Reti asincrone

Problema del consenso in caso di malfunzionamenti

Borello Nazareno, Galatola Marco, Mecca Francesco

Importanza del problema

“Before this paper, it was generally assumed that a three-processor system could tolerate one faulty processor.”

Marshall Pease, Robert Shostak, Leslie Lamport Journal of the Association for Computing Machinery 27 | April 1980, Vol 2

“Over the years, I often wondered whether the people who actually build airplanes know about this problem. In 1997, I received email from John Morgan who used to work at Boeing. He told me that he came across our work in 1986 and that, as a result, the people who build the passenger planes at Boeing are aware of the problem and design their systems accordingly.”

2005 Edsger W. Dijkstra Prize in Distributed Computing

Problema del consenso

Input:

$init(v)_i$: viene inizializzata una variabile interna con il valore scelto

$stop_i$: modella il fallimento di un processo

Output:

$decide(v)_i$: viene offerto il consenso per il valore v

$v \in V,\ 1 \leq i \leq n$

Proprietà

- **Agreement**: tutti i processi decidono lo stesso valore
- **Validity**: se viene effettuata una *init* con lo stesso valore *v* su ogni processo allora *v* sarà l'unico valore possibile per l'operazione *decide*
- **Well Formedness**: ogni sequenza di $init_i$ e $decide_i$ è un prefisso della sequenza $init(v)_i, decide(w)_i$

Terminazione

- **Failure-free**: in ogni esecuzione fair e failure-free ogni processo effettua una *decide*
- **F-failure** ($0 \leq f \leq n$): in ogni esecuzione fair in cui gli eventi di *init* avvengono su tutte le porte, se ci sono al massimo f eventi di *stop*, gli eventi *decide* avvengono per tutti i processi sani
- **Wait-free**: f-failure termination con $f = n$

Impossibilità di consenso

- Non esiste un algoritmo, per il modello broadcast asincrono con canali sicuri, che risolva il problema del consenso e garantisca *1-failure termination*
- La dimostrazione di basa su:
  - il teorema di impossibilità per il modello a memoria condivisa
  - l'esistenza di una trasformazione dal modello broadcast asincrono al modello con memoria condivisa
- Lo stesso risultato si ottiene per il modello asincrono con send/receive dimostrandone l'equivalenza computazionale rispetto al modello broadcast

Equivalenza modelli

- A: modello di rete asincrona con broadcast
- B: modello di rete asincrona con send / receive FIFO
- A e B hanno la stessa interfaccia
- B simula A $\rightarrow \forall$ esecuzione α di B, $\exists$ α' di A tale che:
  - α ed α' sono indistinguibili per U (insieme utenti)
  - $\forall$ processo i, in α avviene una $stop_i$ se e solo se avviene in α'
  - Se α è fair lo è anche α'

Possibili soluzioni

Sono state proposte varie soluzioni per risolvere il problema del consenso in presenza di malfunzionamenti, tra le più importanti:

- algoritmi randomizzati
- failure-detector
- k-agreement

Algoritmi randomizzati

- Alcune scelte vengono effettuate in maniera probabilistica, utilizzando la funzione *RANDOM(A)* che restituisce con probabilità uniforme un elemento dell'insieme A scelto casualmente
- Rilassiamo le condizioni di correttezza: la terminazione è ora probabilistica.
- I processi sani eseguiranno $decide_i(v)$ ad un tempo t con probabilità $p(t)$ dove p è una funzione monotona, non decrescente e non limitata
- Basso costo computazionale

Algoritmo di Ben-Or

- Ogni processo ha due variabili locali: $x = null$, $y = null$
- Il valore di decisione appartiene a {0, 1}
- Supponendo che f sia il numero di processi che falliscono, occorrono almeno $n > 3f$ processi
- L'algoritmo esegue una serie potenzialmente infinita di iterazioni, ognuna delle quali consiste di due fasi

Algoritmo - Pseudocodice

/* FASE 1 */

send(x, r) // r è il timestap

for all i, $0 \leq i \leq (n - f)$ **do**

wait(S_i, r) / S = insieme dei valori ricevuti */*

if $\exists\ v \in \{0,1\} | \forall\ s \in S, s = v$ **then**

$y \leftarrow s$

/* FASE 2 */

send(y, r)

for all i, $0 \leq i \leq (n - f)$ **do**

wait(S_i, r)

if $\forall\ s \in S, \nexists\ s' \in S \mid s' = s$ **then**

$x \leftarrow s$

decide(s) // se non ha mai deciso prima

else if $\exists$ S' $\subset$ S, $|S'| \geq n\text{-}2f \mid \forall\ s', s'' \in S', s' = s''$ **then**

$x \leftarrow s$ *// almeno n - 2f valori identici*

else $x \leftarrow$ *random* {0; 1} // x scelto *casualmente*

Esempio Ben-Or (animazione)

Validita` dell’algoritmo

- Well Formedness:

  una volta eseguita un’azione di decide, questa non si ripete mai.

- Validity:

  immaginiamo che nella fase 1 tutti i processi ricevono lo stesso valore *v*: allora in fase 2 tutti i processi compieranno l’azione di decide con *v*.

- Agreement:

  Supponiamo che il processo P_i decide per primo all’iterazione *s:* questo significa che ha ricevuto *n - f* messagi con valori identici *v*. Tutti gli altri processi hanno ricevuto almeno n - 2f contenente il valore v. Quindi I rimanenti processi sani P_j possono esclusivamente:

  - eseguire *decide(v)*;
  - settare *x = v* e compiere l’azione di *decide* alla successiva iterazione

Un’esecuzione senza decisioni

- Abbiamo un numero di processi *m: $f < m \leq 2f$, $n = 3f+1$*
- Gli *m* processi hanno $x = 0$ e i restanti $x = 1$
- Alla fine della fase 1 tutti i processi hanno $y = null$ e alla fase 2 tutti i processi scelgono il proprio valore di *x* casualmente
- Se pensiamo che questa decisione casuale ci porta ad una situazione in cui esiste un numero *m'* di processi: $f < m' <= 2f$ dove $x = 0$ ed i restanti hanno $x=1$
- Torniamo alla situazione iniziale che si puo` ripetere all'infinito senza portare ad una decisione

Terminazione

- Caso $s = 0$: $p(t) \geq 0$, sicuramente c'e` una probabilita` non nulla che tutti i processi sani decidano alla prossima iterazione.
- Caso $s \geq 1$: consideriamo un processo sano P_i che riceve almeno $n-f$ messaggi del tipo *("phase 1", s, *)*
  - Se almeno $f + 1$ messaggi contengono lo stesso valore v, chiameremo v valore "buono": possono esistere al piu` due valori buoni.

Terminazione #2

- Nel caso ci sia un solo valore “buono”, ogni messaggio inviato nella fase due *(“phase 2”, s, *)* conterra` lo stesso valore *v* o *null*.
- I processi che non impostano *x* = *v* allora devono scegliere il valore casualmente.
- La probabilita` che i restanti processi scelgano lo stesso valore e` **almeno** $\frac{1}{2^n}$

Terminazione #3

- Con due valori “buoni” i processi sani riceveranno sia il valore 0 che il valore 1 ed invieranno nella seconda fase il messaggio:

  (“phase 2”, s, null)

- Di nuovo, la probabilita` che tutti i processi scelgano lo stesso valore e` $\frac{1}{2^n}$

Terminazione #4

- Per tutti i processi sani, la probabilita` che venga scelto lo stesso valore e` almeno $\frac{1}{2^n}$
- La probabilita` che non venga compiuta una scelta unica in una sola iterazione e`:

$$\textbf{p(nessuna scelta unica)} \leq (1 - \tfrac{1}{2^n})$$

- Poiche` ogni iterazione e` indipendente, otteniamo che:

$$\textbf{p(nessuna scelta unica nelle prime s iterazioni)} \leq (1 - \tfrac{1}{2^n})^s$$

- Ne deriva che i processi sani avranno deciso lo stesso valore v prima della fase s+1 come:

$$\textbf{p(unico valore deciso prima della fase s+1)} \geq 1 - (1 - \tfrac{1}{2^n})^s$$

Terminazione - risultato

- Per ogni iterazione s ≥ 0 vogliamo dimostrare che tutti i processi sani compiono un'azione di *decide(v)* entro l'iterazione *s+1* con probabilita` *p(t)* per cui:

$$p(t) \geq 1 - (1 - \frac{1}{2^n})$$

- Per infinite iterazioni:

$$s \rightarrow \infty: p = 1$$

- L'algoritmo di Ben-Or garantisce la terminazione con probabilita` $p(t) = 1$

Probabilità 1 ≠ Certezza

- Se un evento ha probabilità 1 di verificarsi possiamo affermare che avverrà quasi sicuramente ma non ne abbiamo la certezza matematica

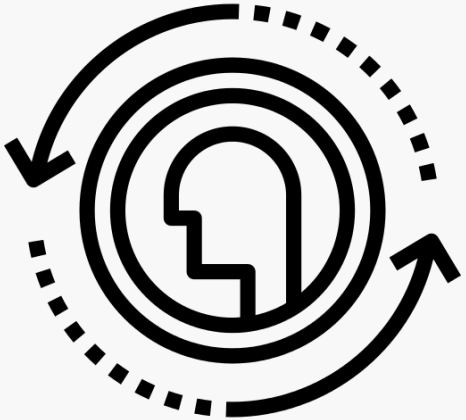
- **Esempio significativo**:

  Supponiamo di eseguire infiniti lanci di una moneta

  La probabilità che esca solo croce equivale a 0 ($p = (½)^{\infty}$)

  La probabilità che esca almeno una volta testa equivale a 1

  Esiste comunque la possibilità di ottenere una sequenza composta da sole croci



Complessita` della Terminazione

- In ogni esecuzione fair in cui gli eventi di init avvengono su tutte le porte ogni processo che non fallisce esegue infiniti step
- *d* è l'upper bound per il delivery time del broadcast del messaggio più vecchio
- *l* è l'upper bound del tempo necessario per il completamento del task di ogni processo
- Si può dire che ogni processo completa ogni volta un'iterazione *s* in $O(s(d+l))$ dall'ultimo evento di init.

Efficienza dell'Algoritmo

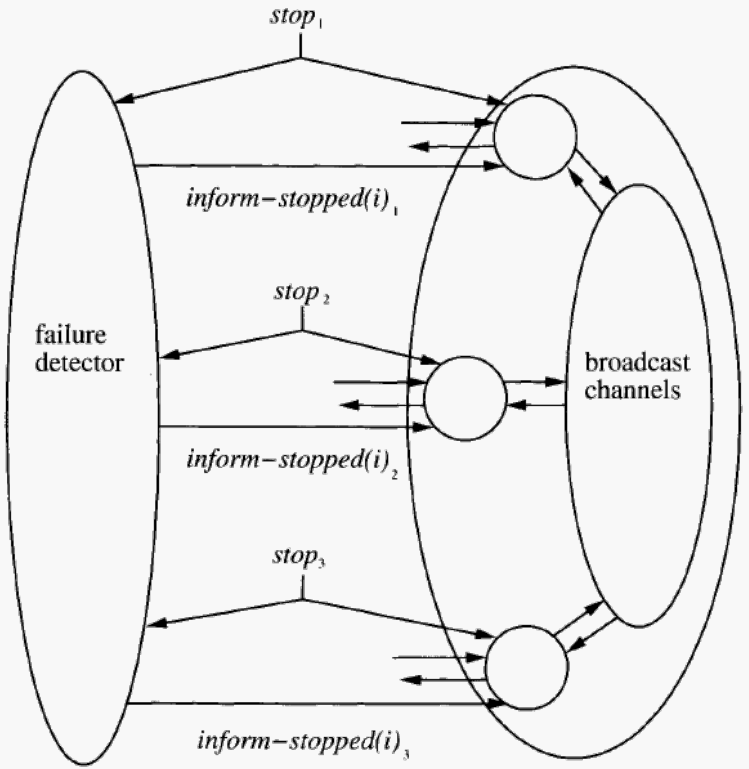
- Immaginiamoci di avere un generale bizantino che agisce in maniera malevole ed in ogni istante conosce:
  - stato e contenuto dei messaggi degli altri generali
  - le decisioni e le mosse degli altri generali
  - il valore dei lanci casuali di una qualsiasi moneta
- Questo si definisce formalmente come "Adversary Model".
- Si dimostra che l'algoritmo di Ben-Or risolve il problema del consenso bizantino anche in presenza di un avversario "onnisciente" (**adaptative offline adversary**)

Global Coin Variant

- Nell'algoritmo di Ben-Or un processo può scegliere all'iterazione *s* un valore di *x* casuale (modellato come il lancio indipendente di una moneta {0; 1} per processo)
- Se tutti i processi "estraggono" lo stesso valore, allora la decisione e la terminazione avviene all'iterazione successiva
- Possiamo immaginarci l'esistenza di una sola moneta, detta Global Coin, che tutti i processi lanciano allo stesso istante nel momento in cui uno o più processi deve assegnare un valore casuale ad *x*
- Questa astrazione riduce drasticamente il tempo di esecuzione dell'algoritmo di Ben-Or
- Si può dimostrare che nel caso di $n/3 \leq n < n/2$ e in presenza di un avversario "forte" (che può schedulare i processi) l'algoritmo ha probabilità di terminazione $p < 1$ (ma solo nei modelli con message passing!)

Failure Detector

- Metodo alternativo per la risoluzione del problema del consenso
- Automa del sistema in una rete asincrona che informa i processi sui fallimenti avvenuti
- Perfect FD: segnala tutti e solo i fallimenti avvenuti a tutti i processi attivi



Automa PerfectFDAgreement

Input: $init(v)_i$

$receive(w, I)_{j,i}$

$inform\text{-}stopped(j)_i$

Output: $bcast(w, I)_i$

$decide(\ v\ )_i$

Variabili:

$val \in W$, inizialmente *null*

$stopped \subseteq \{1,\dots, n\}$, inizialmente $\emptyset$

$ratified \subseteq \{1,\dots, n\}$, inizialmente $\emptyset$

decided, un Boolean, inizialmente *false*

Transizioni input

init(v)$_i$

Effetto: *val(i)* ← *v*

ratified ← {*i*}

inform-stopped(j)$_i$

Effetto: *stopped* ← *stopped* U {*j*}

ratified ← {*i*}

receive(w, I)$_{j,i}$

Effetto: **if** $j \notin$ *stopped* **then**

if (*w*, *I*) = (*val*, *stopped*) **then**

ratified ← *ratified* U {*j*}

else if (*w*, *I*) $>_d$ (*val*, *stopped*) **then**

stopped ← *stopped* U *I*

for all *k*, 1 ≤ *k* ≤ *n*, **do**

if *val(k)* = *null* **then** *val(k)* ← *w(k)*

ratified ← {*i*}

Transizioni output

$bcast(w, I)_i$

Precondizione: $w = val$

$I = stopped$

$val(i) \neq null$

Effetto: none

$decide(v)_i$

Precondizione: $ratified \; U \; stopped = \{1, \ldots, n\}$

$v = val(j)$

j = indice più piccolo con $val(j) \neq null$

$decided = false$

Effetto: $decided := true$

Proprietà dell'algoritmo

- **Well-formedness:** per costruzione
- **Validity:** evidente
- **Terminazione:** durante un'esecuzione *fair* (*init* su tutti i processi), dal momento che *val* e *stopped* possono essere modificati solo con l'aggiunta di informazione e che i processi eseguono un broadcast continuo dei propri valori, si giungerà ad un momento nel quale ognuno sarà in grado di effettuare la *decide*
- **Agreement**: In seguito alla prima *decide(v)* tutti i processi attivi condividono le stesse informazioni, affinché venga effettuata una *decide(w), $w \neq v$*, almeno un processo deve aver ricevuto dei valori differenti in input, ma questi potrebbero provenire solo da un processo *stopped* che però verrebbe ignorato