RANDOMIZATION

Daniele Vannella

2010/2011
Algoritmi Avanzati

INTRODUZIONE

Un algoritmo randomizzato è semplicemente un algoritmo che ha accesso ad un generatore di bit casuali. Il comportamento di tale algoritmo non è quindi determinato unicamente **dall'input** ma dipende anche *dai bit casuali che vengono generati.*

Gli algoritmi randomizzati hanno basso costo computazionale e forniscono margini di robustezza che sono in generale meno conservativi di quelli classici, ovviamente con la limitazione di un "piccolo" rischio probabilistico

INTRODUZIONE

Un algoritmo randomizzato è semplicemente un algoritmo che ha accesso ad un generatore di bit casuali. Il comportamento di tale algoritmo non è quindi determinato unicamente **dall'input** ma dipende anche *dai bit casuali che vengono generati.*

Gli algoritmi randomizzati hanno basso costo computazionale e forniscono margini di robustezza che sono in generale meno conservativi di quelli classici, ovviamente con la limitazione di un "piccolo" rischio probabilistico

INTRODUZIONE

Un algoritmo randomizzato è semplicemente un algoritmo che ha accesso ad un generatore di bit casuali. Il comportamento di tale algoritmo non è quindi determinato unicamente **dall'input** ma dipende anche *dai bit casuali che vengono generati.*

Gli algoritmi randomizzati hanno basso costo computazionale e forniscono margini di robustezza che sono in generale meno conservativi di quelli classici, ovviamente con la limitazione di un "piccolo" rischio probabilistico

INTRODUZIONE

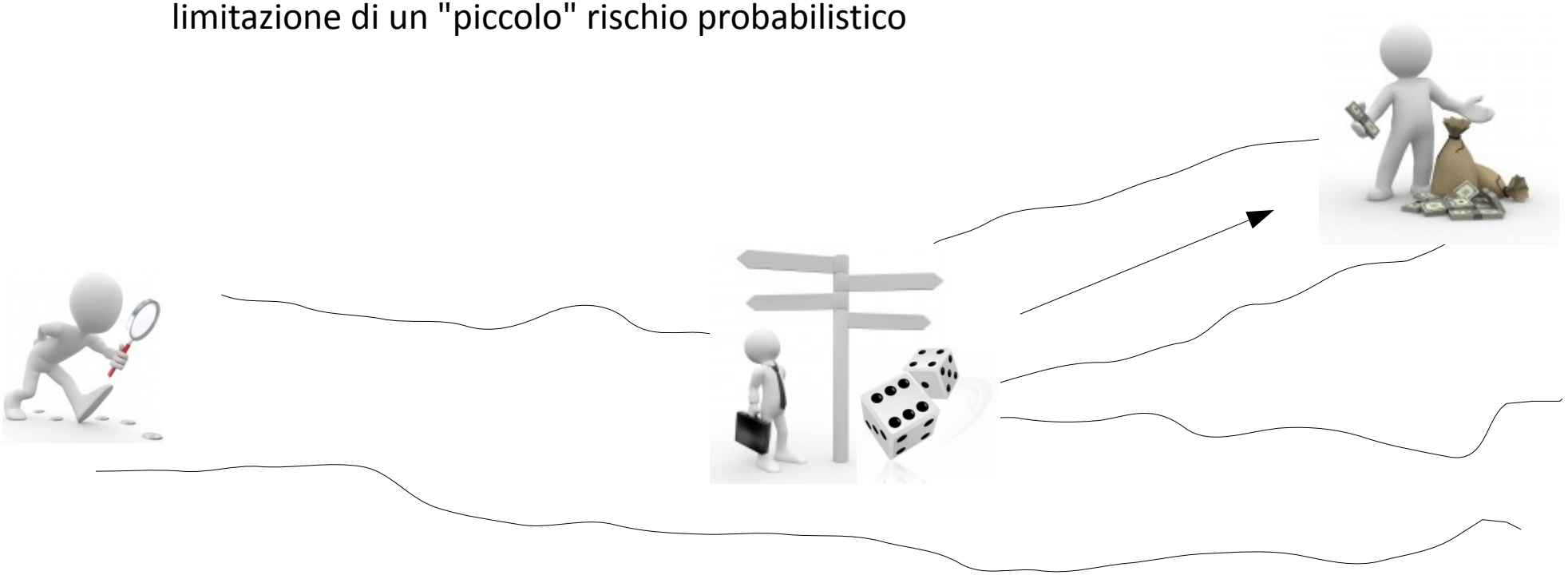
Un algoritmo randomizzato è semplicemente un algoritmo che ha accesso ad un generatore di bit casuali. Il comportamento di tale algoritmo non è quindi determinato unicamente **dall'input** ma dipende anche *dai bit casuali che vengono generati.*

Gli algoritmi randomizzati hanno basso costo computazionale e forniscono margini di robustezza che sono in generale meno conservativi di quelli classici, ovviamente con la limitazione di un "piccolo" rischio probabilistico

INTRODUZIONE

Un algoritmo randomizzato è semplicemente un algoritmo che ha accesso ad un generatore di bit casuali. Il comportamento di tale algoritmo non è quindi determinato unicamente **dall'input** ma dipende anche *dai bit casuali che vengono generati.*

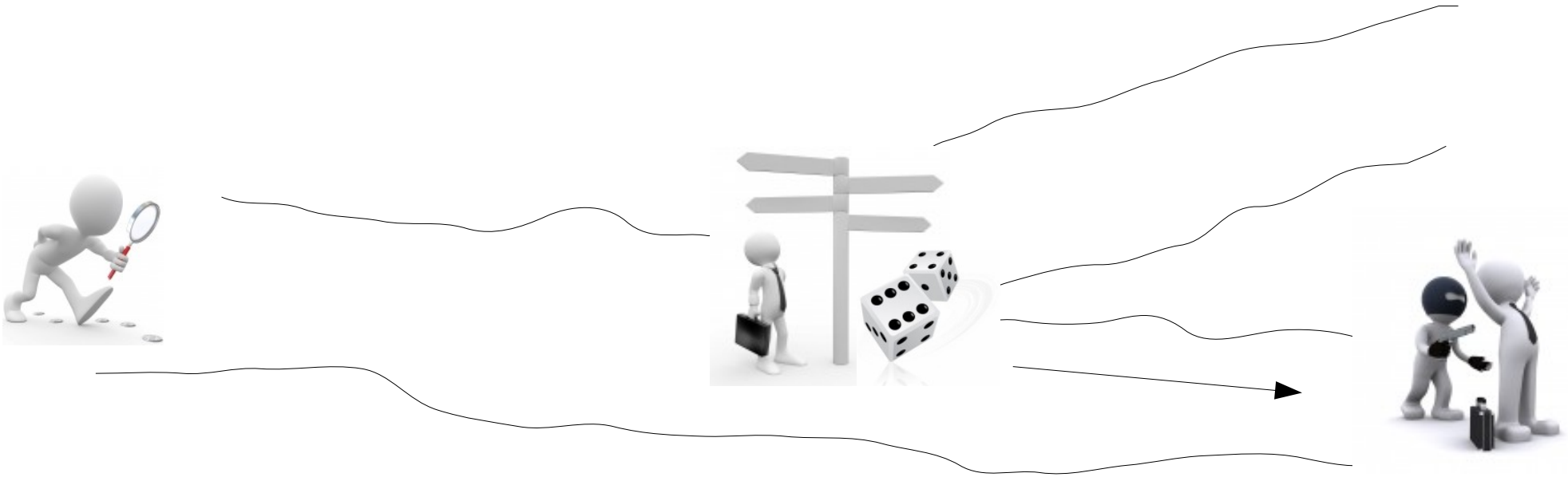
Gli algoritmi randomizzati hanno basso costo computazionale e forniscono margini di robustezza che sono in generale meno conservativi di quelli classici, ovviamente con la limitazione di un "piccolo" rischio probabilistico



INTRODUZIONE

Un algoritmo randomizzato è semplicemente un algoritmo che ha accesso ad un generatore di bit casuali. Il comportamento di tale algoritmo non è quindi determinato unicamente **dall'input** ma dipende anche *dai bit casuali che vengono generati.*

Gli algoritmi randomizzati hanno basso costo computazionale e forniscono margini di robustezza che sono in generale meno conservativi di quelli classici, ovviamente con la limitazione di un *"piccolo" rischio probabilistico*



INTRODUZIONE

Algoritmi

Elezione del leader in rete ad anello

Mutua esclusione

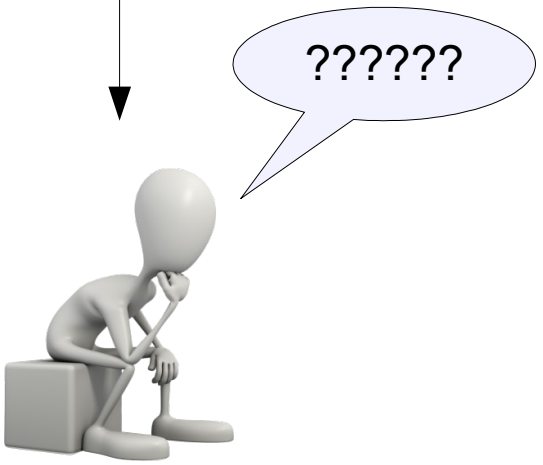
Consenso

ELEZIONE DEL LEADER

Problema

•Scelta del leader in un “anonymous ring”

•E' impossibile eleggere un leader in “anonymous ring”



ELEZIONE DEL LEADER

Obiettivo

• *Eleggere un leader in un* ***anonymous ring***

•Un algoritmo randomizzato **può** garantire che, con una ***certa probabilità verrà eletto un leader***

• Possiamo quindi risolvere una variante del problema, **rilassando** la condizione finale, cioè che un ***leader deve essere eletto in ogni esecuzione***

ELEZIONE DEL LEADER

Indebolimento

*Rilassamento delle condizioni di **Safety** e **Liveness***

Safety: in ogni configurazione di ogni ammissibile esecuzione, al più un processore è nello stato di leader **.**

Liveness: almeno un processore è eletto con una certa probabilità diversa da zero

un algoritmo che soddisfa queste condizioni può non eleggere un leader o non terminare affatto.

ELEZIONE DEL LEADER

Synchronous One-Shot Algorithm

Idea:

*Ogni processore si assegna uno **pseudo-identificatore** in maniera casuale, in un range di valori predefinito.*

Ogni processore quando riceve un messaggio contenente gli pseudo-identificatori che lo precedono, lo rinvia accodando anche il suo pseudo-identificatore.

Quando un processore ha ricevuto gli pseudo-identificatori di tutti i processori nella rete, sarà in grado di vedere se esiste un unico leader oppure no.

ELEZIONE DEL LEADER

Synchronous One-Shot Algorithm

Un pò di numeri

I valori che un processore può assumere come *pseudo-identificatore(id)* sono 1 o 2.

Pr[id =1] = $1-\frac{1}{n}$

Pr[id =2] = $\frac{1}{n}$

Gli elementi dell'inseme delle possibili configurazioni accettabili sono elementi dell'insieme

$$R=\{1,2\}^n$$

Dato un elemento ***r*** *di R,* definiamo la corrispondente configurazioni come *exec(**r**)*

ELEZIONE DEL LEADER

Synchronous One-Shot Algorithm

Algoritmo:

<inizio>

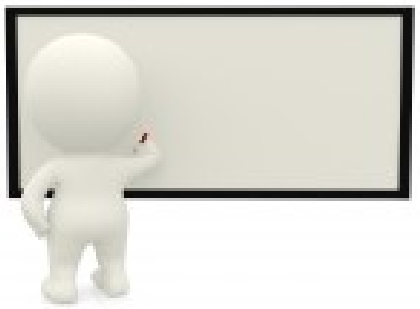
1. *Id* := { 1 con prob. 1-1/n , 2 con prob 1/n }
2. *send<id> a sinistra*

3. Quando riceve dal vicino destro un messaggio < S >
4. If |S| = *n* then
5. Se id è il solo massimo della sequenza
6. *Imposta il processore come **leader***
7. else
8. *Imposta il processore come **non leader***
9. *else send<S conc id> a sinistra*

ELEZIONE DEL LEADER

Synchronous One-Shot Algorithm

comportamento probabilistico:



Vogliamo calcolare con quale probabilità il seguente evento è verificato:

$$Pr[P] = \{ r \in R : exec(r) \; soddisfa \; P \}$$

P= l'algoritmo termina con un leader

L'evento è verificato quando tutti i processori tranne uno, hanno id = 1

Esempio:1 1 1 1 1 1 **2** 1 1 1 1 1 1 1 1 1

$$\binom{n}{1} \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} = \left(1 - \frac{1}{n}\right)^{n-1} = c$$

All'aumentare di *n*, *c* converge a e^{-1}

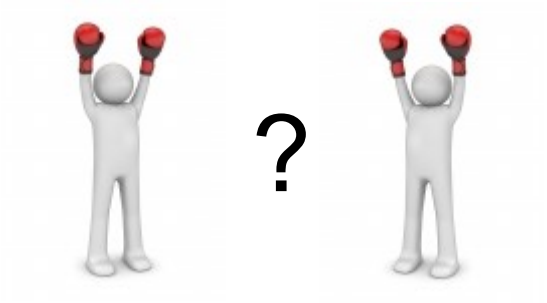
ELEZIONE DEL LEADER

Synchronous One-Shot Algorithm

comportamento probabilistico:

Abbiamo visto che l'algoritmo termina con un solo leader con probabilità 1/e.

Vediamo qual'è la probabilità che ci sia più di un leader



Dall'analisi precedente possiamo dire che la probabilità del evento descritto è minore di 1 - e^{-1}

ELEZIONE DEL LEADER

Synchronous One-Shot Algorithm

Dall'analisi fatta possiamo enunciare il seguente teorema per gli algoritmi randomizzati

Teorema:

C'è un algoritmo randomizzato che, con probabilità c > 1/e elegge un leader in una rete ad anello sincrona; l'algoritmo invia $O(n^2)$ messaggi.

MUTUA ESCLUSIONE

Problema

In un sistema distribuito la mutua esclusione è spesso necessaria per accedere a risorse condivise (sezione critica)

Si noti che in un sistema distribuito non esiste una memoria condivisa (come in un sistema centralizzato o parallelo) quindi non è possibile usare oggetti condivisi come i semafori per implementare la mutua esclusione

Assunzioni

Il sistema ha n processi; ogni processo risiede in un differente processore.
Ogni processo ha una sezione critica che richiede la mutua esclusione.

Richieste

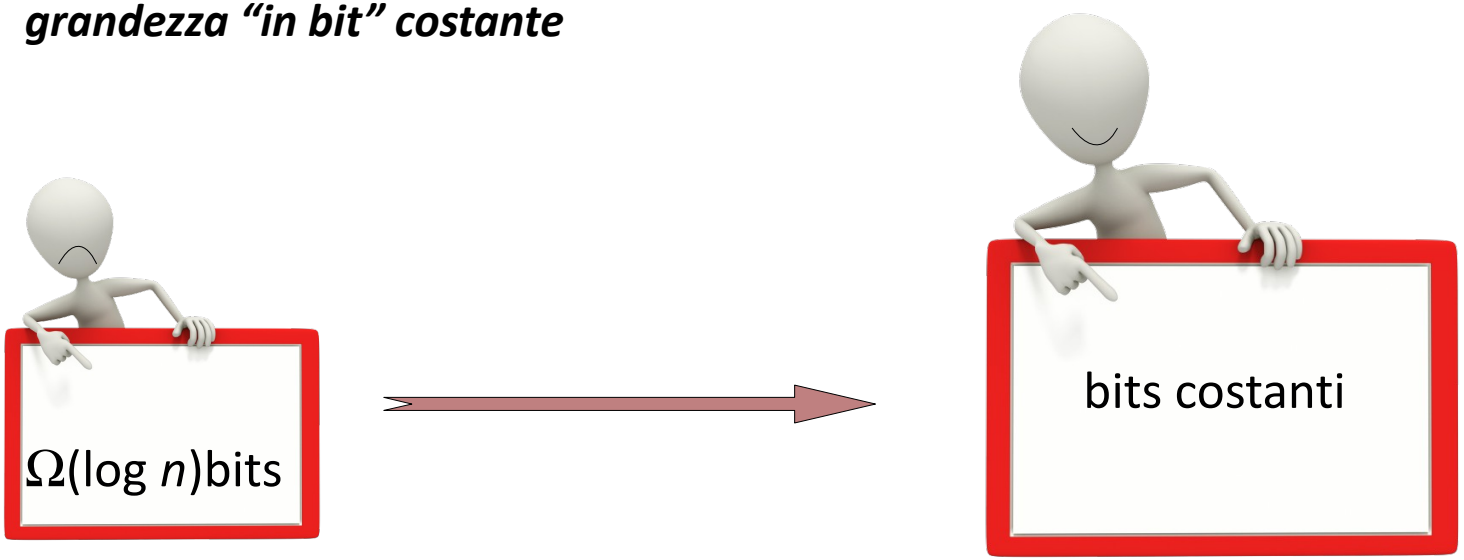
Se Pi è dentro la propria sezione critica allora nessun altro processo deve essere dentro la propria sezione critica.

MUTUA ESCLUSIONE

Obiettivo

Ogni algoritmo deterministico per risolvere il problema della mutua esclusione per *n* processori richiede una variabile condivisa di almeno **log *n*** bit

Dimostreremo che, con un algoritmo randomizzato possiamo ridurre il numero dei bit richiesti, usando solamente una ***variabile condivisa di grandezza "in bit" costante***



MUTUA ESCLUSIONE

Descrizione Algoritmo

Idea:

Ogni processore ha una probabilità di successo, cioè di entrare nella sezione critica, pari a 1/*n* dove *n* è il numero di processori che concorrono alla sezione.

L'idea è quella di dividere l'algoritmo in *due fasi* identiche, che a loro volta sono divise in 3 step

Step:

Assegnamento

Scelta del vincitore

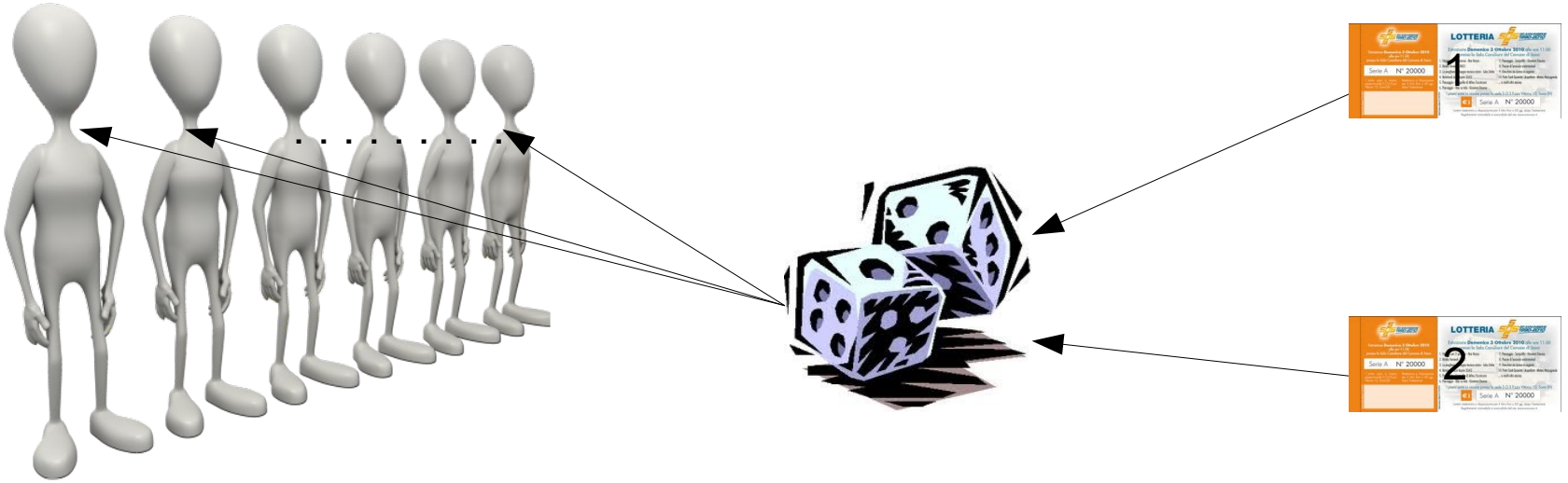
Notificazione

MUTUA ESCLUSIONE

Descrizione Algoritmo

Assegnamento:

Ad ogni processore viene assegnato un ticket.
Un ticket corrisponde a un valore, 1 o 2, come nell'elezione del leader.



MUTUA ESCLUSIONE

Descrizione Algoritmo

Scelta del vincitore:

Il processore che avrà il ticket più alto, sarà designato come il **vincitore**

N.B. se a più di un processore è stato assegnato il ticket più alto, verrà designato come vincitore il primo ad aver “ricevuto” il ticket vincente

MUTUA ESCLUSIONE

Descrizione Algoritmo

Notificazione:

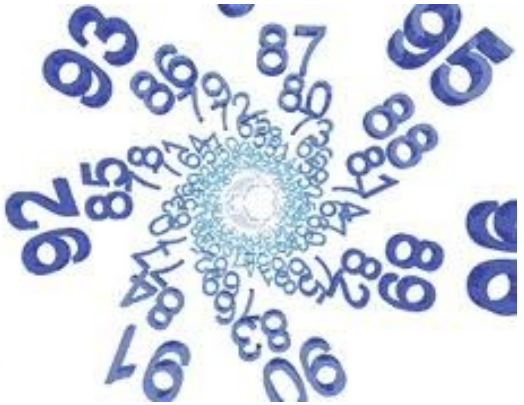
Quando la sezione critica si libera, il processore vincitore entra nella sezione critica

Tutti i processori ritornano alla fase di assegnamento

MUTUA ESCLUSIONE

Descrizione Algoritmo

Un pò di numeri



Probabilità di assegnazione dei ticket

PR[ticket=1] = $1-\frac{1}{n}$

PR[ticket=2] = $\frac{1}{n}$

Probabilità che un processore p_i entri nella sezione critica

PR[l'unico ad avere il ticket= 2] = $\frac{c}{n}$ per una costante c >0

Questo valore può essere usato come bound per il numero di tentativi di un processore

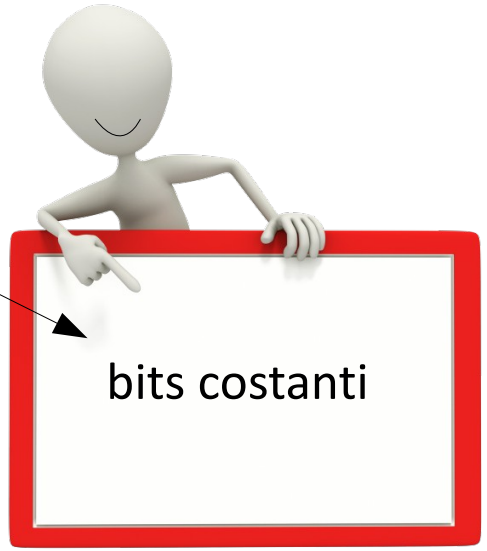
MUTUA ESCLUSIONE

Descrizione Algoritmo

Un pò di numeri

Il range dei ticket è **indipendente** come visto dal numero di processori. Infatti è costante

Il numero massimo del ticket può essere tenuto in una **variabile condivisa di grandezza costante**...Quindi



Descrizione Algoritmo

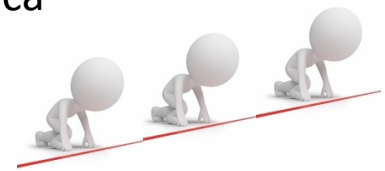
Fasi:

Ogni processore memorizza la sua ultima fase (0 o 1)

Il processore che entra nella sezione critica, decide quale sarà la nuova fase.

$$Pr[fase=0]=\frac{1}{2} \quad | \quad Pr[fase=1]=\frac{1}{2}$$

Ogni processore che si trovava in una fase diversa da quella assegnata, concorrerà alla sezione critica



$$Pr[p_i wins]=Pr[p_i wins \mid p_i partecipi]\cdot Pr[p_i partecipi]\geq \frac{c}{n}\cdot\frac{1}{2}$$

MUTUA ESCLUSIONE

Descrizione Algoritmo

Algoritmo:

<entry>
1. If last <> Fase then
2. ticket: { 1 con prob. 1-1/n , 2 con prob 1/n }
3. *last := fase*
4. if(*ticket > Max*) then
5. *Max := ticket*
6. else *ticket := 0*

7. *wait until Free = true*
8. If *ticket <> Max* then
9. *Ticket* :=0
10. GotoLine 1

11. *Free := false*
12. *Fase* := { 0 con prob. ½ , 1 con prob ½}
13. *Max := 0*
14. *Ticket := 0*
    <Critical Section >
15. *Free := true*

MUTUA ESCLUSIONE

Descrizione Algoritmo

Analisi dell'algoritmo:

<entry>

1. If last <> Fase then
2. ticket: { 1 con prob. 1-1/n , 2 con prob 1/n}

Assegnazione

3. *last := fase*
4. if(*ticket > Max*) then
5. *Max := ticket*

Scelta del vincitore

6. else *ticket := 0*

7. *wait until Free = true*
8. If *ticket <> Max* then
9. *Ticket* :=0
10. GotoLine 1

Notifica

11. *Free := false*
12. *Fase* := { 0 con prob. ½ , 1 con prob ½}
13. *Max := 0*
14. *Ticket := 0*

<*Critical Section* >

15. *Free := true*

MUTUA ESCLUSIONE

Descrizione Algoritmo

Analisi dell'algoritmo:

<entry>

1. If last <> Fase then
2. ticket: { 1 con prob. 1-1/n , 2 con prob 1/n}
3. *last := fase*
4. if(*ticket > Max*) then
5. *Max := ticket*
6. else *ticket := 0*

7. *wait until Free = true*
8. If *ticket <> Max* then
9. *Ticket* :=0
10. GotoLine 1

11. *Free := false*
12. *Fase* := { 0 con prob. ½ , 1 con prob ½}
13. *Max := 0*
14. *Ticket := 0*
    <Critical Section >
15. *Free := true*

Variabili condivise

MUTUA ESCLUSIONE

Descrizione Algoritmo

Analisi dell'algoritmo:

<entry>

1. If last <> Fase then
2. ticket: { 1 con prob. 1-1/n , 2 con prob 1/n}
3. *last := fase*
4. if(*ticket > Max*) then
5. *Max := ticket*
6. else *ticket := 0*

7. *wait until Free = true*
8. If *ticket <> Max* then
9. *Ticket* :=0
10. GotoLine 1

11. *Free := false*
12. *Fase* := { 0 con prob. ½ , 1 con prob ½}
13. *Max := 0*
14. *Ticket := 0*
    <Critical Section >
15. *Free := true*

Variabili locali

MUTUA ESCLUSIONE

Descrizione Algoritmo

Analisi dell'algoritmo:

<entry>

1. If last <> Fase then
2. ticket: { 1 con prob. 1-1/n , 2 con prob 1/n}
3. *last := fase*
4. if(*ticket > Max*) then
5. *Max := ticket*
6. else *ticket := 0*

7. *wait until Free = true*
8. If *ticket <> Max* then
9. *Ticket* :=0
10. GotoLine 1

11. *Free := false*
12. *Fase* := { 0 con prob. ½ , 1 con prob ½}
13. *Max := 0*
14. *Ticket := 0*
    <Critical Section >
15. *Free := true*

Fase: variabile di fase di un bit

Max: variabile condivisa di 2 bit contenente il massimo ticket assegnato

MUTUA ESCLUSIONE

Descrizione Algoritmo

Analisi dell'algoritmo:

```
<entry>
1.  If last <> Fase then
2.      ticket: { 1 con prob. 1-1/n , 2 con prob 1/n}
3.  last := fase
4.  if(ticket > Max ) then
5.      Max := ticket
6.  else ticket := 0

7.  wait until Free = true
8.  If ticket <> Max  then
9.      Ticket :=0
10.     GotoLine 1

11. Free := false
12. Fase := { 0 con prob. ½ , 1 con prob  ½}
13. Max := 0
14. Ticket := 0
    <Critical Section >
15. Free := true
```

Fase: variabile di fase di un bit

Max: variabile condivisa di 2 bit contenente il massimo ticket assegnato

Free: un flag che indica se un processore è nella sezione critica

ticket: l'ultimo ticket assegnato al processore

last: il bit dell'ultima fase in cui il processore ha partecipato

MUTUA ESCLUSIONE

Conclusione

L'algoritmo usa un numero costante di bits condivisi.

L'algoritmo garantisce che, solo un processore alla volta entri nella propria sezione critica.

L'attesa di un processore non è infinita.

Mutua esclusione Randomizzata

Mutua esclusione deterministica

CONSENSO

Descrizione

Il gruppo di processori devono mettersi d'accordo su un valore (es. una transazione). E' l'astrazione di una classe di problemi in cui i processori partono con le loro "opinioni" (forse divergenti) e devono **accordarsi** su un'opinione comune.

E' un problema fondamentale: qualsiasi soluzione per mutua esclusione leader election risolve il Consenso

Fischer et alii hanno dimostrato (1985) **che nessun algoritmo può garantire il raggiungimento del consenso in un sistema asincrono**, anche nel caso di un unico fallimento per crash di un processore

CONSENSO

Obiettivo

Risolvere il problema del consenso in **sistemi asincroni in cui anche un processore può guastarsi**

Risolvere il problema del consenso in un sistema asincrono con meno di *f+1* round/fasi, in presenza di *f* guasti

Assunzione

n>3f

CONSENSO

Proprietà

Safety

Accordo: tutti i processori (funzionanti) che giungono a decisione, decidono lo stesso valore

Validità: se un processore decide di cambiare il valore della sua preferenza, allora quel valore è stato proposto da qualche processore

Liveness

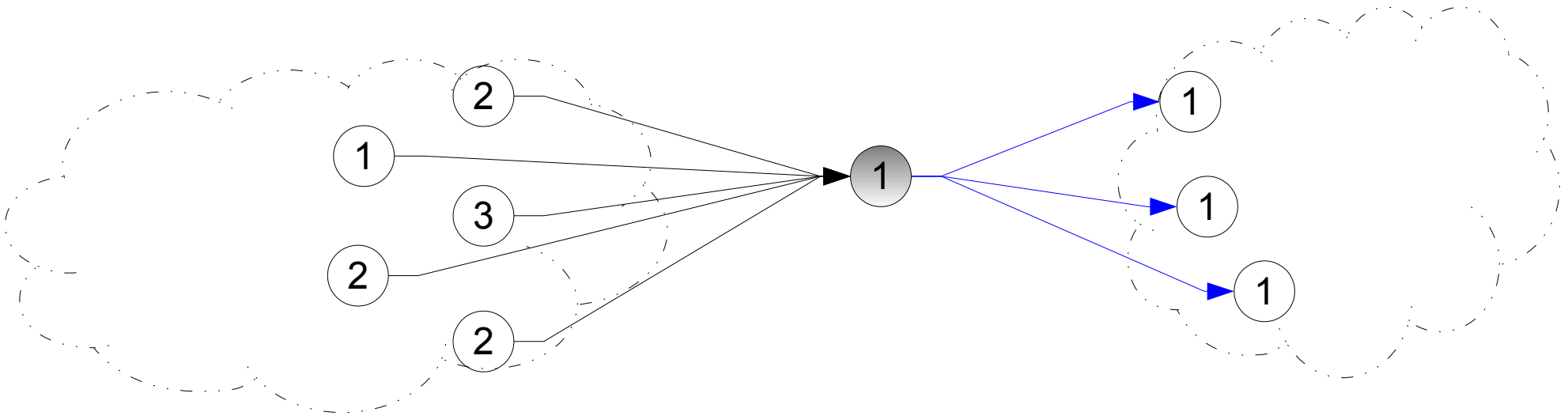
Terminazione: Tutti i processore funzionanti decidono con una certa probabilità

CONSENSO

Descrizione algoritmo 1/ 2

idea

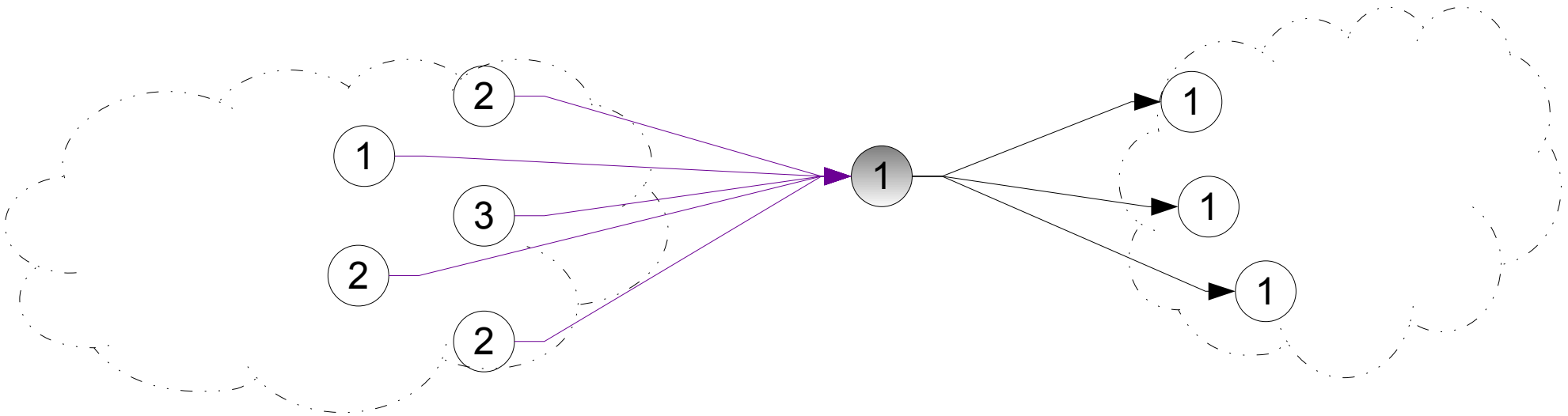
L'algoritmo è diviso in più fasi, ad ogni fase il processore deve inviare la propria preferenza,



Descrizione algoritmo 1/ 2

idea

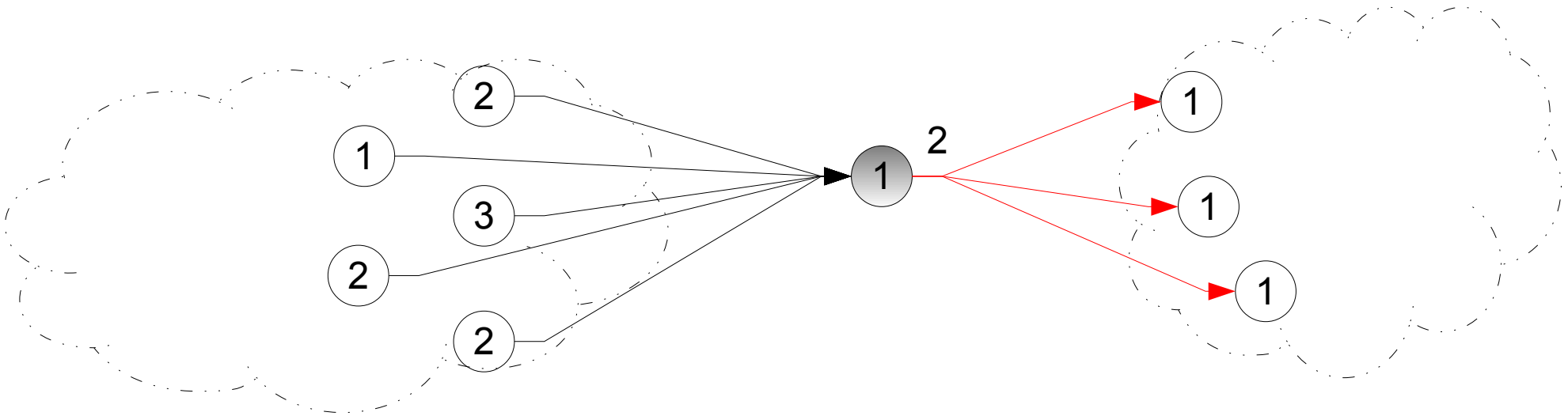
L'algoritmo è diviso in più fasi, ad ogni fase il processore deve inviare la propria preferenza, dopo di che aspetterà di ricevere la preferenza di almeno *n-f processori*



Descrizione algoritmo 1/ 2

idea

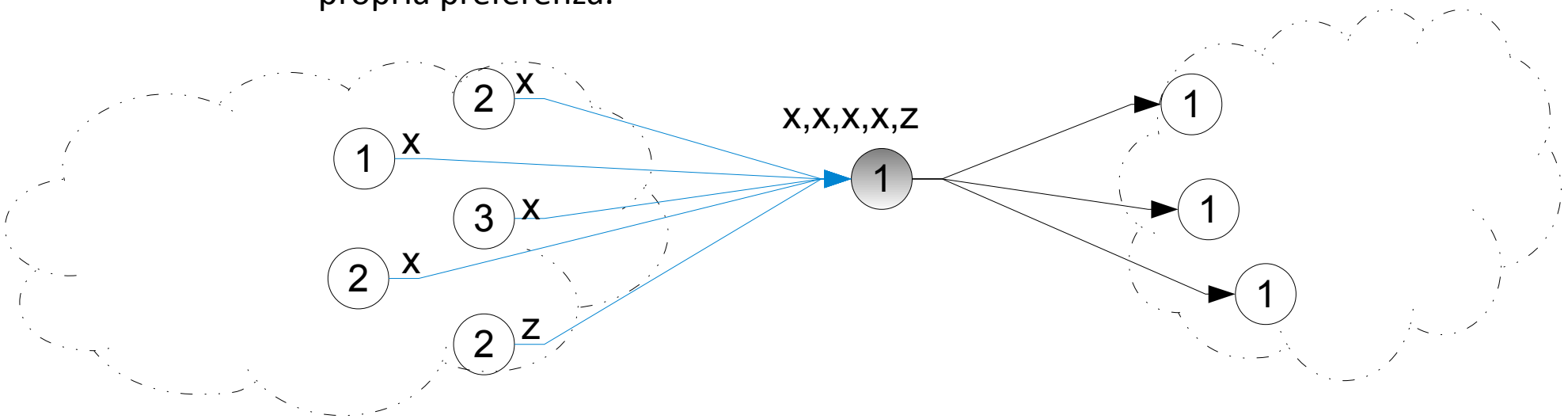
L'algoritmo è diviso in più fasi, ad ogni fase il processore deve inviare la propria preferenza, dopo di che aspetterà di ricevere la preferenza di almeno *n-f processori.* Il processore memorizzerà la preferenza che gli è arrivata dalla maggioranza dei processori (in caso non ci sia sceglierà un valore di default)e la invierà.



Descrizione algoritmo 1/ 2

idea

L'algoritmo è diviso in più fasi, ad ogni fase il processore deve inviare la propria preferenza, dopo di che aspetterà di ricevere la preferenza di almeno *n-f processori.* Il processore memorizzerà la preferenza che gli è arrivata dalla maggioranza dei processori (in caso non ci sia sceglierà un valore di default)e la invierà. Aspetterà di ricevere almeno *n-f* maggiori preferenze prima di decidere se cambiare la propria preferenza.



CONSENSO

Descrizione algoritmo 2 / 2

L'algoritmo del consenso randomizzato si divide in due principali funzioni.

deterministica:

- Decisione del valore preferito
- Invio della preferenza
- Ricezione delle preferenze di *n-f* processori
- Invio della preferenza più gettonata
- Ricezione della preferenza più gettonata
- Cambiare o no la propria preferenza

Nondeterministica:

- “lancio della moneta”

CONSENSO

Descrizione algoritmo

Algoritmo: f guasti tollerati

Inizializza *r=1* e *prefer = x*

1. While true do
2. bc-send((<*voto*,*prefer*,*r*>),affidabile)
3. Attendi il messaggio *voto* di almeno *n-f* processori
4. *V := valore più gettonato* nella fase *r*
   //valore di default se non esiste
5. *se* in tutta la fase *r,* ricevi *v { y:=v } "valore finale v"*
6. bc-send((<*risultato*,*v*,*r*>),affidabile)
7. Attendi il messaggio *risultato* di almeno *n-f* processori
8. *se* in tutta la fase *r* ricevo come messaggio *risultato w{ prefer:=w}*
9. else *prefer :=* **lancio-moneta()**
10. r *:= r+1*

CONSENSO

Analisi Algoritmo

Dalla riga 5(*se* in tutta la fase *r,* ricevi *v { y:=v } "valore finale v"*) si implica:

Lemma 1

Per ogni r > 0, se tutti i processori alla fase r preferiranno v, allora tutti i processori non guasti decidono v non andando oltre alla fase r

Lemma 2

Per ogni r > 0, se alcuni processori durante la fase r decidono v, allora tutti i processori non guasti decideranno v nella fase r o deterministicamente preferiranno v nella fase r+1

n-f processori preferiscono v (*perchè se almeno un processore decide v vuol dire che gli sono arrivati f-n v)*
v, v, v, v,....v, v

f processori preferiscono z *Siccome n> 3f => n-2f votano v => il processore preferirà v*
z, z... z

CONSENSO

Analisi Algoritmo

<u>Lemma 3</u>

Per ogni r > 0, se alcuni processori deterministicamente preferiranno v, nella fase r+1, allora nessun processore avrà deciso z nella fase r o deterministicamente preferirà z nella fase r+1

<u>Lemma 4</u>

Se alcuni processori decidessero v, allora tutti i processori non guasti eventualmente decideranno v

<u>Lemma 5</u>

*Per ogni r>0 , la probabilità che tutti i processori non guasti decidano nella fase r è almeno **p***

CONSENSO

Analisi Algoritmo

Lemma 5

Per ogni r>0 , la probabilità che tutti i processori non guasti decidano nella fase r è almeno ***p***

dim

Caso 1:

La probabilità che due processori nella fase r restituiscano lo stesso valore con il lancio della moneta è almeno 2p- con probabilità p di scegliere 0 o 1.Dal lemma 1 è verificato il lemma 5

Caso 2:

Alcuni processore scelgono deterministicamente come valore preferito V nella fase r. Nessun altro deterministicamente preferirà z, dal lemma 3. Così ogni processore che raggiunge la fase r, o ha deciso v nella fase r-1, o deterministicamente preferirà v, oppure lancerà la moneta. Quindi ci riduciamo al caso precedente.

CONSENSO

Analisi Algoritmo

Teorema 1

Se c'è un procedura casuale di lancio della moneta, con probabilità ***p*** *con complessità T, la complessità del tempo di esecuzione dell'algoritmo visto è* $O(p^{-1}T)$

dim

La probabilità che termini nella prima fase, dal lemma 5 è almeno ***p****, quindi la probabilità che termini dopo i fasi è* $(1-p)^{i-1}\ p \Rightarrow p^{-1}$

Geometric random variable

Analisi Algoritmo

Teorema 1

*Se c'è un procedura casuale di lancio della moneta, con probabilità **p** con complessità T, la complessità del tempo di esecuzione dell'algoritmo visto è $O(p^{-1}T)$*

dim

*La probabilità che termini nella prima fase, dal lemma 5 è almeno **p**, quindi la probabilità che termini dopo i fasi è $(1-p)^{i-1}\ p \Rightarrow p^{-1}$*

Geometric random variable

Chiaramente il tempo di esecuzione è determinato dalla complessità della procedura di lancio della moneta

FINE

Domande