RELAZIONE ALGORITMO MCBYZANTINEGENERAL

Lavoro svolto da:
-Magno Alessandro : 4478234

Descrizione

L'algoritmo si basa su un'ipotesi, ovvero che ad ogni round i processi sono informati sul risultato del lancio di una moneta, che può essere testa o croce con probabilità 1/2. Quindi tutti i processi possono accedere a questo risultato e comportarsi di conseguenza. In input si ha ogni processo, si è interessati solo a quelli affidabili (l'algoritmo corre sugli affidabili più che su tutti) e tutti i processi sono inizializzati con un valore iniziale. In output con probabilità almeno 1/2 si ha il processo i-esimo uguale a v, bit sul quale si raggiungesse il controllo. Il protocollo consiste nel ripetere i seguenti passi:

1. trasmettere il proprio bit agli altri n-1 processi
2. ricevuti i valori spediti dagli altri n-1 processi
3. calcolare valore maggioritario tra i ricevuti incluso il processo i-esimo
4. calcolare numerosità valore maggioritario (chiamato tally)
5. se questo tally fosse maggiore o uguale di 2t + 1
6. allora processo i-esimo darebbe uguale al valore maggioritario
7. altrimenti se l'esito del lancio della moneta è testa
8. processo i-esimo viene posto uguale a 1
9. se croce uguale a 0

Implementazione

```
1.  import numpy as np
2.  import random as rd
3.
4.  def update_mat(proc,n,a):
5.      mat = np.empty((n, 0)).tolist()
6.      for i in range(0,a):
7.          mat[i].append(proc[i])
8.      for i in range(a, n):
9.          mat[i].append(rd.randint(0,1))
10.     return mat
11.
12. def send_random_value(mat,a,n):
13.   value = rd.randint(0,1)
14.   for j in range(a, n):
15.     for i in range(0, n):
16.         if(j != i):
17.             mat[i].append(value)
18.
19. def send_value(mat,a,n):
20.   for j in range(0,a):
21.     for i in range(0, n):
22.         if(j != i):
23.             mat[i].append(mat[j][0])
24.   send_random_value(mat,a,n)
25.
26. def calculate_maj(r,a):
27.     maj = np.empty((a, 0)).tolist()
28.     for i in range(0,a):
29.        if r[i].count(0)>r[i].count(1):
30.             maj[i].append(0)
31.             maj[i].append(r[i].count(0))
32.        else:
33.             maj[i].append(1)
34.             maj[i].append(r[i].count(1))
```

```
    return maj

def calculate_tally(maj,a):
    tally = []
    for i in range(0,a):
        tally.append(maj[i][1])
    return tally

def check_tally(tally, t, maj, moneta,a):
    proc = []
    for i in range(0,a):
        if tally[i] >= 2*t+1:
            proc.append(maj[i][0])
        elif moneta == "testa":
            proc.append(1)
        else:
            proc.append(0)
    return proc

def check_process(proc,a):
    first = proc[0]
    for i in range(0,a):
        if(first != proc[i]):
            return 0
    return 1

#implementazione algo
def algorithm(mat,a,n,t):
    round = 1
    while(1):
        #lancio della moneta
        moneta = rd.choice(["testa","croce"])
        #trasmetti b(i) agli altri n-1 processi e spedizione valori degli n-1 processi
        ##i valori vengono salvati come: [[val1,ric1,ric2,ric3..], [val2,ric1,ric2..]..]
        send_value(mat,a,n)
        #ricevi i valori spediti dagli altri n-1 processi + b(i) incluso
        r = mat.copy()
        #valutazione maggioritario
        maj = calculate_maj(r,a)
        ##al maj ho aggiunto anche il tally in modo da estrarlo dopo
        ##es maj = [[num_maj, tally]]
        #numerosità di maj
        tally = calculate_tally(maj,a)
        #check tally >= 2t + 1 e ritorna nuova matrice con solo i processi
        proc = check_tally(tally, t, maj, moneta, a)
        #controllo se tutti i processi hanno lo stesso valore, in tal caso
        ##hanno raggiunto il consenso
        if(check_process(proc,a)):
            print("Consenso raggiunto in round: ", round)
            print(proc
            break
        #aggiorno la matrice con i nuovi valori dei processi
        mat = update_mat(proc,n,a)
        round = round + 1

def fill_matrix(mat,a,n,v0):
    for i in range(0,a):
        mat[i].append(v0)

    for i in range(a,n):
        mat[i].append(0)

def fill_matrix_2(mat,a,n):
    for i in range(0,n):
        mat[i].append(rd.randint(0,1))
```

```
100.        def main():
101.            #processi totali
102.            n = 22
103.            #traditori
104.            t = 1
105.            #affidabili
106.            a = n-t
107.            #v0 inziale (esercizio 1)
108.            #v0=1
109.            #matrice generale
110.            mat = np.empty((n, 0)).tolist()
111.
112.            #per esercizio 1
113.            #fill_matrix(mat,a,n,v0)
114.            #per esercizio 2
115.            fill_matrix_2(mat,a,n)
116.            """
117.            valori dei processi nella matrice sono
118.            inseriti come [[val1],[val2]..]
119.            """
120.            for i in range(0,100):
121.                algorithm(mat,a,n,t)
122.
123.        main()
```

1. Inizializzando tutti i processi affidabili con lo stesso valore iniziale es.1, ogni processo finale converge a 1 nel primo round
2. $\bar{x}(valor\ medio) = \frac{x_1+\ldots+x_n}{n} = \frac{1+\ldots+1}{661} = 1 \qquad \sigma^2(varianza) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n} = \frac{\sum_{i=1}^{661}(1-1)^2}{661} = 0$
   La varianza è uguale a 0 perché tutti i valori dei round sono uguali e quindi non c'è variabilità nella distribuzione.
3. Sapendo che la probabilità di errore è di $(1/2)^{round}$, maggiore è il numero dei round, minore è la probabilità di errore e quindi è più probabile che si raggiunga il consenso. Mi sceglierei il numero di round in base alla probabilità di errore che voglio accettare ad esempio voglio essere sicuro che dopo quel numero di round sia raggiunto il consenso, scelgo 6 round, $(1/2)^6$ probabilità di errore dell'1%.