Appunti di Distributed Algorithms

Emanuele Gentiletti

Capitolo 1

Introduzione

Il termine *algoritmi distribuiti* include una grande varietà di algoritmi concorrenti usati per una vasta gamma di applicazioni. Originalmente, il termine si riferiva ad algoritmi progettati per essere eseguiti su molti processori "distribuiti" su una vasta area geografica. Nel corso degli anni, date le similarità riscontrate tra queste categorie, il termine racchiude anche gli algoritmi eseguiti su LAN e su multiprocessori a memoria condivisa.

Il termine "processori" suggerisce che si stia trattando di parti hardware. Ai fini di questo trattato è più utile pensare a questi come processi sofware in esecuzione sui processori hardware.

Ci sono molti tipi di algoritmi distribuiti. Alcuni degli attributi per cui si differenziano sono:

- *Il modello di comunicazione tra processi (IPC)*: Gli algoritmi distribuiti sono eseguiti su più processori, che hanno bisogno di comunicare in qualche modo.
- Il *modello di temporizzazione*: I processori possono essere considerati da un lato **completamente sincroni**, eseguendo comunicazioni e computazioni di pari passo. Dall'altro, possono essere **completamente asincroni**, esegendo passi a velocità arbitrarie e in ordine arbitrario. Nel mezzo, si trova una diversa gamma di possibili presupposizioni che possono essere raggruppate con la designazione **parzialmente sincroni**: in questo caso, i processori hanno informazioni parziali sulla temporizzazione degli eventi (es. i processori hanno dei limiti sulle loro velocità relativa o hanno accesso a un clock approssimativamente sincronizzato).
- Il *modello di fallimento*: Si può supporre che il sistema sia completamente affidabile, o che l'algoritmo debba tollerare una certa quantità di comportamenti errati. I processi possono interrompersi, senza avvertimento, oppure

possono esibire un più grave *fallimento bizantino*, dove si comportano in maniera arbitraria.

- I *problemi affrontati*: Ad esempio allocazione di risorse, comunicazione, consenso tra processori distrubuiti, controllo della concorrenza in un database, rilevamento dei deadlock, snapshot globali, e implementazione di diversi tipi di oggetto.

Il comportamento degli algoritmi distribuiti è spesso difficile da comprendere per molti fattori dovuti da un maggiore tasso di *incertezza* e *indipendenza delle attività*. Alcuni dei tipi di incertezza con cui gli algoritmi devono confrontarsi sono:

- numero di processori non noto
- topologia di rete non nota
- input indipendenti in diverse locazioni
- diversi programmi in esecuzione tutti insieme, avviati in diversi momenti, e in operazione a diverse velocità
- non determinismo del processore
- tempi di arrivo dei messaggi incerti
- fallimenti dei processori e delle comunicazioni

Negli algoritmi distribuiti, invece di comprendere tutto sul loro comportamento, il meglio che si può fare spesso è capire alcune proprietà specifiche e certe del loro comportamento.

Capitolo 2

Modello di rete sincrona

Sistemi di rete sincroni

Un *sistema di rete sincrono* consiste di una collezione di processi locati ai nodi di un grafo orientato di rete.

Per definire un sistemda di rete sincrono formalmente, si inizia con un grafo diretto $G = (V, E)$. Si usa la lettera n per indicare $|V|$, il numero di nodi nel grafo di rete.

Per ogni nodo i di G, si usano le notazioni:

- $out\text{-}nbrs_i$ per indicare i *vicini in uscita* di i, ovvero quei nodi che hanno archi entranti che partono da i
- $in\text{-}nbrs_i$ per indicare i *vicini in entrata* di i, ovvero quei nodi da cui partono archi entranti in i.
- $distance(i, j)$ indica la lunghezza del cammino minimo tra i e j in G, se esiste. Altrimenti, $distance(i, j) = \infty$
- $diam$, il *diametro*, indica la massima $distance(i, j)$ tra tutte le coppie di nodi (i, j)

Si suppone anche di avere un alfabeto fisso di messaggi M, e si considera *null* un segnaposto che indica l'assenza di un messaggio.

Associato a ogni nodo $i \in V$, si ha un *processo*, che consiste formalmente dei seguenti componenti:

- $states_i$, un (non necessariamente finito) insieme di *stati*
- $start_i$, un sottoinsieme non vuoto di $states_i$, indicato come gli *stati iniziali*
- $msgs_i$, una *funzione di generazione dei messaggi*, che associa $states_i \times out\text{-}nbrs_i$ a elementi di $M \cup \{null\}$.
- $trans_i$, una *funzione di transizione di stato* che associa a $states_i$ e a vettori di elementi $M \cup \{null\}$ (indirizzati da $in\text{-}nbrs_i$) a $states_i$

Ogni processo ha quindi un insieme di stati, di cui un sottoinsieme è di stati iniziali. L'insieme di stati non deve essere finito, permettendo quindi di modellare sistemi che includono strutture dati illimitate come contatori.

Funzione di generazione dei messaggi specifica, per ogni stato e vicino in uscita, il messaggio (o *null*) che il processo i invia al vicino indicato, a partire dallo stato dato.

Funzione di transizione di stato specifica, per ogni stato e collezione di messaggi da tutti i vicini in entrata, il nuovo stato verso cui i si sposta.

Canale (o link) una locazione associata a ogni arco (i, j) che può, in ogni momento, contenere al massimo un singolo messaggio di M.

L'esecuzione dell'intero sistema inizia con tutti i processi in stati iniziali arbitrari, e con tutti i canali vuoti. Poi i processi, insieme e passo per passo, eseguono i seguenti due passi:

1. Applicano la funzione di generazione dei messaggi allo stato corrente per generare i messaggi da inviare ai vicini in uscita, e mettono questi messaggi nei canali appropriati.

2. Applicano la funzione di transizione di stato allo stato corrente e ai messaggi in entrata per ottenere il nuovo stato, rimuovendo i messaggi dai canali.

La combinazione dei due passi è chiamata *round*. Notare che, in generale, non vengono imposte restrizioni sull'ammontare di computazione che un processo deve eseguire per computare i valori delle sue funzioni di generazione dei messaggi e di transizione di stato.

Notare anche che il modello presentato è deterministico, nel seno che le funzioni di generazione dei messaggi e di transizione di stato hanno valori singoli. Per cui, a partire da una particolare combinazione di stati iniziali, la computazione si svolge in modo univoco.

Stati di halt L'halt dei process può essere tenuto in considerazione in questo modello designando alcuni stati dei processi come *stati di halt*, e specificando che da questi stati non ci può più essere alcuna attività, ovvero: nessun messaggio viene più generato, e l'unica transizione possibile è un loop verso lo stato stesso. Questi stati non svolgono lo stesso ruolo che hanno negli automi finiti, e *non* sono quindi considerati *di accettazione*, ma hanno il solo scopo di interrompere il processo. Ciò che viene computato dal processo deve essere determinato in un altro modo.

Nodo e processo di ambiente Occasionalmente, si vuole considerare un sistema sincrono in cui i processi si avviano in diversi round. Si modella questa situazione estendendo il grafo di rete per includere uno speciale *nodo di ambiente*, che ha archi verso tutti i nodi ordinari. Lo scopo del corrispondente *processo di ambiente* è di inviare degli speciali *messaggi di wakeup* a tutti gli altri processi. Ogni stato inziale di ognuno degli altri processi deve essere *quiesciente*, che significa che non causa la generazione di messaggi, e che può essere cambiato a uno stato diverso solo in seguito

alla ricezione di un messaggio di wakeup dall'ambiente o da un messaggio non nullo da un altro precesso. Per cui, un processo può essere svegliato:

- **direttamente**, da un messaggio di wakeup dall'ambiente
- **indirettamente**, da un messaggio non nullo da un altro processo.

Grafi non orientati A volte si vuole considerare il caso in cui il grafo di rete non è diretto. Questa situazione viene modellata considerando un grafo diretto con archi bidirezionali tra tutte le coppie di vicini. In questo caso, si usa la notazione $nbrs_i$ per indicare i vicini di i nel grafo.

Fallimento

Si considerano diversi tipi di fallimento nei sistemi sincroni, tra cui *fallimento dei processi* e *fallimento dei collegamenti*.

Un processo può esibire uno *stopping failure* semplicemente interrompendosi nel mezzo della sua esecuzione. Nei termini del modello, il processo potrebbe fallire prima o dopo aver eseguito una qualche istanza del Passo 1 o Passo 2 descritti prima. In aggiunta, potrebbe interrompersi nel mezzo del Passo 1. Questo significa che il processo potrebbe riuscire a mettere nei canali solo un sottoinsieme dei messaggi che dovrebbe produrre. Si da per scontato che questo possa essere un *qualunque sottoinsieme* - non si considera che il processo produce messaggi sequenzialmente e che fallisce nel mezzo della sequenza.

Un processo può anche esibire un *fallimento bizantino*, con cui si intende che può generare i suoi prossimi messaggi e il suo prossimo stato in maniera arbitraria, senza necessariamente seguire le regole stabilite dalle funzioni di generazione dei messaggi e di transizione di stato.

Un collegamento può fallire perdendo messaggi. Nei termini del modello, un processo può tentare di inserire un messaggio nel canale durante il Passo 1, mentre il collegamento fallace non lo registra.

Input e output

Si usa la convenzione di codificare gli input e gli output negli stati. In particolare, gli input sono inseriti in variabili designate negli stati iniziali. Il fatto che il processo possa avere diversi stati iniziali è importante qui, perché permette di accomodare diversi input. Di fatto, si da normalmente per scontato che l'unica fonte di molteplicità di stati iniziali è la possibilità di avere diversi valori di input nelle variabili di input.

Esecuzioni

Per ragionare sul comportamento di un sistema di rete sincrono, è necessaria una notazione formale dell'esecuzione di un sistema.

- Un *assegnazione di stato* C_i di un sistema è definita come l'assegnazione di uno stato a ogni processo del sistema.
- Un *assegnazione di messaggio* è un'assegnazione di messaggi (possibilmente *null*) a ogni canale del sistema.

Un'esecuzione del sistema è definita come una sequenza infinita

$$C_0, M_1, N_1, C_1, M_2, N_2, C_2, \ldots$$