



Sincronizzazione

SCD

Anno accademico 2013/14
Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@math.unipd.it

Laurea Magistrale in Informatica, Università di Padova

1 / 24



Sistemi distribuiti: sincronizzazione

Stato del sistema – 2

❑ Distributed snapshot

○ Riflette uno stato globale consistente come potrebbe essere stato nel recente passato

- Esempio di stato inconsistente: P ha ricevuto un messaggio da Q il cui invio non risulta dallo stato globale

○ Rappresenta un “taglio” (*cut*) nell’evoluzione temporale individuale dei processi del sistema

- Fissa ciò che appartiene allo stato globale e ciò che ne è escluso
- Il “percorso” del taglio (causato dall’algoritmo usato) determina la consistenza dello stato

○ Algoritmo ideato da K. Chandy e L. Lamport

- Distributed Snapshots: Determining Global States of Distributed Systems. ACM Transactions on Computer Systems, 3(1):63-75, 1985

Laurea Magistrale in Informatica, Università di Padova

3 / 24



Sistemi distribuiti: sincronizzazione

Stato del sistema – 1

❑ Stato globale di un sistema distribuito

○ Lo stato locale di ciascun processo

- Non necessariamente tutto
- Solo ciò che è importante ai fini dello stato globale

○ L’insieme dei messaggi in transito

❑ Conoscere lo stato globale consente di

○ Verificare se il sistema è globalmente attivo oppure no

- Nessun messaggio in transito → nessuna attività globale

○ Se no, quali cause: normale terminazione oppure stallo?

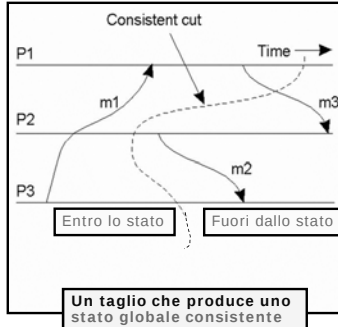
Laurea Magistrale in Informatica, Università di Padova

2 / 24



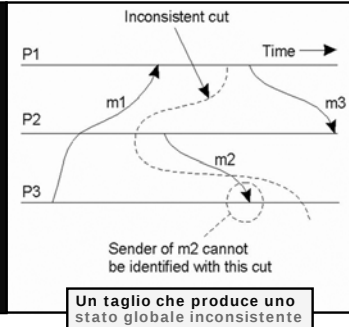
Sistemi distribuiti: sincronizzazione

Stato del sistema – 3



Consistent cut

Un taglio che produce uno stato globale consistente



Inconsistent cut

Un taglio che produce uno stato globale inconsistente

Laurea Magistrale in Informatica, Università di Padova

4 / 24



Sistemi distribuiti: sincronizzazione

Stato del sistema – 4

❑ Situazioni pericolose

○ Messaggi inconsistenti

- Inviati dal processo M dopo aver salvato il proprio stato (*checkpoint*) ma ricevuti dal processo D prima di aver salvato il proprio stato
- L'invio di un messaggio inconsistente rischia di essere duplicato in caso di ripristino dello stato
- Se il suo effetto non è idempotente lo stato di D ne viene corrotto!

○ Messaggi in transito (*in-flight*)

- Inviati da M prima di aver salvato il proprio stato
- Ricevuti da D dopo aver salvato il proprio stato

❑ Uno stato consistente non ammette messaggi inconsistenti

Laurea Magistrale in Informatica, Università di Padova

5 / 24



Sistemi distribuiti: sincronizzazione

Distributed snapshot – 2

❑ Il processo che riceve un *marker* da un suo canale di ingresso C

○ Se non ha ancora salvato il suo stato locale lo salva e invia il *marker* su tutti i propri canali in uscita

○ Se già lo ha salvato inizia a salvare lo stato del canale C fino all'arrivo del *marker* di fine istantanea

- L'insieme dei messaggi ricevuti su C a partire dall'ultimo salvataggio del sistema

❑ Un processo ha fatto la sua parte quando abbia trattato tutti i *marker* ricevuti in ingresso

❑ Quando tutti i processi coinvolti dall'iniziatore hanno completato, l'istantanea è finita

Laurea Magistrale in Informatica, Università di Padova

7 / 24



Sistemi distribuiti: sincronizzazione

Distributed snapshot – 1

❑ Sistema visto come insieme di processi connessi da canali diretti punto-a-punto

○ *Overlay network* sulla topologia fisica

❑ Qualunque processo può iniziare

○ Più istantanee possono essere simultaneamente in corso

❑ L'iniziatore salva il suo stato e invia un *marker* sui suoi canali di uscita

- Chiedendo a destinatari di partecipare all'istantanea
- Il *marker* identifica l'iniziatore e la versione di istantanea

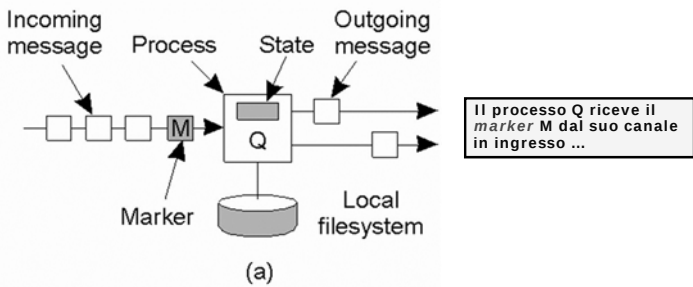
Laurea Magistrale in Informatica, Università di Padova

6 / 24



Sistemi distribuiti: sincronizzazione

Distributed snapshot – 3



(a)

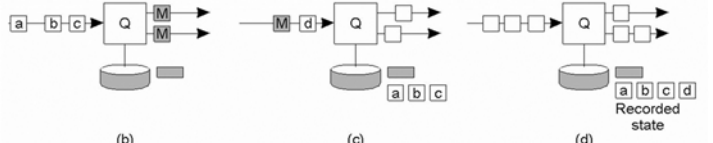
Laurea Magistrale in Informatica, Università di Padova

8 / 24



Sistemi distribuiti: sincronizzazione

Distributed snapshot – 4



(b) (c) (d)

... Salva il proprio stato locale e invia il *marker* su ciascuno dei suoi canali in uscita ...

... e comincia a salvare anche lo stato del suo canale in ingresso ...

... All'arrivo del successivo *marker* sul suo canale in ingresso ha completato il suo contributo alla cattura dello stato globale

Recorded state

Questo algoritmo produce stati consistenti?

Laurea Magistrale in Informatica, Università di Padova

9 / 24



Sistemi distribuiti: sincronizzazione

Esempio d'uso: terminazione – 2

- Il processo Q invia il messaggio "FINITO" se
 - Tutti i suoi successori hanno inviato indietro il messaggio "FINITO"
 - Q non ha ricevuto dai predecessori alcun messaggio successivo al completamento del suo stato
- Altrimenti Q invia il messaggio "CONTINUA" al suo predecessore
 - L'iniziatore allora invia un nuovo *marker* per una nuova istantanea finché non riceva solo messaggi "FINITO" dai suoi successori

Laurea Magistrale in Informatica, Università di Padova

11 / 24



Sistemi distribuiti: sincronizzazione

Esempio d'uso: terminazione – 1

- Il processo Q che ha ricevuto un *marker* per la prima volta ne considera il mittente M sul canale in ingresso come suo predecessore
 - Q diventa dunque il successore di M
- Quando Q ha fatto la sua parte invia a M il messaggio "FINITO"
- Lo stato globale per l'iniziatore M' è pronto quando ogni suo successore M abbia ricevuto messaggi "FINITO" da tutti i suoi successori Q'
 - Nel caso generale lo stato globale può includere messaggi in transito
 - Segno di attività non completate

Laurea Magistrale in Informatica, Università di Padova

10 / 24



Sistemi distribuiti: sincronizzazione

Esempio d'uso: terminazione – 3

- Esiste moltissima documentazione su rete che applica, discute, ed estende l'algoritmo di Chandy & Lamport
 - Per un interessante esempio animato, si veda
 - <http://www.risc.uni-linz.ac.at/software/daj/snapshot/index.html>
 - Perdonando l'inglese ☺
 - Vale la pena fare qualche ricerca e qualche riflessione al riguardo!

Laurea Magistrale in Informatica, Università di Padova

12 / 24



Sistemi distribuiti: sincronizzazione

Elezione del coordinatore

- ❑ La presenza di un coordinatore facilita la costruzione di algoritmi distribuiti
- ❑ Eleggere il coordinatore richiede accordo distribuito
 - L'obiettivo dell'algoritmo di elezione è assicurarne la terminazione con l'accordo di tutti i partecipanti
- ❑ Prerequisiti
 - Un identificatore ordinale unico per ciascun processo
 - Ogni processo conosce gli identificatori degli altri processi

Laurea Magistrale in Informatica, Università di Padova

13 / 24



Sistemi distribuiti: sincronizzazione

Algoritmo del «bullo» – 2

- ❑ Un processo che riceve “ELEZIONE” da un processo di identificatore minore risponde “OK” e rileva l'elezione
 - Se P riceve “OK” ha finito il suo lavoro
- ❑ L'algoritmo designa sempre come coordinatore il processo in vita con identificatore maggiore
 - Il vincente informa tutti i processi del sistema che c'è un nuovo coordinatore

Laurea Magistrale in Informatica, Università di Padova

15 / 24



Sistemi distribuiti: sincronizzazione

Algoritmo del «bullo» – 1

- ❑ Il processo P che non conosce il coordinatore o ne rileva l'assenza promuove una elezione
 - P invia “ELEZIONE” a tutti i processi di identificatore maggiore
 - Se nessuno risponde P si proclama coordinatore

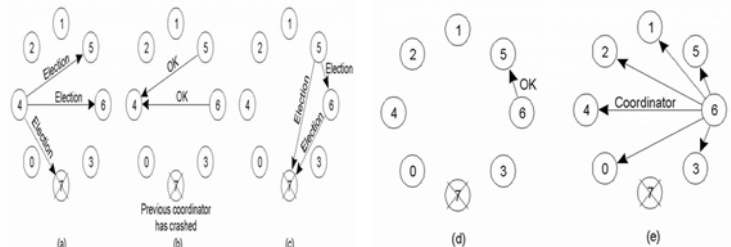
Laurea Magistrale in Informatica, Università di Padova

14 / 24



Sistemi distribuiti: sincronizzazione

Algoritmo del «bullo» – 3



(a) Il processo 4 inizia una nuova elezione

(b) I processi 5 e 6 rilevano l'elezione in parallelo

(c) Il processo 6 rileva l'elezione del processo 5

(d) Il processo 6 non conosce processi di identificatore maggiore, quindi si proclama coordinatore

Laurea Magistrale in Informatica, Università di Padova

16 / 24



Sistemi distribuiti: sincronizzazione

Mutua esclusione – 1

❑ Algoritmo centralizzato: facile ma fragile

- Le richieste “ENTER” di accesso in mutua esclusione a una risorsa condivisa vengono inviate a un coordinatore centrale
- Se la risorsa è libera il coordinatore risponde “GRANTED”
- Altrimenti il coordinatore accoda la richiesta con politica FIFO e risponde (“DENIED”)
- L'utente che rilascia la risorsa invia “RELEASED” al coordinatore
- Il coordinatore allora preleva la prima richiesta in attesa e invia “GRANTED” al suo mittente

❑ Il coordinatore è il *Single Point of Failure* (SPF) dell'algoritmo e il collo di bottiglia

Laurea Magistrale in Informatica, Università di Padova

17 / 24



Sistemi distribuiti: sincronizzazione

Mutua esclusione – 3

❑ Algoritmo distribuito / II

- P aspetta di ricevere “OK” da tutti i processi
 - Quando ciò avviene P può accedere a R in mutua esclusione
 - L'accesso avviene con garanzia di assenza sia di *deadlock* che di *starvation*
- Quando P rilascia R risponde “OK” a tutti i processi mittenti di richieste accodate presso di sé e le rimuove dalla coda
 - A quel punto solo un process riceverà “OK” da tutti gli altri
 - Quello la cui richiesta aveva il C “minore”

Laurea Magistrale in Informatica, Università di Padova

19 / 24



Sistemi distribuiti: sincronizzazione

Mutua esclusione – 2

❑ Algoritmo distribuito / I

- Il processo P che chiede accesso esclusivo alla risorsa R invia un messaggio M (“GRANT?”) con <P, R, C> a tutti i processi del sistema
 - C = ora locale di P
- Il processo che riceva M
 - Se non sta usando R e non la vuole (per ora) risponde “OK”
 - Se sta usando R non risponde e accoda M presso di sé
 - Se ha chiesto R ma non l'ha ancora ottenuta confronta C di M con la sua ora locale di richiesta: risponde “OK” solo se C è precedente

Laurea Magistrale in Informatica, Università di Padova

18 / 24



Sistemi distribuiti: sincronizzazione

Mutua esclusione – 4

❑ Critica

- Gli SPF aumentano da 1 (il coordinatore) a N (tutti i processi)
 - Tutti i processi devono sempre rispondere a ogni richiesta
 - La mancata risposta (*time-out*) interpretata come “occupato”
 - Alla ricezione del primo “occupato” il richiedente deve bloccarsi in attesa del primo “OK” successivo
- E in più servono una infrastruttura di comunicazione affidabile e orologi coerenti (!)

Laurea Magistrale in Informatica, Università di Padova

20 / 24



Sistemi distribuiti: sincronizzazione

Mutua esclusione – 5

❑ Algoritmo a *token ring*

○ Processi collegati in sequenza circolare ordinata e punto a punto

- Il gettone transita circolarmente

○ Il processo in posizione 0 riceve per primo il gettone

- Il possesso del gettone il processo può accedere una singola risorsa (senza cumulo)
- Poi deve passare il gettone al suo vicino
- Se il processo non ha immediato bisogno di risorse passa subito il gettone al vicino
- Il vicino conferma la ricezione altrimenti viene rimosso dalla sequenza

○ Nel caso peggiore un processo richiedente aspetta una intera rotazione del gettone

❑ Il gettone è 1 SPF → se perso va rigenerato

Laurea Magistrale in Informatica, Università di Padova

21 / 24



Sistemi distribuiti: sincronizzazione

Mutua esclusione – 6

Algoritmo	# Messaggi per accesso e rilascio risorsa	Max attesa di accesso spesa per invio messaggi (costo >> lavoro)	Punti deboli (SPF)
Centralizzato	3 (ENTER, GRANTED, RELEASED)	2 (ENTER, GRANTED)	Guasto del coordinatore
Distribuito	2 (n – 1) (GRANT?, RELEASED) da uno a tutti gli altri	2 (n – 1)	Guasto di qualsiasi processo
Gettone circolante	1 .. ∞ (se tutti [1] o nessuno [∞] sono interessati alla risorsa)	0 .. n – 1 (gettone in possesso, gettone all'altro capo)	Gettone perso Guasto di processo

Raffronto prestazionale tra gli algoritmi

Laurea Magistrale in Informatica, Università di Padova

23 / 24



Sistemi distribuiti: sincronizzazione

Mutua esclusione – 6

❑ I 3 algoritmi possono essere raffrontati in relazione a 3 criteri fondamentali

- Numero di messaggi necessari al processo per poter operare sulla risorsa richiesta (ingresso e uscita)
- Il tempo necessario perché la richiesta abbia successo
- Le debolezze dell'algoritmo

❑ Questi 3 criteri possono essere applicati a varie classi di algoritmi distribuiti

Laurea Magistrale in Informatica, Università di Padova

22 / 24



Sistemi distribuiti: sincronizzazione

Argomenti non trattati

❑ Argomenti importanti per la problematica di questa lezione non trattati per limiti temporali del corso

- Sincronizzazione degli orologi fisici
 - Il *middleware* di ogni nodo del sistema distribuito aggiusta il valore del suo orologio fisico in modo coerente con quello degli altri
- Sincronizzazione degli orologi logici
 - Leslie Lamport ha mostrato come l'accordo degli orologi fisici non sia necessario ma lo sia solo l'ordinamento degli eventi (relazione "precede")
- Transazioni distribuite
 - Come ottenere mutua esclusione e **operazioni atomiche** su dati condivisi (*Atomicity – Consistency – Isolation – Durability*)

Laurea Magistrale in Informatica, Università di Padova

24 / 24

UniPD - SCD 2013/14 - Sistemi Concorrenti e Distribuiti

6