PCD Module 4.1

Sistemi distribuiti: introduzione

Sistemi distribuiti

- Computer che contengono più processori connessi tramite una rete di comunicazione.
- **Perchè usarli?** Scalabilità, modularità, eterogeneità, condivisione dei dati e delle risorse, struttura geografica, affidabilità, basso costo.
- **Perchè non usare solo i distribuiti?** (es. paralleli). Efficienza: aggiornare la memoria è comunque più veloce che scambiarsi messaggi.

Caratteristiche chiave e sfide

- **Assenza di un clock condiviso**: impossibile sincronizzare clock di processori diversi a causa dell'incertezza del ritardo di comunicazione
- **Assenza di una memoria condivisa**: impossibile per un processore conoscere lo stato globale del sistema
- **Assenza di un'accurata gestione dei fallimenti**: in sistemi asincroni distribuiti è impossibile distinguere tra un processore lento o fallito.

Modelli

Modello di programmazione canonico

- Applicazioni come processi "pesanti" che comunicano tramite scambio messaggi su dei canali
- Ogni processo può avere più thread, comunicando tramite una memoria condivisa
- Es.: procedurale (RPC), OO (distribuito, remoto), implementato tramite middleware (CORBA, Java RMI, ecc.)

Modellazione di calcoli distribuiti

- insieme di processi poco accoppiati che mandano messaggi tramite canali unidirezionali: buffer infinito, senza errori, nessun ordine dei messaggi, qualsiasi messaggio inviato ha un ritardo arbitrario ma finito
- stato del canale: sequenza di messaggi inviati ma non ancora ricevuti

Processi: stati ed eventi

- Modello dei processi: set di stati, condizioni iniziali e set di eventi. Ogni evento può cambiare stato del processo e lo stato di un canale incidente a quel processo
- Il comportamento di ogni processo può essere descritto visualmente tramite diagrammi di transizione tra stati.

Modelli principali

- **interleaving**: presupponendo un ordinamento totale tra eventi
- **happended-before**: presupponendo un ordinamento parziale, con ordinamento totale per eventi relativi allo stesso processo
- **potenzialmente causale**: non si presuppone l'ordine di singoli processi composti da più di un thread

Modello a sovrapposizione (interleaving)

- Un'esecuzione è modellata come una sequenza globale di eventi: tutti gli eventi in esecuzione sono sovrapposti. Esempio del sistema bancario con 2 processi (server della banca e cliente)
- Modello formale:
  - *Stato globale*: prodotto incrociato di stati locali e stati dei canali
  - *Stato globale iniziale*: stati locali iniziali + canali vuoti
  - Una *sequenza di eventi* $seq = (e_i : 0 \leq i \leq m)$ è una computazione dei sistemi con modello a sovrapposizione se esiste una sequenza di stati globali $(G_i : 0 \leq i \leq m + 1)$ tale che $G0$ è uno stato iniziale e $G_{i+1} = next(G_i, e_i)$ for $0 \leq i \leq m$, dove $next(G, e)$ è la prossima funzione di stato globale, che ritorna il prossimo stato globale quando l'evento e è eseguito nello stato globale G

Modello "successo prima" (happened-before)

In un sistema distribuito reale possono essere definiti solamente ordini parziali tra eventi, definiti tramite la relazione **happened-before**

Modello:

- il processo P_i genera una sequenza di stati locali ed eventi: $s_{i,0}e_{i,1}\ s_{i,1}e_{i,2}\ ...\ e_{i,l-1}s_{i,l}$ (boh)
- la relazione *happened-before* è la più piccola relazione che soddisfa $(e \lessdot f)$ or $(e \rightsquigarrow f) \Rightarrow e \rightarrow f$
  - $\lessdot$ = relazione *immediately-precedes*, definita tra eventi dello stesso processo: $e \lessdot f$ se e precede immediatamente f
  - $\rightsquigarrow$ = relazione *remotely-precedes*: $e \rightsquigarrow f$ se e è l'evento inviato di un messaggio e f è l'evento ricevuto dello stesso messaggio.

Esecuzione in un modello happened-before: un'esecuzione o computazione in un modello *happened-before* è definito come una tupla $(E, \rightarrow)$ dove E è l'insieme di tutti gli eventi e $\rightarrow$ è un'ordinamento parziale di eventi in E tali che tutti gli eventi all'interno di un singolo processo sono totalmente ordinati. (diagramma sulle slide)

- $e \to f$ se contiene un percorso direzionato dall'evento e a f
- se due eventi non sono relazionati da $\to$ allora sono *concorrenti*: $e \vee f = \neg(e \to f) \wedge \neg(f \to e)$

Modello potenzialmente causale

- *happened-before*: ordinamento totale tra eventi dentro allo stesso processo.
    - Non è vero che tutti questi eventi abbiano relazione causa/effetto
    - La relazione di causalità è parzialmente ordinata tra eventi dello stesso processo
    - difficile o dispendiosa da determinate, consideriamo la relazione potenzialmnete causale
- **potential causality relation** $\to^p$ sul set degli eventi: la più piccola relazione che soddisfa:
    - se un evento e potenzialmente causa un altro evento f sullo stesso processo, allora $e \to^p f$
    - se e è l'invio di un messaggio e f la ricezione, allora $e \to^p f$
    - se $e \to^p f$ and $f \to^p g$, allora $e \to^p g$
    - se due eventi non sono relazionati da $\to^p$ allora sono chiamati indipendenti

Esempi: un processo riceve 2 messaggi da porte differenti e aggiorna oggetti differenti basandosi su questi due messaggi. Un processo basato su 2 thread che accede a insiemi di oggetti mutualmente disgiunti.

Diagramma sulle slide

Modello appropriato

Un programma distribuito può essere visto come un insieme di diagrammi potenzialmente causali che può generare. Un diagramma potenzialmente causale è equivalente a un set di diagrammi happened-before. Ogni happened-before è equivalente a un set globale di sequenze di eventi.

- Interleaving: on a physical time basis, total order on all events
- Happened-before: on a logical order basis, total order on one process
- Potential causality: on a causality basis, partial order on a process

Modelli ad eventi vs. a stati

A seconda delle applicazioni, può essere che un'applicazione sia modellata meglio in termini di stati e non di eventi.

Meccanismi:

- Clock logici
- Clock vettoriali

Clock logici

Tracciano la relazione *happened-before*, dando un ordine totale che può essere successo invede dell'ordine totale che è successo.

Un clock logico C è una mappatura dall'insieme degli stati S all'insieme dei numeri naturali N con i seguenti vincoli: $\forall s, t \in S : s \preccurlyeq t$ or $s \rightsquigarrow t \Rightarrow C(s) < C(t)$. Data la definizione della relazione *happened-before*, il clock logico può essere definito dalla condizione $\forall s, t \in S : s \to t \Rightarrow C(s) < C(t)$.

Implementazione su slide.

Problema: con i clock logici $\forall s, t \in S : s \to t \Rightarrow s.c < t.c$, ma non viceversa:
$\forall s, t \in S : s.c < t.c \Rightarrow s \to t$ non è vero. Risolvibile tramite clock vettoriali.

Clock vettoriali

A vector clock is a map from S to Nk (vector of size k) with the following constraint: ∀ s,t: s → t ⇔ s.v < t.v where – s.v is the vector assigned to state s – given 2 vectors x,y: - x<ymeans∀k∈[1..N],x[k] <=y[k]and∃j∈[1..N]|x[i]<y[j] - x<=ymeans(x<y)||(x=y)

Theorem: Let s and t be states in processes Pi and Pj with vectors s.v and t.v. Then: s → t iff s.v < t.v