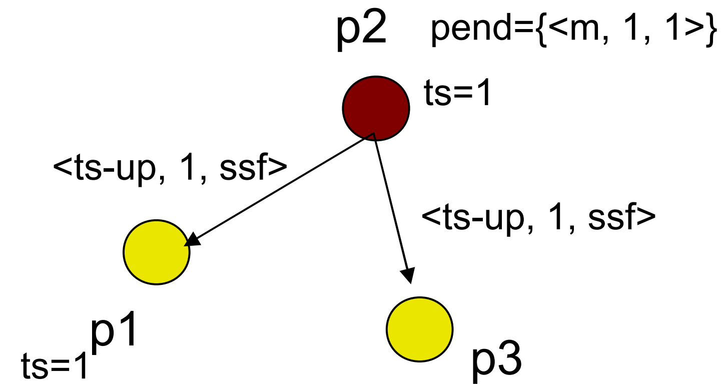


5: when bc-recv$_i$($\langle m,T \rangle$,j,ssf), $j \neq i$, occurs:
// j indicates sender; ignore messages from self
6: $ts[j] := T$
7: add $\langle m,T,j \rangle$ to *pending*
8: if $T > ts[i]$ then
9: $ts[i] := T$
10: enable bc-send$_i$($\langle ts\text{-}up,T \rangle$,ssf) // bcast timestamp update message

Esempio

bc-recv$_i$(<m,T>, j, ssf)

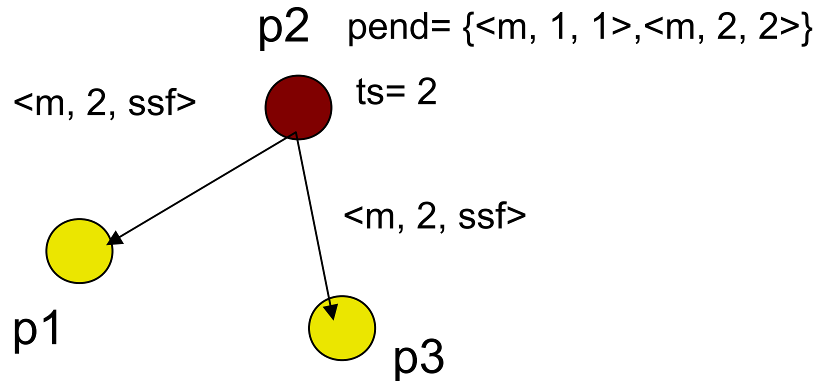




5: when bc-recv$_i$($\langle m,T\rangle$,j,ssf), $j \neq i$, occurs:
// j indicates sender; ignore messages from self
6: $ts[j] := T$
7: add $\langle m,T,j\rangle$ to *pending*
8: if $T > ts[i]$ then
9: $ts[i] := T$
10: enable bc-send$_i$($\langle$*ts-up*,$T\rangle$,ssf) // bcast timestamp update message

Esempio

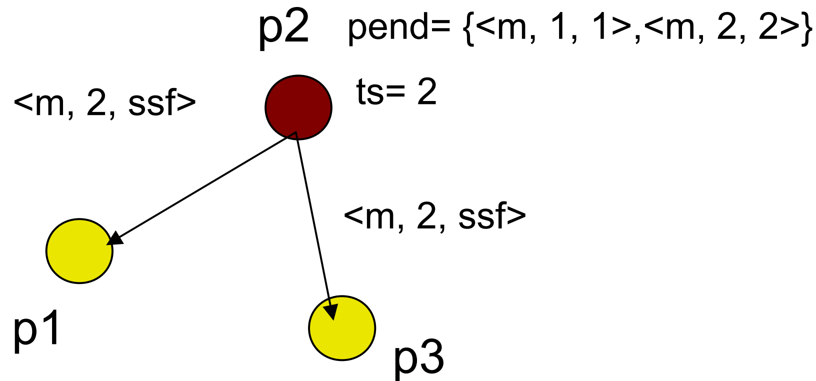
bc-recv$_i$(m, j, to)

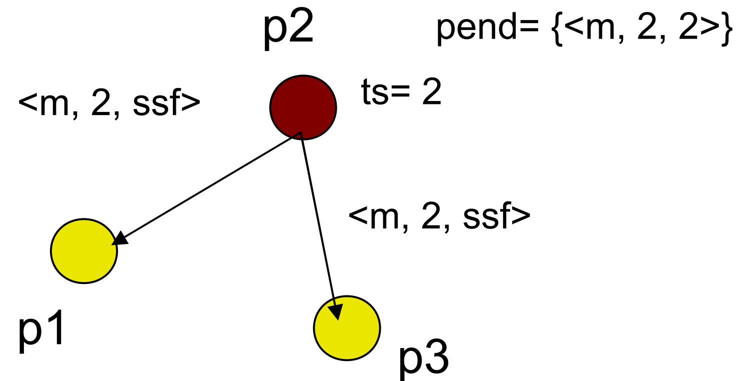


13: enable bc-recv$_i$(m,j,to) when
14: $\langle m, T, j \rangle$ is the entry in *pending* with the smallest (T, j)
15: $T \leq ts[k]$ for all k
16: result: remove $\langle m, T, j \rangle$ from *pending*

Esempio

bc-recv$_i$(m, j, to)





13: enable bc-recv$_i$(m,j,to) when
14: $\langle m, T, j\rangle$ is the entry in *pending* with the smallest (T, j)
15: $T \leq ts[k]$ for all k
16: result: remove $\langle m, T, j\rangle$ from *pending*

Broadcast causalmente ordinato (1)

Algoritmo

- usa vettore di timestamps
- ogni processore mantiene un *vector clock* e ogni messaggio è taggato con il *vector timestamp* che rappresenta il valore corrente del *vector clock*
- prima che un messaggio possa essere ricevuto (causalmente), deve passare attraverso un filtro di “ordinamento logico” che assicura che i messaggi vengano ricevuti nel corretto ordine

Vantaggi

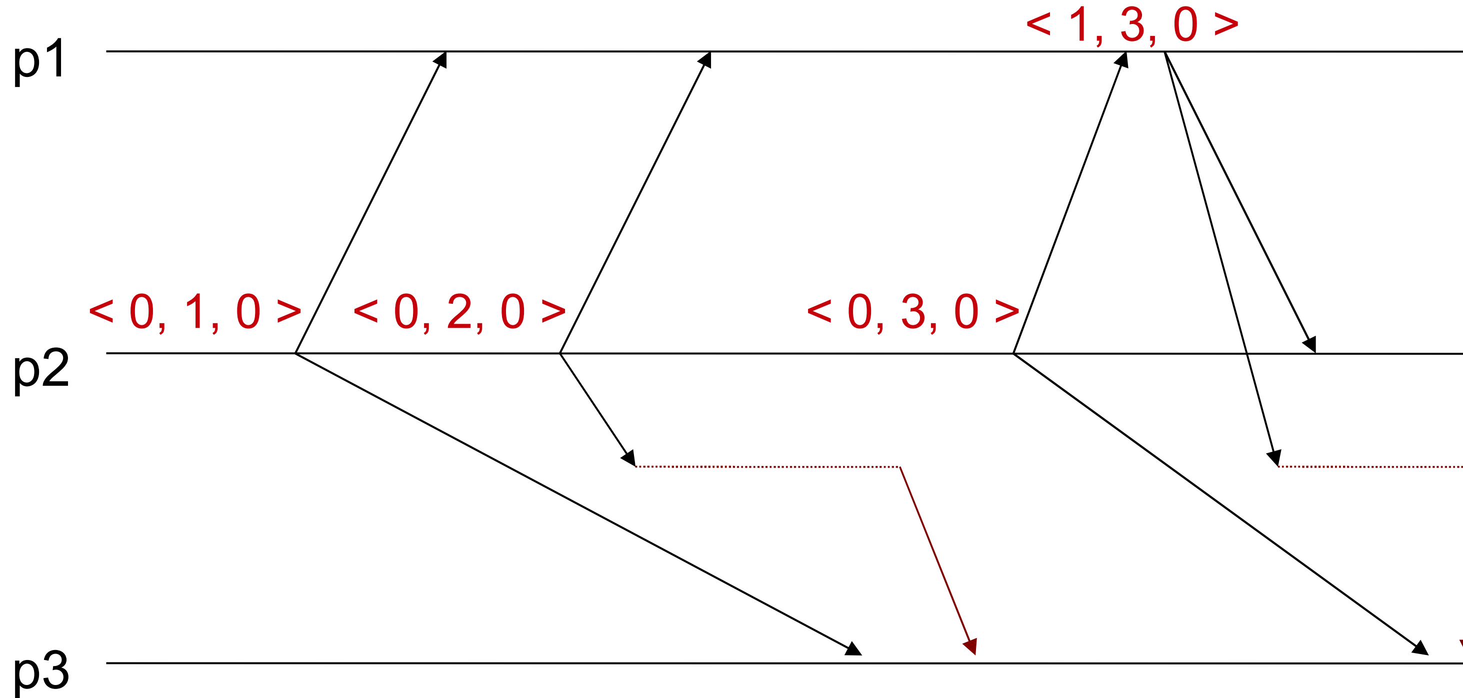
- più efficiente rispetto all’algoritmo totalmente ordinato perchè non tiene conto dell'ordine
- funziona anche in caso di guasti

Broadcast causalmente ordinato (2)

Algorithm 22 Causally ordered broadcast algorithm: code for p_i, $0 \leq i \leq n-1$.

```
Initially vt[j] = 0, 0 ≤ j ≤ n − 1, and pending is empty

1:  when bc-send_i(m,co) occurs:
                                // quality of service co means causally ordered
2:      vt[i] := vt[i] + 1
3:      enable bc-recv_i(⟨m⟩,co)                // receive the message locally
4:      enable bc-send_i(⟨m,vt⟩,basic)
                     // quality of service basic means ordinary broadcast

5:  when bc-recv_i(⟨m,v⟩,j,basic), j ≠ i, occurs:
                         // j indicates sender; ignore messages from self
6:      add ⟨m, v, j⟩ to pending

7:  enable bc-recv_i(m, j, co) when:
8:      ⟨m, v, j⟩ is in pending
9:      v[j] = vt[j] + 1
10:     v[k] ≤ vt[k] for all k ≠ i
11: result: remove ⟨m, v, j⟩ from pending
12:     vt[j] := vt[j] + 1
```

Esempio



- Il messaggio < 0,2,0 > resta in attesa del messaggio < 0, 1, 0 > da p2

- Il messaggio < 1,3,0 > resta in attesa del messaggio < 0, 3, 0 > da p1