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

Impossibilità di consenso

- Non esiste un algoritmo, per il modello broadcast asincrono con un canale senza possibilità di errori, che risolva il problema del consenso e garantisca *1-failure termination*
- La dimostrazione di basa sulla trasformazione da modello broadcast asincrono a modello con memoria condivisa asincrono e l'impossibilità di risolvere il problema del consenso per quest'ultima

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'

Schema della simulazione

- Q_i : processo di B, P_i: processo di A, Q_i simula P_i
- Simulazione $bcast(m)_i$:
  - $send(m, t)_{i,j}$ $\quad \forall j \neq i$, t: *tag* (numero *bcast*)
  - $receive(m)_{i,i}$
- Q_i riceve un messaggio (m, t) da $Q_j \rightarrow send(m, t, j)_{i,n}$ $\quad \forall n \neq$ i, j
- Qi riceve un messaggio (m, t, j) da $Q_k \rightarrow send(m, t, j)_{i,n}$ $\quad \forall n \neq i, j, k$

Schema della simulazione (cont.)

- Q_i può simulare $receive(m)_{j,i}$ se:
  - Ha ricevuto un messaggio (m, t) inviato originariamente da P_j
  - Ha già reinviato (m, t, j) a tutti i processi $\neq i, j$
  - Ha già simulato $receive_{j,i}$ per tutti i messaggi da P_j con $tag < t$

Problema del consenso

Input:

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

$stop_i$: modella il fallimento

Output:

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

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

Proprietà

- **Consenso**: 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 non fallimentari
- **Wait-free**: f-failure termination con $f = n$
- **Con probabilità *p***: con probabilità p, i processi eventualmente eseguono un'azione di *decide*

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
- Basso costo computazionale
- Rilassiamo le condizioni di correttezza: la terminazione è ora probabilistica.
- I programmi non falliti eseguiranno $decide_i(v)$ ad un tempo t con probabilità $p(t)$ dove p è una funzione monotona, non decrescente e non limitata

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

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 $\forall s \in S, \nexists s' \in S \mid s' \neq s$ **then** *// tutti i valori di S sono identici*

$y \leftarrow s$

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) // solo se non ha mai deciso prima

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

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

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

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.
- L'algoritmo non garantisce che un processo esegua $decide_i$. Questo è garantito solo probabilisticamente.

Probabilità 1 ≠ Certezza

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

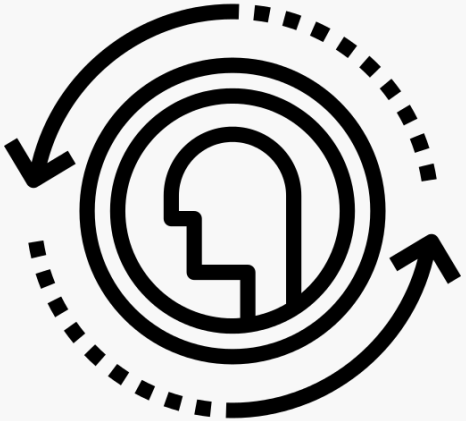
- **Esempio significativo**:

  Supponiamo di eseguire infiniti lanci di una moneta

  La probabilità che esca solo croce equivale a 0 (p = (½)^∞)

  La probabilità che esca almeno una volta testa equivale a 1

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



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

The Adversary Model

Un modello formale che impone la fairness di ogni scelta non deterministica nell'algoritmo:

- Fairness dell'automa I/O
- Fairness dell'automa di broadcast
- Limiti temporali: limite superiore *l* per i tasks, limite superiore *d* per l'invio del messaggio più vecchio
- Determina la distribuzione di probabilità durante l'esecuzione

The Adversary Model (cont.)

- Per ogni avversario ed ogni $s \geq 0$,
- tutti i processi non fallimentari decidono entro l'iterazione $s+1$ con probabilità $p = 1 - (1- ½^n)^s$
- $s = 0$: $p = 0$
- $s = 1$: $p = ½^n$ che la decisione avvenga all'iterazione $s+1 = 2$
- $s \rightarrow \infty$: $p = 1$

Caso s > 1

- Un processo non fallimentare mantiene un valore *v* “buono” se almeno *f+1* messaggi del tipo *("phase 1", s, *)* contengono lo stesso valore *v*
- Ci possono essere al più due valori buoni
- Nel caso in cui esista un solo valore “buono”, allora con probabilità $\frac{1}{2^n}$ tutti i processi sceglieranno un valore *x* identico al valore “buono”
- Allo stesso modo, nel caso esistano due valori “buoni”, con probabilità $\frac{1}{2^n}$ tutti i processi sceglieranno lo stesso valore *x*

Caso s > 1 (cont.)

Dato che le probabilità ad ogni iterazione sono indipendenti, possiamo combinarle per ottenere che:

- 1 - (probabilità che almeno un processo non sia d'accordo su ogni iterazione) = = 1 - $(1- ½^n)^s$
- Ovvero la probabilità che i processi decidano allo stage *s+1*

Riassumendo: Terminazione

- *T* è una funzione *T(s)* tale per cui un processo termina l'iterazione *s* in un tempo *T(s)* dall'evento di *init*
- $T(s) = O(s(d+l))$ (dimostrato precedentemente)
- $p(t) = 0$ se $t < T(1)$
- $p(t) = 1 - (1 - \frac{1}{2^n})^s$ per $s \geq 1$ e $T(s) \leq t < T(s+1)$

Efficienza dell'Algoritmo

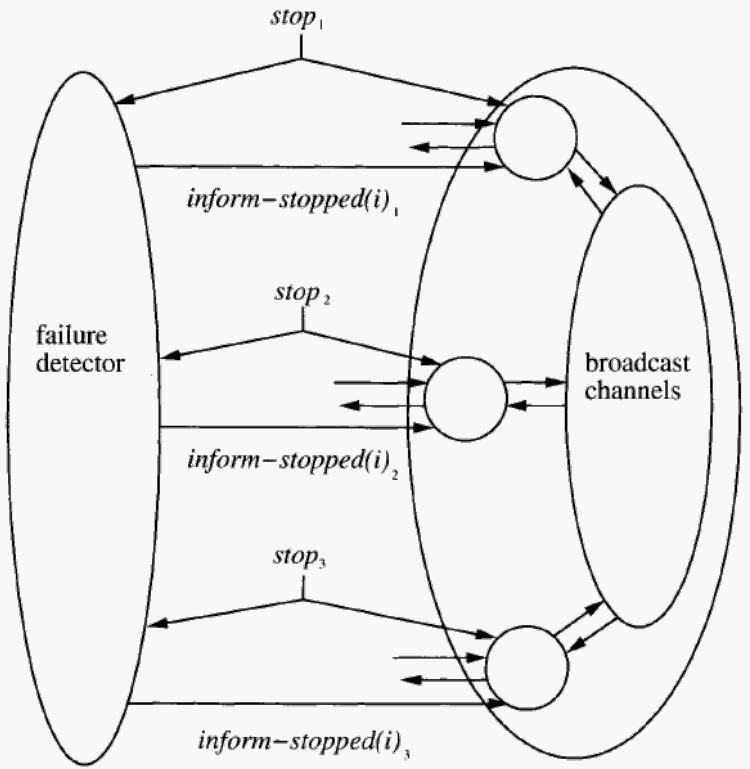
- L'algoritmo risolve anche il problema del consenso bizantino, con $n > 5f$
- Il numero di iterazioni per raggiungere il consenso può essere esponenziale
- Nel caso in cui f è $O(\sqrt{n})$ allora si può dimostrare che il numero di iterazioni è costante

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* ∪ {*j*}

ratified ← {*i*}

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

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

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

ratified ← *ratified* ∪ {*j*}

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

stopped ← *stopped* ∪ *I*

for all *k*, $1 \leq k \leq n$, **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*
- **Consenso**: 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