Implementazione Accordo Bizantino – Cattaneo Kevin – S4944382
Protocollo Monte Carlo

Per l’implementazione dell’accordo bizantino, protocollo Monte Carlo, si è scelto di utilizzare il linguaggio C++ e il linguaggio Python. In particolare, l’utilizzo del secondo è ristretto alla grafica della frazione richiesta.

Di seguito il codice C++

```
#include <iostream>
#include <random>
#include <time.h>
#include <fstream>
#include <vector>

const int t = 1; // numero processi inaffidabili
const int numProc = 3*t+1; // numero processi totali, gli ennesimi sono inaffidabil
i
const int numRun = pow(2,10);

int MCByzantine(int bit[], int& round){
  int a[numProc][numProc-t];

  while (true){
    // Trasmetto bit (fra righe), tranne quello inaffidabile
    for(int k=0; k<numProc-t; k++)
      for(int i=0; i<numProc-t; i++)
          a[k][i] = bit[k];

    // Trasmissione della confusione da parte degli inaffidabili
    for(int k=0; k<numProc-t; k++){
      for(int i=numProc-t; i<numProc; i++)
        a[i][k] = 1-bit[k];
    }

    int numZero[numProc-t];
    int numOne[numProc-t];
    int maj[numProc-t];
    int tally[numProc-t];
    int moneta = rand()%2; // moneta globale: esito uguale x tutti ad ogni round

    // Conta dei valori maggioritari (solo per processi affidabili)
    for(int k=0; k<numProc-t; k++){
      numZero[k] = 0;
      numOne[k] = 0;

      for(int i=0; i<numProc; i++){
        if(i != k){
          if(a[i][k] == 0)
            numZero[k]++;
          else numOne[k]++;
        }
      }
```

```
      // Includo il proprio bit trasmesso
      if(bit[k] == 0)
        numZero[k]++;
      else numOne[k]++;

      // Calcolo maggioritari
      if(numZero[k] > numOne[k]){
        maj[k] = 0;
        tally[k] = numZero[k];
      }
      else{
         maj[k] = 1;
         tally[k] = numOne[k];
      }
      //std::cout << "Bit: " << bit[k] << std::endl;
      if(tally[k] >= 2*t+1)
        bit[k] = maj[k];
      else if(moneta==0)
        bit[k] = 1;
      else bit[k] = 0;
    }

    // Verifico se si è raggiunto il consenso fra gli affidabili
    bool consenso = true;
    for(int i=0; i<numProc-t-1; i++)
      if(a[i][0] != a[i+1][0])
        consenso = false;
    if(consenso) return a[0][0];
    round++;
  }
}

int main(){
  srand(time(NULL));
  int bit[numProc-t], round; // bit trasmessi da processi affidabili
  std::vector<int>roundTot;
  std::vector<int>freq(numRun+1,0);

  for(int i=0; i<numRun; i++){
    bit[0] = bit[1] = 0;
    bit[2] = 1;

    /***
    NOTA: Di per sè non saranno mai d'accordo al 'primo' round
    per come ho impostato i bit, quindi non considero l'inizializzazione
    dei bit come un round (che dunque parte da zero).
    ***/

    MCByzantine(bit, round=0);
    roundTot.push_back(round);
  }

  // Freq[0] = numero di run in cui l'accordo è raggiunto in 1 round
  for(int i=0; i<numRun; i++)
```

```
    for (int e : roundTot)
      if(e == i+1)
        freq[i]++;

  // Stampa su file della frequenza di ottenimento dei round
  std::cout << "Round | Frequenza consensi\n";
  std::ofstream f("freq.txt");
  if (f.is_open())
  {
    for(int i=0; i<numRun; i++)
      if(freq[i] != 0){
        f << freq[i] << "\n";
        std::cout << i+1 << ": " << freq[i] << std::endl;
        if(accumulate(freq.begin(),freq.begin()+i,0) == numRun) break;
      }
    f.close();
  }
  else std::cout << "Errore nell'apertura del file";
}
```

Screenshot dell'output ottenuto

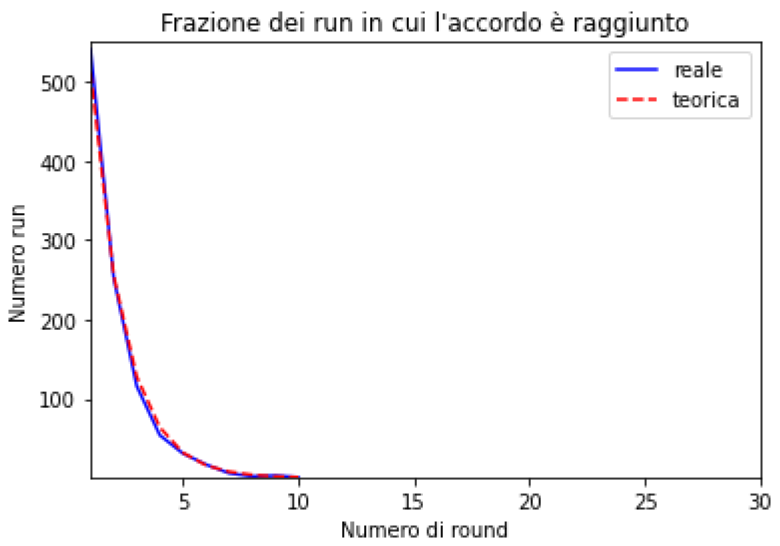
```
Round | Frequenza consensi
1: 541
2: 253
3: 116
4: 54
5: 31
6: 17
7: 6
8: 2
9: 3
10: 1
```

Di seguito il codice Python

```
## Cattaneo Kevin - S4944382 - APA - Informatica UniGE ##
# Plotting del grafico di MCByzantine
import matplotlib.pyplot as plt # libreria per i grafici
import numpy as np

# Elaboro il file in input
freq = "/content/freq.txt"
myfile = open(freq, "r")
FileContent = [int(k) for k in myfile.readlines()]

# Creazione grafico
x = np.linspace(1,len(FileContent), len(FileContent))
y = pow(2,10)/pow(2,x) # teorica

plt.plot(x, FileContent, 'b', label="reale")
plt.plot(x, y, 'r--', label="teorica")
plt.legend(loc="upper right")
plt.title("Frazione dei run in cui l'accordo è raggiunto")
```

```
plt.xlabel("Numero di round")
plt.ylabel("Numero run")
plt.xlim([1, 30])
plt.ylim([1, 550])

myfile.close()
plt.show()
```

Screenshot del grafico ottenuto



Commento

A valle di quanto si osserva sul grafico, si può confermare la tesi per cui la frazione dei run R in cui l'accordo è raggiunto in r round è all'incirca pari a R/2^r. Il motivo per cui ciò è vero è da ricondursi a un fattore probabilistico: l'algoritmo Monte Carlo, nel caso in cui i processi affidabili non siano tutti d'accordo, determina il consenso dal lancio di una moneta globale per ogni round, dunque con probabilità pari a 1/2 dopo l'esito del lancio della moneta, tutti i processi convergeranno sul valore maggioritario.

Da quanto affermato si denota quindi che date R run, R/2 avranno trovato consenso al primo round, e le altre R'=R/2 lo avranno trovato con un numero maggiore a uno; di queste ultime R'/2 (cioè R/4) troveranno il consenso al secondo round, mentre le restanti R'/2 lo troveranno con più di due round, e così via. Si osserva che ad ogni round il denominatore cresce di un fattore 2 in accordo alla formula R/2^r.