Sincronizzazione in distribuito

SCD

Anno accademico 2019/2020
Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@unipd.it

Stato del sistema – 1

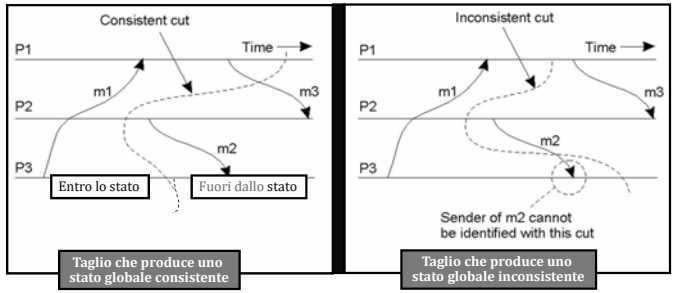
- **Stato globale di un sistema distribuito**
  - **Frazioni dello stato locale di ciascun processo**
  - **L'insieme dei messaggi in transito tra nodi**
- **Conoscendo lo stato globale è possibile**
  - **Verificare se il sistema sia globalmente attivo**
    - Nessun messaggio in transito significa nessuna attività globale
  - **Diagnosticare cause di mancata attività globale**
    - Normale terminazione oppure stallo?

Stato del sistema – 2

- ***Distributed snapshot***
  - **Riflette uno stato globale consistente come potrebbe essere stato nel recente passato**
    - Che quindi ha in se (in potenza) l'evoluzione attesa
    - Lo stato è inconsistente se il nodo P ha ricevuto un messaggio dal nodo Q, il cui invio non sia compreso nello stato globale
  - **Rappresenta un "taglio" nell'evoluzione temporale individuale dei processi del sistema**
    - Fissa ciò che appartiene allo stato globale e ciò che ne è fuori
    - Il "percorso" del taglio (causato dall'algoritmo usato) determina la consistenza dello stato

Stato del sistema – 3

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



Taglio che produce uno stato globale consistente

Taglio che produce uno stato globale inconsistente

Stato del sistema – 4

- **Essenziale discriminare tra**
  - **Messaggi inconsistenti**
    - Inviati da M dopo aver salvato il proprio stato (*checkpoint*) ma ricevuti da D prima di aver salvato il proprio stato
    - Sarebbero nello stato di D senza essere in quello di M
    - Un messaggio inconsistente rischia duplicazione al ripristino dello stato
    - Se il suo effetto non è **idempotente**, il suo arrivo corrompe lo stato di D
  - **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 contiene messaggi inconsistenti**

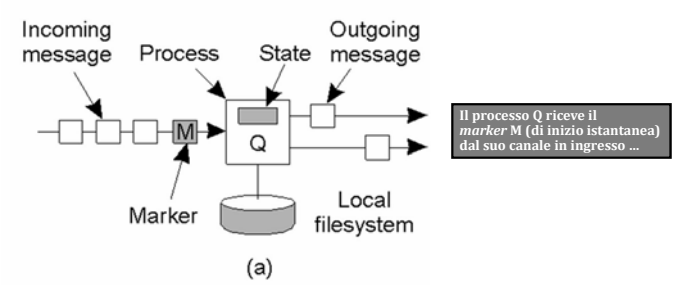
Distributed snapshot – 1

- **Sistema visto come insieme di processi connessi da canali diretti punto-a-punto**
  - **Una *overlay network* sulla topologia fisica**
- **Qualunque processo può esserne iniziatore**
  - **Più istantanee possono procedere simultaneamente**
- **L'iniziatore salva il suo stato e invia un *marker* sui suoi canali di uscita**
  - **Chiedendo ai destinatari di partecipare all'istantanea**
  - **Il *marker* identifica l'istantanea e il suo iniziatore**

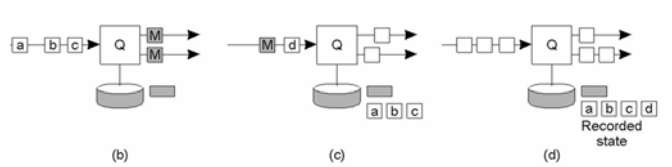
Distributed snapshot – 2

- **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 propaga il *marker* su tutti i propri canali in uscita**
  - **Se già lo ha salvato, continua 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 di stato locale
- **Un processo finisce il suo contributo trattando tutti i *marker* della stessa istantanea ricevuti in ingresso**
- **Quando tutti i processi coinvolti dall'iniziatore hanno completato, l'istantanea è finita**
  - **Come si fa a saperlo?**

Distributed snapshot – 3



Distributed snapshot – 4



... Salva il proprio stato locale e propaga il *marker* sui suoi canali in uscita ...

... Poi comincia a salvare lo stato del suo canale in ingresso ...

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

Perché questo algoritmo produce stati consistenti?

Esempio d'uso: terminazione – 1

- Il processo Q che ha ricevuto un *marker* da un mittente M per la prima volta
  - Considera M come suo predecessore
  - E se stesso (Q) come successore di M
- Quando Q ha fatto la sua parte di chiusura, 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 propri successori
  - Lo stato globale può includere messaggi in transito, segno di attività non completate

Esempio d'uso: terminazione – 2

- Il processo Q invia a M il messaggio "FINITO" s/se
  - Tutti i suoi successori hanno inviato messaggio "FINITO"
  - Q non ha ricevuto dai predecessori alcun messaggio successivo alla sua chiusura di stato
- Altrimenti Q invia il messaggio "CONTINUA" al suo predecessore
  - L'iniziatore allora ci riprova con un nuovo *marker*
  - Finché non riceva messaggi "FINITO" da tutti i suoi successori

Per approfondire

- Fate l'esercizio associato a questa lezione
  - Studiare come il programma concorrente proposto attui l'algoritmo di terminazione distribuita
- Vastissima documentazione sul tema, p.es.
  - https://blog.acolyer.org/2015/04/22/distributed-snapshots-determining-global-states-of-distributed-systems/
  - Un interessante esempio animato (perdonando l'inglese zoppicante):
    - http://www.risc.uni-linz.ac.at/software/daj/snapshot/index.html

Elezione del coordinatore

- La presenza di un coordinatore facilita la costruzione di algoritmi distribuiti
- Eleggere il coordinatore richiede accordo distribuito
  - L'algoritmo di elezione ne assicura la terminazione con l'accordo di tutti i partecipanti
- Prerequisiti
  - Un identificatore ordinale unico per processo
  - Ogni processo conosce gli ID degli altri processi

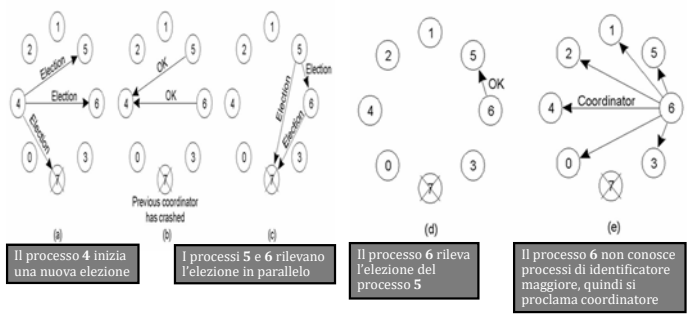
Algoritmo del «bullo» – 1

- Il processo P che non conosca il coordinatore o ne rilevi l'assenza promuove una elezione
- P invia "ELEZIONE" a tutti e soli i processi di ID maggiore del suo
- Se nessuno risponde P si proclama coordinatore

Algoritmo del «bullo» – 2

- Un processo che riceva "ELEZIONE" da un processo di ID minore risponde "OK" e rileva la procedura di elezione
  - Se P riceve "OK" ha finito il suo lavoro
- L'algoritmo designa sempre come coordinatore il processo in vita (attivo e raggiungibile) con ID maggiore
  - Il vincente informa tutti i processi del sistema che c'è un nuovo coordinatore

Algoritmo del «bullo» – 3



Il problema del consenso – 1

- Partizionare dati e lavoro agevola la scalabilità
- Farlo però comporta il rischio che i partecipanti producano valori errati

- Servono soluzioni per assicurare consenso sui valori prodotti, garantendo che, quando serve
  - Possa essere scelto solo un valore che sia stato prima effettivamente proposto
  - Venga scelto un solo valore tra tutti quelli proposti
  - Nessun partecipante riceva notifica di scelta di un valore se questa non sia avvenuta

Il problema del consenso – 2

- Una famosa soluzione a questo problema appare in
  - L. Lamport, *The Part-Time Parliament*, ACM TOCS, 1998
- Divulgata e nota con il nome di "Paxos"
- Tema d'esame #1
  - Studiare, progettare, implementare, presentare questo algoritmo (o della sua versione Raft)

Mutua esclusione – 1

- Algoritmo centralizzato: facile ma fragile
  - Le richieste "ENTER" di accesso esclusivo 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
  - A quel punto il coordinatore 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 suo collo di bottiglia

Mutua esclusione – 2

- Algoritmo distribuito /I
  - Il processo P che chiede accesso esclusivo alla risorsa R invia un messaggio M :: <"GRANT?", P, R, C> a tutti i processi del sistema
    - Dove C = ora locale di P (*timestamp*)
  - 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 il suo C' di richiesta: risponde "OK" se C<C'

Mutua esclusione – 3

- **Algoritmo distribuito /II**
  - **P aspetta di ricevere "OK" da tutti i processi**
    - Quando ciò avviene, P ha ottenuto accesso esclusivo a R
    - Con garanzia di assenza sia di *deadlock* che di *starvation*
  - **Rilasciando R, P invia "OK" a tutti i processi che avevano richieste accodate @P e le rimuove dalla sua coda**
    - A quel punto solo un processo riceverà "OK" da tutti gli altri
    - Quello la cui richiesta aveva il C "minore"

Mutua esclusione – 4

- **Critica della soluzione distribuita**
  - **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*) corrisponde a "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 fisici o logici coerenti (!)**

Mutua esclusione – 5

- **Algoritmo a *token ring* (senza *hold-and-wait*)**
  - **Processi collegati punto a punto in topologia di anello**
    - Il gettone transita circolarmente
  - **Il processo in posizione 0 riceve per primo il gettone**
    - Il processo con gettone può accedere una singola risorsa
    - Poi deve passare il gettone al suo vicino
    - Il processo che non ha immediato bisogno di risorsa passa 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**

Mutua esclusione – 6

- **I 3 algoritmi possono essere messi a confronto su 3 criteri base**
  - **# messaggi necessari al processo per poter operare sulla risorsa richiesta (ingresso e uscita)**
  - **Tempo necessario perché la richiesta abbia successo**
  - **Vulnerabilità (SPF) dell'algoritmo**
- **Questi 3 criteri possono essere applicati a varie classi di algoritmi distribuiti**

Mutua esclusione – 6

Algoritmo	# Messaggi per accesso e rilascio risorsa	Max attesa di accesso spesa per invio messaggi (costo >> lavoro)	Vulnerabilità (SPF)
Centralizzato	**3** (ENTER, GRANTED, RELEASED)	**2** (ENTER, GRANTED)	**Guasto del coordinatore**
Distribuito	**2 (n – 1)** (GRANT?, RELEASED)	**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

Argomenti non trattati

- **Argomenti importanti per la problematica di questa lezione non trattati per limiti temporali**
  - **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 (***A****tomicity* – ***C****onsistency* – ***I****solation* – ***D****urability*)

Tema d'esame #2: l'alternativa della *eventual consistency*