Approfondimento di Sistemi Distribuiti

Autore: Lionello Marco matricola: 810079

15 Maggio 2006

Indice

Elenco delle figure

Capitolo 1

Introduzione

In questo approfondimento si introduce il problema del consenso, dei generali bizantini e della consistenza iterattiva. Questi tre problemi vengono considerati come problemi di agreement (accordo). Il problema, spiegato in maniera rozza, è di trovare un comune accordo su di una variabile dopo che uno o più processi propongono un valore per quella variabile. Nel terzo capitolo viene affrontato in maggior dettaglio il problema dei generali bizzantini e in particolare il problema della decisione di far attaccare o far ritirare consistentemente due armate. Un esempio di problema di agreement può essere la coerenza di una transazione bancaria nella quale i rispettivi computer devono essere consistentemente d'accordo sulle operazioni (una di credito e una di debito). Un altro esempio è quello della mutua esclusione nella quale un processo entra nella sezione critica (i computer devono essere anche in questo in maniera consistentemente d'accordo). Infine, ad esempio, in una elezione, i processi concordano su quale processo deve venire eletto. Protocolli che si adattano al particolare tipo di problema di agreement esistono e vengono illustrati nel capitolo 13 e 14 del Coulouris[2005]. E' utile ad ogni modo ricercare le caratteristiche e soluzioni comuni. Nel secondo capitolo si approfondirà il problema del consenso e si relazionerà con i tre problemi: il generale bizantino, consistenza interattiva e multicast ordinato totale.

Capitolo 2

Modelli di sistema e definizione del problema

Siano *pi* (i=1,2,.,N) processi comunicanti tramite il message passing. Un importante obbiettivo che si prefigge il consenso, è che questo deve essere ottenuto anche se c'é qualche processo difettoso. Si assume quindi che la comunicazione sia certa mentre i processi possono essere difettosi. Si assume inoltre che sono f di questi siano difettosi (anche con crash) mentre il resto sono corretti. Se un difetto occore nel nostro sistema il processo segna il messaggio che manda. In questo modo siamo in grado di limitare il danno che il processo può fare. In maniera specifica si può dire che durante un algoritmo di accordo il processo non può fare falsi reclami su di un valore che un processo corretto ha mandato a lui. Questo sistema ci tornerà utile nel problema del generale bizantino.

2.1 Definizione del problema del consenso

Per raggiungere il consenso ogni processo *pi* comincia nel suo stato di undecided e successivamente propone un singolo valore *vi*, da un insieme *D*. Il processo comunica con gli altri, scambia valori. Ogni processo pone il valore della decision variable *di*.

La figura 2.1 mostra tre processi agganciati nell'algoritmo del consenso. Nella figura si vedono tre processi: due corretti che propongono

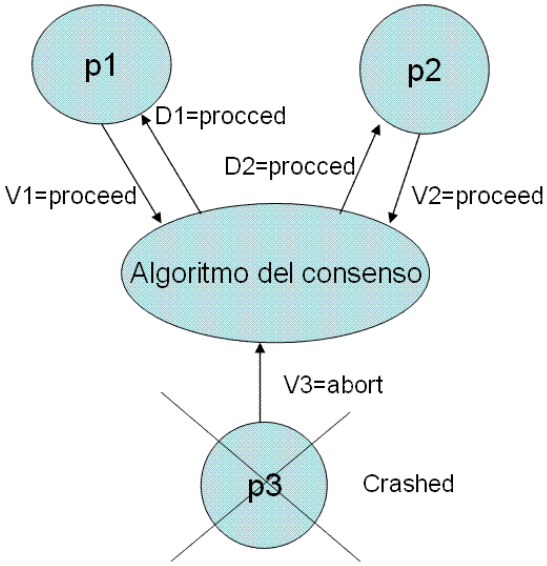

Figura 2.1: Esempio di tre processi

procede, uno diettoso che propone *abort* e poi va in crash. I due processi rimasti corretti decidono entrambi *procede*. Le richieste fondamentali di un algoritmo del consenso sono :

- Termination: Ogni processo corretto pone la sua variabile di decisione
- Agreement: La variabile di decisione di tutti i processi corretti è la stessa
- Inegrity: se tutti i processi corretti propongono lo stesso valore, allora ogni processo corretto nello stato di decisione ha scelto quel valore.

Una nota va fatta sulla nozione di integrità. Essa si può variare introducendo una soglia anzichè forzare il fatto che tutti devono aver proposto quel determinato valore. In questo approfondimento si usa la definizione. Per aiutarci nel capire il problema si assuma che il sistema non abbia processi difettosi. Per esempio possiamo collezionare i processi in un gruppo e avere tutti i processi comunicanti tramite un multicast certo i quali propongono un valore per i membri del gruppo. I processi attendono fino a quando tutti N i valori sono stati proposti incluso il processo stesso. I processi valutano la funzione *mayority(v1,v2,,vn)* che ritorna il valore che occorre più spesso o il valore `?` che non appartiene a D se mayority non esiste. La condizione di termination è garantita dalla certezza delle operazioni di mulicast. Agreement e Integrity sono garantite dalla definizione di majority, e integity anche dalla certezza del multicast. Ogni processo riceve lo stesso insieme di valori proposti, e ogni processo valuta la stessa funzione. In questo modo tutti concordano lo stesso valore. La funzione di decisone può essere anche diversa ad esempio minimo o massimo. Se si introducono i processi difettosi si introduce il problema di determinare il processo difettoso. La condizione di terminazione non è più immediata. Ad esempio, in sistemi asincroni la condizione di terminazione diventa un forse (may be). Se un processo fallisce arbitrariamente, può comunicare valori arbitrari agli altri. Questo può sembrare poco pratico, ma non se si tiene conto dei bug dei processi. Un fallimento inoltre può

essere causato da operazioni maligne o malevolenti. Qualcuno può deliberatamente creare un processo per mandare valori agli altri peers per contrastare gli altri, che cercano di arricare al consenso. In caso di inconsistenza, i processi corretti devono comparere quello che hanno ricevuto con quello che gli altri dichiarano di aver ricevuto.

2.2 Il problema del generale Bizantino

Nel problema dei generali bizantini tre o più generali devono concordarsi nell'attaccare o nel ritirarsi. Uno, il comandante impartisce i comandi. Gli altri, tenenti del comandante, devono decidere se attaccare o ritirarsi. Uno o più dei generali può essere perfido (ossia un processo difettoso). Se il comandante è perfido dice a un generale di attaccare e a un altro di ritirasi. Se un tenente è perfido, dice a uno dei suoi peers che il comandante ha detto di attaccare e agli altri di ritirasi. Il problema dei generali bizantini differisce dal problema del consenso in quanto un processo distinfo rifornisce un valore che gli altri devono concordare, invece che tutti propongano un valore. I fattori che si richiedono sono: *procede*. Le richieste fondamentali per il problema del generale Bizantino sono :

- Termination: Tutti gli eventuali processi corretti settano la loro *variabile decisionale*

- Agreement: la *variabile decisionale* di tutti i processi corretti è la stessa se *pi* e *pj* sono corretti e hanno deciso lo stato allora *di=dj* per ogni *i,j*

- Integrity: se il comandante è corretto, allora tutti i processi corretti decidono sul valore che il comandante ha deciso.

Si nota che , per il problema del generale bizantino, l'integrità implica l'accordo solo se il comandante è corretto.

2.3 Consistenza iterativa

Il problema della consistenza iterativa è un'altra variante del consenso, nella quale tutti i processi propongono un valore. Il goal dell'algoritmo è per i processi corretti di accordarsi in un vettore di valori uno per ogni processo. Chiameremo questo vettore *decision vector*. Per esempio il goal può essere per ogni insieme di processi ottenere le stesse informazioni riguardo i rispettivi stati. Le richieste fondamentali per il problema della consistenza iterativa sono :

- Termination: Tutti gli eventuali processi corretti settano la loro *variabile decisionale*
- Agreement: il *decision vector* di tutti i processi corretti lo stesso
- Integrity: Se *pi* è corretto, allora tutti i processi corretti decidono su *vi* come i-esimo componente del loro vettore.

2.4 Relazionare il consenso ad altri problemi

E' giusto considerare i problemi del consenso, dei generali bizantini e della consistenza interattiva come problemi con processi difettosi. E' inoltre necessario frammentare i problemi in due sottocategorie : in sistemi sincroni e asincroni. E' a volte possibile passare da soluzioni di un problema a soluzioni di un altro. Questo risulta possibile anche nel problema del consenso. Consideriamo una soluzione al problema del consenso che chiameremo C, una per il problema dei generali bizantini BG e una per la consistenza iterativa IC come segue:

- `Ci(v1,v2,,vn)` ritorna la decision value di *pi* nella soluzione di una esecuzione del problema del consenso, dove *vi* sono i valori che i processi hanno proposto.
- `BGi(j,v)` ritorna la decision value di *pi* nella soluzione di una esecuzione del problema dei generali bizantini , dove *pj* il comandante propone il valore *v*

- `ICi(v1,vn)[j]` ritorna il j-esimo valore nel decision vector di *pi* nella soluzione dell'esecuzione del problema della consistenza iterativa, dove *v1,.,vn* sono valori che i processi propongono

La definizione di Ci,BGi e ICi assume che che i processi difettosi propongono un singolo valore rotazionale. Questa è solo una convenzione, che non influenza il problema. E' possibile passare da una soluzione di un problema alla soluzione di un altro problema:

- IC da BG: Costruiamo una soluzione a IC da BG facendo eseguire BG N volte una per ogni processo *pi* usandola come comandante: ICi(v1,,vn)[j] = BGi(j,vj) (i,j=1.N)
- C da IC: Per il caso dove majority dei processi è corretta, si costruisce una soluzione di C da IC eseguendo IC per produrre un vettore di valori a ogni processo, che applicato alla funzione appropriata nel vector's value per derivare un singolo valore Ci(v1,,vn) = majority(ICi(v1,..,vn)[1],....., ICi(v1,..,vn)[N]) per ogni i
- BG da C: Costruiamo una soluzione di BG da C come descritto qui sotto:

  1.
  2. Il commandante *pj* manda una proposta di valore *v* a se stesso e a ogni altro processo
  3. Tutti i processi lanciano C con i valori *v1,v2,.,vn* che ricevono (*pj* potrebbe essere caduto)
  4. I processi derivano BGi(j,v)=C(v1,..,vn) per ogni i

I tre problemi dimostrati da Fisher nel 1983 così dimostrati presevano le tre propriet: terminazione, accordo, integrità. Nei sistemi con difetti di crash il problema è equivalente a risolvere problemi di multicast ordinato totale: risolvere uno è equivalente a risolvere l'altro. Implementare il consenso con RTO-Multicast (total ordered multicast operation) diretto. Per raggiungere il consenso ogni processo pi fa una RTO-Multicast(g,vi). Quindi ogni processo *pi* sceglie *di=mi*, dove *mi* è il primo valore che

pi RTO-delivers (consegna). La terminazione è garantita dalla certezza del multicast. Le propriet di agreement e integrità sono rispettate dalla certezza e dalla ordinazione totale della consegna multicast Chandra and Toueg[1996].

Capitolo 3

Consenso in sistemi sincroni

Questa sezione mostra un'algoritmo che fa uso di un basilare protocollo multicast per risolvere il problema del consenso. L'algorimo assume che al massimo f di N processi possono andare in crash. Per raggiungere il consenso ogni processo corretto colleziona i valori proposti dagli altri processi. L'algoritmo procede in f+1 giri, in ogni uno dei quali i processi corretti B-Multicast i valori tra loro. Nel caso pessimo , tutti gli f crash occorrono durante il giro , ma l'algoritmo garantisce che alla fine del giro tutti i corretti processi che sono sopravissuti sono in posizione di accordo.

L'algoritmo mostrato in figura 3.1 è stato presentato nel 1998 da Attiya e Welch . La variabile *Values* tiene l'insieme dei valori proposti conosciuti dal processo *pi* all'inizio del giro *r*. Ogni processo multicasts l'insieme di valori che non ha mandato nel giro precedente. In questo si prende consegna dei messaggi multicast simili da un altro processo e annota i nuovi valori. La durata di un giro è limitata settando un timeout sul massimo tempo che un processo corretto ci impiega a fare un multicast di un messaggio. Dopo f+1 giri, ogni processo sceglie il valore minimo che ha ricevuto come valore per la sua decision value. La terminazione è ovvia perchè il sistema è sincrono. Per controllare la correttezza dell'algoritmo, dobbiamo mostrare che ogni processo arriva allo stesso insieme di valori alla fine dell'ultimo giro. Agreement e Integrity seguono, perchè il processo applica la funzione *minimum* al suo insieme. Assumimo per assurdo che due processi differiscono nel loro insieme finale di

Consensus in a synchronous system

Algorithm for process $p_i \in g$; algorithm proceeds in $f + 1$ rounds

On initialization

$Values_i^1 := \{v_i\}$; $Values_i^0 = \{\}$;

In round r $(1 \leq r \leq f + 1)$

B-multicast$(g, Values_i^r - Values_i^{r-1})$; // Send only values that have not been sent

$Values_i^{r+1} := Values_i^r$;

while (in round r)

{

On B-deliver(V_j) *from some* p_j

$Values_i^{r+1} := Values_i^{r+1} \cup V_j$;

}

After $(f + 1)$ *rounds*

Assign d_i = *minimum*$(Values_i^{f+1})$;

Figura 3.1: Algoritmo di Attiya and Welch

valori. Senza perdita di generalità, alcuni processi corretti pi posseggono valori v che altri processi correti pj con i diverso da j non posseggono. L'unica spiegazione è che un terzo processo pk che gestiva la spedizione di v a pj sia andato in crash prima che v arrivasse a pj. Questo significa che per ogni processo che spediva v deve essere andato in crash nel giro precedente. Procedendo in questo modo si ha al massimo un crash per ogni giro. Ma per assunzione abbiamo detto che ci possono essere al massimo f crash e ci sono f+1 giri. Siamo arrivati ad un assurdo. In altre parole abbiamo dimostrato che qualsiasi algoritmo che arriva al consenso necessita come minimo di f+1 giri di scambi di messaggi e non importa come questo viene implementato.

3.1 Il problema del generale bizantino in un sistema sincrono

Si discute il problema dei geneali bizantini in sistemi sincroni. In aggiunta al problema visto prima qui si considera la possibilità che i processi possono avere difetti arbitrari (es. si spediscono messaggi senza valore, con valore sbagliato,o addirittura non spediscono messaggi). Come prima al massimo f di N possono essere i processi difettosi. I processi corretti possono capire quando c'è un'assenza di messaggio tramite il timeout, ma non possono concludere che il processo è andato in crash, in quanto il processo difettoso potrebbe rimanere silenzioso per un lasco di tempo e poi ricominciare a spedire messaggi. Un'ulteriore assunzione è che il canale di comunicazione tra due processi sia privato. Se un processo può esaminare tutti i messaggi degli altri processi allora pu capire se c'è inconsistenza. La nostra assunzione principale è che la certezza del canale comporta che nessun processo difettoso può iniettare messaggi nel canale tra due processi corretti. Lamport[1982] considera il caso di tre processi che mandano messaggi unsigned a un'altro. Si è mostrato che non esiste soluzione per garantire le condizioni del problema di generali bizantini se a uno dei processi è permesso di fallire. Questa cosa è stata generalizzata dicendo che non c'è soluzione esistente se $N \leq 3f$. Dimostriamo questo

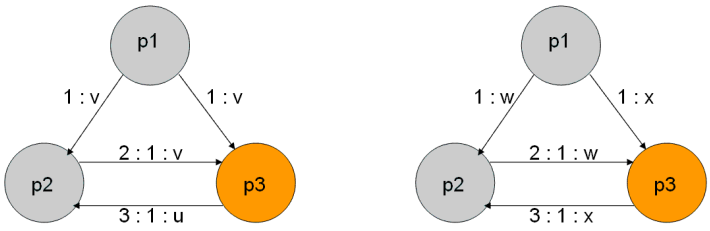

Figura 3.2: Problema dei generali bizantini con tre processi

brevemente. Un algoritmo che risolve il problema dei generali bizantini in sistemi sincroni con $N \geq 3f + 1$ per messaggi unsigned è *oral*.

3.2 Impossibilità con tre processi

Figura 3.2 mostra due scenari nei quali un processo è difettoso. Nella configurazione di sinistra il tenente *p3* è difettoso; nella destra il comandante *p1* è difettoso. Entranbi gli scenari mostrano due giri di messaggi: il primo che è il messaggio che il comandante manda ai tenenti, e il secondo del tenente all'altro tenente. Il prefisso numerico serve a specificare la sorgente del messaggio e a distinguere i vari giri. Il simbolo : può essere interpretato come la parola 'dice'.

Nel scenario di sinistra. Il comandante correttamente manda lo stesso messaggio a *p2* e *p3* e successivamente *p2* fa un eco corretto a *p3*. Purtroppo *p3* manda a *p2* il valore sbagliato. A questo punto *p2* si accorge dell'errore ma non può stabilire quale sia il processo difettoso. Nel scenario di destra il comandante manda due differenti valori ai tenenti. Dopo che *p3* ha fatto un echo a *p2*, *p2* si ritrova nella stessa situazione di prima. Se una soluzione esiste, allora il processo *p2* è limitato nel decidere sul valore di v quando il comandante è corretto, per la condizione di integrità. Se decidiamo che nessun algoritmo può distinguere tra due scenari, *p2* deve scegliere il valore mandato dal comandante nello scenario di destra. Di conseguenza per le stesse ragioni *p3* sceglie v

dal comandante. Questo però contraddice la condizione di agreement. Quindi nessuna soluzione è possibile. Nota che questa spiegazione resti nella nostra intuizione che niente può essere fatto per migliorare la corretta conoscenza del generale oltre il primo passo, dove non si può dire quale processo è difettoso. E' possibile provare la correttezza di questa intuizione Pease[1980]. L'accordo bizantino pu essere raggiunto per i tre generali, con uno di questi difettoso, se il generale firma i suoi messaggi.

3.3 Soluzione con un processo caduto

Non c'è spazio per descrivere l'algoritmo di Pease. Si descrive qui l'algoritmo per il caso $N \geq 4$ e f=1 e lo illustriamo nel caso N=4 e f=1. I generali corretti raggiungono l'accordo in due giri di messaggi:

- Nel primo giro il comandante manda un valore a ogni tenente
- Nel secondo ogni tenente manda il valore che ha ricevuto ai suoi peers

Un tenente riceve il valore dal comandante pi N-2 valori dai suoi peers. Se il comandante è difettoso, allora tutti i tenenti sono corretti e ogni uno di questi ha raccolto l'esatto valore che il comandante gli ha spedito. Altrimenti uno dei tenenti è difettoso, ogni tenente riceve N-2 messaggi contenenti il valore che il comandante ha spedito più un valore del tenente difettoso. In entrambi i casi, il tenente corretto deve applicare una semplice funzione di majority che ignora che ignora il valore spedito dal tenente difettoso.

Nel caso dei 4 generali l'esempio dell'algorimo viene mostrato in figura 3.3. In essa sono presenti due scenari simili a quelli di figura 3.3. Nello scenario di sinistra il tenente *p3* è difettoso nello scenario di destra il comandante *p1* è difettoso. Nel caso di sinistra i due tenenti corretti decidono in base al valore del comandante:

- *P2* decide in base a majority v
- *P4* decide in base a majority v

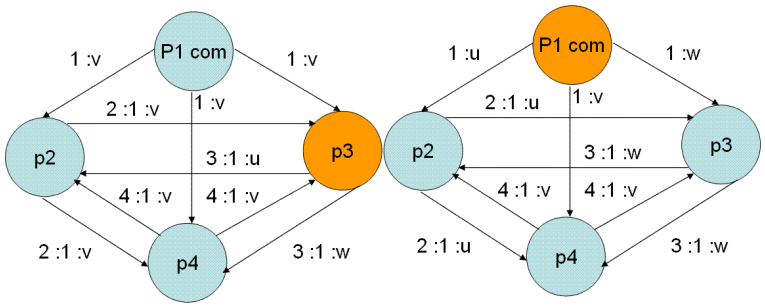

Figura 3.3: Problema dei generali bizantini con quattro processi

Nel caso di destra il comandante è difettoso ma i tre processi tenenti si accordano *p2,p3,p4* decidono in base a majority = ? ossia non esiste maggioranza L'algoritmo tiene conto del fatto che un processo difettoso può omettere l'invio di un messaggio. Se un processo corretto non riceve un messaggio entro un tempo massimo (sistema sincrono) procede come se il processo difettoso avesse spedito ?.

3.4 Discussione

Possiamo chiederci una misura di efficienza nel caso del problema del generale bizantino o di altri problemi di agreement chiedendoci:

- Quanti giri di messaggi si devono fare? (questo è un fattore che determina la lunghezza di esecuzione sell'algoritmo)
- Quanti messaggi sono stati inviati, qual è la loro dimensione?(questa è una misura del bandwhith il quale ha un inpatto sul tempo di esecuzione)

Nel caso ($f \geq 1$) l'algoritmo di lamport di messaggi unsigned opera su f+1 giri. Ad ogni giro un processo manda al suo sottoinsieme di processi il valore che ha ricevuto nel giro precedente. L'algoritmo è veramente costoso. Esso coinvolge la spedizione di di $\Theta(N^{f+1})$ messaggi. Fisher and Lynch[1982] hanno provato che ogni soluzione deterministica al problema

del consenso assumendo fallimenti arbitrari prende al minimo f+1 giri di messaggi. Quindi nessun algoritmo può operare più veloce di lamport. Ma può esserci un miglioramento nella complessit dei messaggi Garray and Mosses[1993]. L'algoritmo di [Dolev and Strong's] prende ancora f+1 giri ma il numero di messaggi spediti sono solo $\Theta(N^2)$. La complessità e il costo della soluzione suggerisce che questo è applicabile solo quando la minaccia è grande. Se un hadware difettoso è la causa della minaccia, allora la probabilit di un vero comportamento arbitrario è bassa. Alcune modelli di gasto si possono trovare in Barborak[1993]. Se utenti maliziosi sono la causa di minaccia allora un sistema per eliminarli è l'uso della firma elettronica, senza di questa non esiste soluzione.

Capitolo 4

Impossibilità nei problemi asincroni

Si è fornito in questo approfondimento una soluzione al problema del consenso e dei generali bizantini nei sistemi sincroni. Fisher[1985] ha dimostrato che non si può raggiungere il consenso in un sistema asincrono. Nei sistemi asincroni i processi possono rispondere in un tempo arbitrario, quindi è indistinguibile il crash dal ritardo. Si può affermare inoltre che non esiste garanzia nel problema dei generali bizantini, del consenso iterativo, o nel problema del multicast ordinato totale certo. Nota che la parola *garanzia* è un'affermazione di impossibilità.

Bibliografia

[1] Attiya, H. and Welch, J. (1998), *Distribuited Computing Fundamentals, Simulations and Advanced Topics.*,
McGraw-Hill.

[2] Chandra, T. and Toueg, S. (1996)., *Unrealable failure detectors for reliable distributed system.*,
Journal of the ACM, Apr., pp. 374-82

[3] Coulouris, G. and Dollimore, J. And Kindberg, T. (2005),
Distribuited System Concepts and Design.,
4th edition, Addison Wesley Masson

[4] Dolev, D. and Strong, H. (1983)., *Authenticated algorithms for byzantine agreement.*,
SIAM Journal of Computing, Vol. 12, No. 4, pp. 656-66

[5] Fisher, M. and Lynch, N. (1982)., *A lower bound for the time to assure interactive consistency.*,
Inf. Process. Letters, Vol. 14, No 4,June, pp. 183-6

[6] Fisher, M.,Lynch, N. and Paterson, M. (1985)., *Impossibility of distribuited consensus with one faulty process.*,
Journal of ACM, Vol. 32, No. 2 Apr., pp.34

[7] Pease, M., Shostack,R. and Lamport, L. (1980)., *Reaching agreement in the presence of faults.*,
Journal of the ACM, Vol27, No. 2, April, pp. 228-34