

- Jollar – Laboratorio di Sistemi Operativi A.A. 2017/2018
- MaiUnaJollar
- Indirizzo mail di riferimento: alberto.donati6@studio.unibo.it
- Componenti:
 - Alberto Donati 788850
 - Nicolò Paganelli 804801
 - Matteo Tramontano 772520
 - Slavcho Vlakeski 767280
 - Giacomo Pazzaglia 691578

DESCRIZIONE GENERALE DEL PROGETTO

Il progetto è basato principalmente su 4 nodi (A, B, C e D)

i quali sono collegati tra loro in rete Peer2Peer, in cui nessun nodo è server o client ma sono entità equivalenti tra loro.

Ogni nodo cerca di effettuare del lavoro per provare di poter scrivere il blocco (“minatura” o più correttamente “mining”).

Questo lavoro è definito tramite la Proof-of-Work, un algoritmo necessario per verificare che sia stata eseguita una notevole quantità di lavoro.

Caratteristica fondamentale di questo algoritmo è costringere i nodi che vogliono “minare” a cercare un valore che sia difficile da trovare e facile da verificare (deve essere immediato verificarne la correttezza, compito che spetta agli altri nodi dal momento che si tratta di una rete peer2peer).

La Proof-of-Work in questione consiste nel generare catene di numeri primi con determinate proprietà (è quella utilizzata dalla moneta Primecoin), e solo una catena con queste determinate proprietà potrà essere validata.

Il nodo ha tre possibilità, ossia tre catene possibili, di cui una sola potrà essere valutata.

Le tre catene:

1) *Cunningham del Primo Tipo:*

una catena di Cunningham del primo tipo di lunghezza n è una sequenza di primi (p_1, \dots, p_n) in cui per tutti gli $1 \leq i < n$, è valida la relazione $p_{i+1} = 2p_i + 1$.

Sotto viene riportato il codice Jolie per la creazione di questa catena, trattandosi di una semplice formula matematica in cui ogni numero generato è uguale al doppio del precedente aumentato di un’unità, questa catena è stata realizzata tramite un semplice ciclo for.

La prima riga “chain.creator = string(“A”);” serve semplicemente a tenere traccia di quale nodo sia il creatore della catena in questione (in questo caso il Nodo A).

Come numero di partenza è stato scelto un numero random compreso tra 1 e 10 in modo da rendere imparziale la scelta.

Ogni nodo partirà da quel numero generato casualmente e tenterà di generare la Proof-of-Work, prima con Cunningham del primo tipo, poi con Cunningham del secondo tipo e in seguito con BitWin (catene descritte in

seguito), solo nel caso di mancata verifica di tutte e tre le catene si genererà un nuovo numero random e si ripeterà l'iter.

```
define cunninghamPrimo{
    chain.creator = string("A");

    println@Console("la lunghezza della catena e' " + lunghezzaCatena());

    chain.type = "CunninghamPrimo";

    println@Console("il tipo della catena e' " + chain.type());

    println@Console("Numero di partenza " + numRand());

    for ( i = 0, i < lunghezzaCatena, i++ ) {
        if(i == 0){
            chain.CunninghamPrimo.numCunP[i] = numRand;
            println@Console(chain.CunninghamPrimo.numCunP[i])()
        }
        else{
            chain.CunninghamPrimo.numCunP[i] = ((2*chain.CunninghamPrimo.numCunP[i-1]) + 1)
            println@Console(chain.CunninghamPrimo.numCunP[i])()
        }
    }
}
```

2)Cunningham del Secondo Tipo:

una catena di Cunningham del secondo tipo di lunghezza è una sequenza di primi (p_1, \dots, p_n) in cui per tutti gli $1 \leq i < n$, è valida la relazione $p_{i+1} = 2p_i - 1$.

(In questo caso il codice è analogo alla catena di Cunningham del primo tipo)

3)-Catene Bi-twin:

una catena Bi-twin di lunghezza $k+1$, è un array di numeri

primi:

$n-1, n+1, 2n-$

$1, 2n+1, \dots, 2^k(n-$

$1), 2^k(n+1)$

(possono essere viste tuttavia come l'unione di cunningham e cunningham2)

```
define biTwin{
    chain.creator = string("A");
    lunghezzaCatena++;
    println@Console("la lunghezza della catena e' " + lunghezzaCatena());

    chain.type = "BiTwin";

    println@Console("il tipo della catena e' " + chain.type());

    println@Console("Numero di partenza " + numRand());
    indicePow = 0;
    for ( i = 1, i < lunghezzaCatena, i = i+2 ) {
        PowRequest.base = 2;
        PowRequest.exponent = indicePow;
        pow@Math(PowRequest)(resultPostPow);
        resultPostPow = int(resultPostPow);
        chain.BiTwin.numBiTw[i-1] = resultPostPow*numRand - 1;
        chain.BiTwin.numBiTw[i] = resultPostPow*numRand + 1;

        println@Console(chain.BiTwin.numBiTw[i-1])();
        println@Console(chain.BiTwin.numBiTw[i])();
        indicePow++;
    }
}
```

La verifica di queste catene avviene tramite il **Piccolo teorema di Fermat** il quale afferma:

Dato un numero primo “p” e un intero “a”:

$$a^{(p-1)} \equiv 1 \pmod{p}$$

I numeri che soddisfano questa proprietà sono gli “pseudoprimi” (cioè non obbligatoriamente primi), tuttavia maggiore è “p” maggiore è la probabilità che esso sia primo.

Quindi la primalità viene controllata per ogni numero della catena.

```
define VerificaFermat{
println@Console("Verifica di Fermat")();
fermat = 0;
PowRequest.base = 2;

PowRequest.exponent--;
pow@Math(PowRequest)(resultPow);
PowRequest.exponent++;
condizione = true;
while(condizione){
    fermat = (resultPow%PowRequest.exponent);

    //da commentare dopo aver eseguito dei test sulla potenza
    println@Console(" Il remainder di Fermat e': " + fermat());

    if(fermat == 1 || PowRequest.exponent == 2){
        println@Console("*****primo*****")();
        condizione = false
    }else {
        println@Console("*****fermat no primo*****")();

        condizione = false;

        chainPrime = false
    }
};
println@Console("FINE Verifica di Fermat")()
```

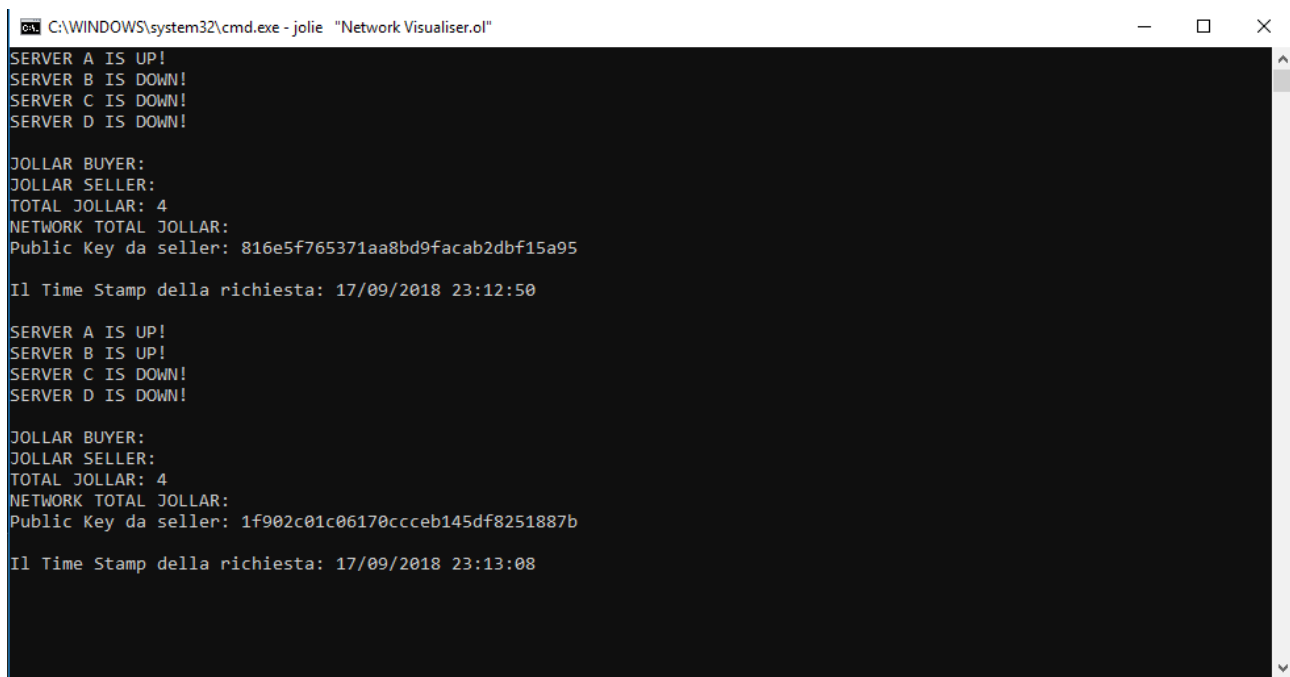
Anche in questo caso nel codice è stato semplicemente implementata la formula come sopra.

Da notare che l’operazione è gestita tramite un booleano settato inizialmente a “true” che viene poi settato a false se il numero in questione è verificato come primo, ciò fa sì che la primalità venga controllata per ogni numero della catena.

Il nodo che sta lavorando dopo aver creato la catena (prima della verifica) controlla lo stato degli altri nodi. Se questi sono attivi, esso invierà la catena generata che sarà poi sottoposta a verifica da tutti i nodi attivi, oltre che da se stesso. Nel caso il nodo sia l’unico nodo attivo, esso validerà la sua catena autonomamente.

Nel caso in cui la catena sia verificata, al nodo creatore della catena verranno assegnati 6 jollar di reward. Da questo momento è possibile effettuare una transazione (una sola) tra i nodi.

Fatto ciò il nodo che ha inviato jollar può decidere di utilizzare il Network Visualizer per visualizzare le informazioni relative ai nodi.



```
C:\WINDOWS\system32\cmd.exe - jolie "Network Visualiser.ol"
SERVER A IS UP!
SERVER B IS DOWN!
SERVER C IS DOWN!
SERVER D IS DOWN!

JOLLAR BUYER:
JOLLAR SELLER:
TOTAL JOLLAR: 4
NETWORK TOTAL JOLLAR:
Public Key da seller: 816e5f765371aa8bd9facab2dbf15a95

Il Time Stamp della richiesta: 17/09/2018 23:12:50

SERVER A IS UP!
SERVER B IS UP!
SERVER C IS DOWN!
SERVER D IS DOWN!

JOLLAR BUYER:
JOLLAR SELLER:
TOTAL JOLLAR: 4
NETWORK TOTAL JOLLAR:
Public Key da seller: 1f902c01c06170ccceb145df8251887b

Il Time Stamp della richiesta: 17/09/2018 23:13:08
```

Conclusa questa transazione i nodi tornano in “waiting”.

ISTRUZIONI PER LA DEMO

- 1) Avvio del server TimeStamp da terminale tramite comando -jolie Server.ol-.
- 2) Avvio de Network Visualizer da terminale tramite comando -jolie Network Visualizer.ol-.
- 3) Avvio del primo nodo da terminale tramite comando -jolie NodeA.ol-
- 4) Avvio del secondo, terzo e quarto nodo da terminale tramite comandi -jolie NodeB.ol-, -jolie NodeC.ol- e -jolie NodeD.ol-
- 5) Invio di un jollar dal primo al secondo nodo con inserimento da terminale (digita “x” per inviare jollar a “y”) e successivamente selezionare la quantità di jollar da inviare (1);
- 6) Invio di due jollar dal primo al terzo nodo con procedura analoga al punto 5.
- 7) Invio di tre jollar dal primo al quarto nodo con procedura analoga al punto 5 e 6.

8) Rispondere “si” alla richiesta del Network Visualizer riguardante la stampa delle informazioni.

STRUTTURA DEL PROGETTO

La divisione del progetto:

Server TimeStamp:

Tutto il gruppo

Interfacce:

Tutto il gruppo

Creazione nodi:

Alberto Donati

Creazione delle catene:

Nicolò Paganelli

Matteo Tramontano

Network Visualizer:

Slavcho Vlakeski

Giacomo Pazzaglia

La divisione dei compiti è avvenuta in maniera spontanea dinamicamente durante lo svolgimento del progetto.

Il gruppo ha lavorato trovandosi in gruppo oppure in contatto online tramite Skype o Discord.

Il materiale è stato condiviso prevalentemente su GitHub o con merge effettuati “manualmente”:

NB: Sopra sono riportate le principali divisioni dei compiti tuttavia, vista la natura del progetto, è stata necessaria la partecipazione di ogni membro del gruppo in tutte le divisioni.

Problemi principali riscontrati:

Difficoltà iniziale di approccio a un nuovo linguaggio di programmazione, il che ha comportato l’allungamento dei tempi per svolgere compiti apparentemente semplici.

Un altro problema è stato che quando un individuo del gruppo apportava modifiche ad una parte del progetto, spesso, anche altre parti del progetto necessitavano di ulteriori modifiche che potevano rivelarsi difficili da trovare. *TimeStamp:*

Difficoltà nel capire da dove partisse il time millis

Creazione dei nodi:

Comprensione del parallelismo

Creazione delle catene:

Comprensione delle tre formule matematiche e relativa implementazione in jolie (in particolare il funzionamento della classe Math).

Problema nella ricezione dei dati fra un nodo e gli altri. Ad esempio i Jollar Buyer, Jollar Seller e il Total Network Jollar.

In generale il problema principale è stato nell' unire le varie componenti in modo che funzionasse il programma come complesso, ciò ha portato a ripetute modifiche.