Jollar
Cryptoccurence and BlockChain

Progetto svolto per il corso di

Sistemi Operativi, Informatica per il Management A.A. 2017-2018

- **Nome gruppo:** Answer42
- **Indirizzo e-mail di riferimento:** nicola.corea@studio.unibo.it
- **Componenti:**
  - **Corea Nicola,** Matricola: 0000731446;
  - **Vatalaro Giada,** Matricola: 0000773711;
  - **Myrtaj Arzana,** Matricola: 0000772501;
  - **Grassano Giulia,** Matricola: 0000765873;
  - **Morabito Diego**, Matricola: 0000772252;

INTRODUZIONE

“*Il progetto di quest'anno propone la creazione di un sistema di scambio elettronico decentralizzato, dove i partecipanti effettuano transazioni certificate all'interno della rete stessa, senza il bisogno di un organo centrale garante. L'implementazione del progetto segue i principi della programmazione orientata ai servizi, rispettandone l'architettura e i paradigmi di gestione della concorrenza. La comunicazione tra i servizi, infine, è implementata nel linguaggio visto a lezione di laboratorio: Jolie.*”

INDICE

SPECIFICHE DI PROGETTO

L'elaborato ha come obiettivo quello di rappresentare una rete decentralizzata, finalizzata allo scambio e alla generazione della criptovaluta “Jollar” basata sulle caratteristiche strutturali del “Bitcoin” e su quelle procedurali del “Primecoin”. Nello specifico, più avanti vengono illustrate queste caratteristiche basandosi sui concetti di: Rete Peer-to-Peer, Blockchain e Proof of Work.

Al contrario delle normali applicazioni Client–Server, questo progetto basa la propria architettura su una rete Peer-to-Peer (P2P), ovvero una rete costituita da connessioni non centralizzate (non si fa riferimento a un server centrale) ma da una serie di “client” (nodi) comunicanti tra loro e in continua sincronizzazione, sia per quanto riguarda le operazioni effettuate sia per le informazioni conservate.

Il modello di rete P2P proposto da questo progetto ha però un'entità in più, il timestamp, atta a registrare la date e l'ora di ogni connessione da parte di un nodo e altre features approfondite nel corso di queste pagine.

Ogni nodo presente e attivo sulla rete svolge delle operazioni dopo ogni transazione (una transazione consiste nello scambio di criptovaluta da un nodo verso un altro) finalizzate al compimento di una prova di lavoro (POW) per stabilire, quale sarà il nodo che dovrà farsi carico di aggiornare lo stato della blockchain e comunicarlo agli altri nodi.

Per Blockchain intendiamo appunto una “catena di blocchi” o sequenza di blocchi, che può essere interpretata come una struttura dati, condivisa tra tutti i nodi e quindi consultabile e modificabile (previa POW).

Ogni nodo quindi compie delle transazioni, al termine di ciascuna di esse viene lanciata una proof-of-work contenente il teorema di Fermat che convalida il blocco.

A questo punto il nodo che ha finito prima la POW inserisce il nuovo blocco all'interno della struttura dati, aggiorna gli altri nodi sulla stato della blockchain e riceve una ricompensa di 6 jollar.

STRUTTURA DEL PROGETTO

Il progetto è strutturato seguendo le risorse proposte dalle specifiche.
STRUTTURA: (Bitcoin) (https://bitcoin.org/bitcoin.pdf)
PROCEDURALITA': (Primecoin) (http://primecoin.io/bin/primecoin-paper.pdf)
La struttura è basata sui concetti di: **P2P - Nodo - Transazione - POW - Validazione - Blocco - Catena.**

Con **"P2P"** indichiamo un particolare tipo di rete serverless per cui ogni partecipante non fa affidamento su un'entità centrale ma comunica indistintamente e senza vincoli con tutte le entità partecipanti.

L'entità **"Nodo"** indica un partecipante alla rete P2P, composta quindi da più nodi comunicanti tra loro, con il compito di eseguire transazioni, creare blocchi inerenti alle transazioni e aggiornare lo stato della blockchain. Ogni nodo ha un identificativo univoco e 2 chiavi, una privata e una pubblica, conosciuta da ogni partecipante alla rete, inoltre ogni transazione implica una prova di lavoro (POW).

Per **"Transazione"** indichiamo lo scambio di oggetti e/o dati (nel nostro caso jollar) tra un nodo seller e un nodo buyer, ogni transazione deve essere verificata, approvata, e archiviata in modo da poter essere inserita all'interno di un blocco.

Per **"POW"** (Proof-of-Work) intendiamo un'insieme di operazioni generatrici delle catene di Cunningham (catene di numeri primi) atte a dimostrare (dare prova) di aver compiuto una certa quantità di lavoro.

"*Questo metodo consiste nell'obbligare i nodi che vogliono scrivere un blocco a cercare un valore che sia difficile da trovare e di cui sia facile controllarne la correttezza da parte degli altri nodi che vogliono validare la scrittura.*"

Per **"Blocco"** intendiamo uno spazio dedicato allo stoccaggio delle informazioni delle transazioni (nel nostro caso una sola, ma potrebbero esserci N transazioni per blocco). Ogni blocco ha un identificativo e 2 stringhe hash per codificare inequivocabilmente il blocco dato e concatenarlo al blocco precedente.

Per **"Catena"** intendiamo la struttura al vertice dei blocchi detta "BlockChain". Essa può essere considerata come un archivio (database) dei blocchi e quindi delle transazioni avvenute e validate.

IMPLEMENTAZIONE

L'implementazione ha richiesto lo studio pregresso di cosa volesse dire rete P2P e Blockchain, la strategia seguita per questo scopo è stata quella di suddividere in quante più sezioni possibile ogni componente dell'elaborato, nello specifico si è proceduto all'implementazione di 10 file elencati e descritti di seguito:

Il file "BlockChain.iol": facendo riferimento all'esempio contenuto nella repository gitHub del professor Zingaro, abbiamo implementato la struttura dati della Blockchain attraverso l'uso dei data type void "*Blockchain - GenesisBlock - Block - Transaction - Node*" questi sono i pilastri su cui si basa la struttura. All'interno dello stesso file è stata implementata un'interfaccia per ciascun partecipante alla rete denominata BlockChain. Nello specifico sono stati inizializzati 17 servizi "one way" e 5 servizi "RequestResponse", questi servizi sono utilizzati dai partecipanti alla rete come servizi di interscambio, aggiornamento ed istruzione.

I file "POW.ol" e "Genesis.ol": questi file contengono istruzioni che vengono richiamate dai nodi. Nello specifico il file POW.ol contiene tutte le operazioni della Proof-Of-Work e del teorema di Fermat per la convalida, viene richiamato dopo ogni transazione, da tutti i nodi contemporaneamente. Il file Genesis.ol invece viene richiamato una sola volta dal primo nodo connesso alla rete che si prende il compito di popolare la blockchain con il primo blocco (il blocco genesi appunto) e ottenere la ricompensa iniziale. **Il file "timestamp.ol"**: implementa un server timestamp (previsto dalla prima versione bitcoin) che si occupa di registrare data e ora delle transazioni e delle connessioni, inoltre si occupa di gestire il semaforo che determina il nodo più veloce a compiere la POW e ha il compito di comunicare le informazioni sui nodi connessi e lo stato dei wallet al NetworkVisualizer. **I file "nodo1.ol", "nodo2.ol", "nodo3.ol", "nodo4.ol"**: sono i nodi partecipanti alla rete, nel nostro caso 4 perché richiesti dalla demo; contengono le istruzioni per le transazioni e i richiami alla POW, grazie al loro sistema di porte sono connessi contemporaneamente a ogni entità della rete e possiedono quindi sia le stesse proprietà di un server che di un client.

Il file "networkVisualizer.ol": non è altro che un visualizzatore dello stato più recente della blockchain, riceve lo stato dei nodi e della blockchain con le relative transazioni e lo mostra all'utente.

SOLUZIONI ADOTTATE

Durante l'implementazione ed i vari ragionamenti non pochi sono stati i problemi. Innanzitutto il concepimento di cosa realmente fosse una blockchain e di quali fossero i principali componenti e di come potessero essere implementati. La soluzione più ovvia e decisa ad unanimità è stata quella di seguire alla lettera la struttura presentata nelle specifiche, aggiungendo alcuni type che si sono rilevati necessari dopo aver consultato alcune discussioni sul Newsgroup.

Sistema peer-to-peer: il problema principale è stato quello di concepire come implementare una rete peer-to-peer in Jolie. Inizialmente l'idea era di creare un servizio di binding dinamico delle porte, in modo tale da avere un unico file "nodo.ol" da poter lanciare più volte e che, a ogni avvio, creasse un nuovo nodo. Questa soluzione sarebbe stata la più efficiente in quanto avrebbe permesso non solo una minore quantità di codice e meno ripetizioni, ma anche una struttura logica delle porte di comunicazione dinamica e più sicura. Questa implementazione però non è stata adottata in quanto abbiamo dovuto confrontarci con l'uso e la comprensione di un linguaggio di programmazione nuovo e orientato ai servizi e non più ad oggetti; questo ha portato via più di un mese di lavoro per cui per necessità di tempo si è adottata la soluzione più intuitiva, ovvero l'inserimento di N inputport ed N outport per ogni componente della rete verso qualunque altro componente. Questa soluzione purtroppo non è la migliore e neanche la più efficiente da un punto di vista implementativo, gestionale e di un futuro aggiornamento della piattaforma.

RIFERIMENTI:

https://github.com/stefanopiozingaro/so_lab/blob/master/project/code/timestamp.ol TIMESTAMP

https://groups.google.com/forum/#!topic/infoman-so/76_Ft6veVTE TIMESTAMP

https://groups.google.com/forum/#!topic/infoman-so/cvsdmGEz6_0 BROADCAST

https://groups.google.com/forum/#!topic/infoman-so/YlW_sy2Ktao NODE

https://groups.google.com/forum/#!topic/infoman-so/yI-YTdrLaAA PUBLIC KEY - PRIVATE KEY

Una volta dichiarata l'interfaccia abbiamo notato che era fondamentale creare due type rivolti: uno al blocco Genesis, blocco origine della BlockChain, indicato con un indice uguale a 0 (sebbene indicato con 1 nella versioni iniziali di bitcoin); l'altro a tutti gli altri blocchi della catena. A questo proposito abbiamo scelto di eliminare l'asterisco proposto dal Professore (significante infinite transazioni) poiché in questa implementazione è possibile una sola transazione per blocco.

Presa in considerazione la struttura di un blocco all'interno di una BlockChain, notiamo che la suddetta si compone di tre parti: dati del blocco, hash del blocco e hash del blocco precedente. Di conseguenza avevamo bisogno di una struttura dati per il blocco genesis che fosse differente dal resto dei blocchi e che non comprendesse nessun riferimento ad un blocco precedente.

RIFERIMENTI:

https://github.com/stefanopiozingaro/so_lab/blob/master/project/code/hash.ol

HASH-MD5

https://github.com/stefanopiozingaro/so_lab/blob/master/project/code/block.ol

STRUTTURA

```
type Blockchain : void {
    .block* : Block | GenesisBlock
}

type GenesisBlock : void {
    .id_block_G : int
    .hashBlock_G : string
    .previousBlockHash_G : string
    .difficulty_G : int
}

type Block : void {
    .id_block : int
    .hashBlock : string
    .previousBlockHash : string
    .difficulty : double
    .transaction : Transaction
    .chainLength : int
}
```

```
type Transaction : void {
    .hash_transaction : string
    .nodeSeller : string
    .nodeBuyer : string
    .jollar : int
    .timestamp : string
}

type Node : void {
    .id_node : string
    .publicKey : string
    .privateKey? : string
    .wallet : int
}
```

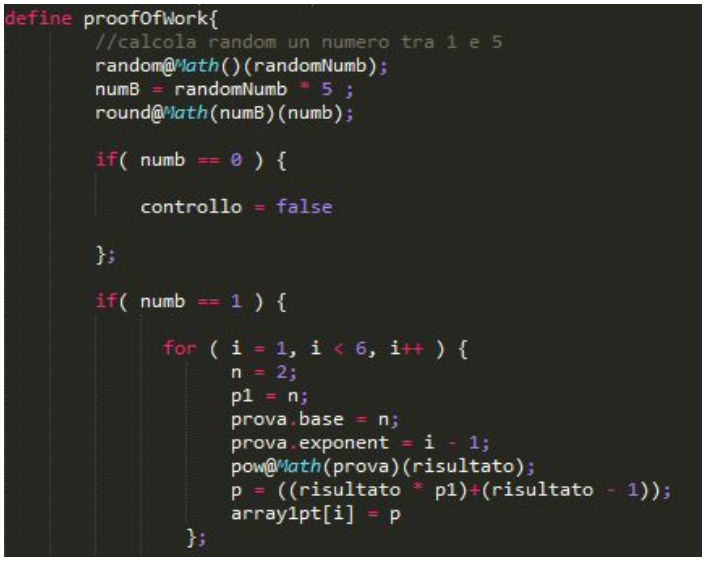
Per analizzare i numeri delle catene (che possono essere di Bi-Twin oppure di Cunningham, 1° o 2° tipo) , verificando se gli stessi sono primi o meno, si utilizza il piccolo Teorema di Fermat. Quindi le catene vengono validate.

La dimensione dei numeri primi delle catene viene limitata appositamente per

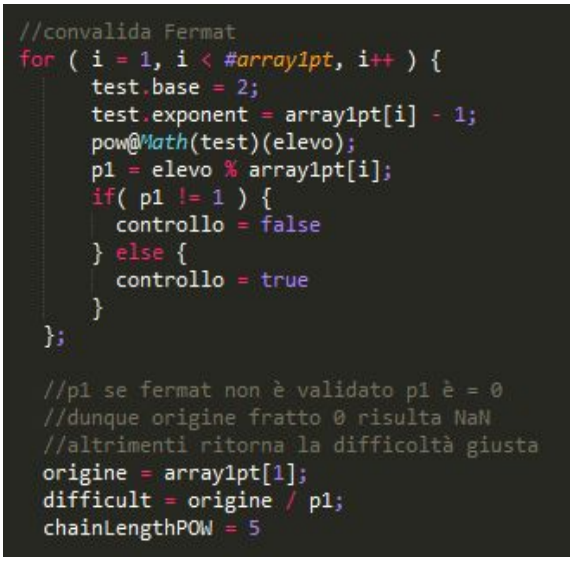
rendere più agevole il lavoro svolto e successivamente verificabile dai nodi connessi alla rete.
Per comprendere cosa si intende per difficoltà, abbiamo fatto riferimento a due post sul gruppo LSO riguardante questo argomento. Inizialmente infatti la difficoltà era data da una variabile incrementata a ogni transazione
https://groups.google.com/forum/#!topic/infoman-so/sL5myqwbKUg 7 giugno
https://groups.google.com/forum/#!topic/infoman-so/y2X-NZFURwc 13 settembre

Un altro problema è stato delineato dagli algoritmi che servono ad originare i differenti tipi di catena: andando ad analizzare le catene di cunningham abbiamo visto che un primo di Sophie Germain è un primo p tale che 2p + 1 sia primo; se continuiamo a raddoppiare e sommare 1 per un paio di volte si ottengono dei numeri primi, la sequenza ottenuta si chiama ''Catena di Cunningham" è sono di prima specie.
RIFERIMENTI
http://www.bitman.name/math/article/1585 (Fiorentini) prima specie

Durante l'implementazione grazie alla segnalazione di un'altro gruppo abbiamo potuto correggere la formula riguardante la generazione delle catene di numeri primi:
RIFERIMENTI:
https://groups.google.com/forum/#!topic/infoman-so/ywpjSVmjcpU FORMULA
https://groups.google.com/forum/#!topic/infoman-so/AX-0xXZSrFI FERMAT



```
define proofOfWork{
        //calcola random un numero tra 1 e 5
        random@Math()(randomNumb);
        numB = randomNumb * 5 ;
        round@Math(numB)(numb);

        if( numb == 0 ) {

            controllo = false

        };

        if( numb == 1 ) {

              for ( i = 1, i < 6, i++ ) {
                    n = 2;
                    p1 = n;
                    prova.base = n;
                    prova.exponent = i - 1;
                    pow@Math(prova)(risultato);
                    p = ((risultato * p1)+(risultato - 1));
                    array1pt[i] = p
                };
```



```
//convalida Fermat
for ( i = 1, i < #array1pt, i++ ) {
      test.base = 2;
      test.exponent = array1pt[i] - 1;
      pow@Math(test)(elevo);
      p1 = elevo % array1pt[i];
      if( p1 != 1 ) {
        controllo = false
      } else {
        controllo = true
      }
  };

  //p1 se fermat non è validato p1 è = 0
  //dunque origine fratto 0 risulta NaN
  //altrimenti ritorna la difficoltà giusta
  origine = array1pt[1];
  difficult = origine / p1;
  chainLengthPOW = 5
```

Una volta che un nodo completa il suo lavoro, sorge il problema di far sì che gli altri nodi non lavorino più' a quel compito. Si è ovviato al problema inserendo un semaforo all'interno del timestamp, che a sua volta invii un broadcast a tutti gli altri nodi intenti a svolgere lo stesso compito, affinché interrompano quello stesso lavoro già portato a termine dal nodo n.

DEMO

Istruzioni per la demo - le istruzioni per eseguire una demo:

- Avvio del Server TimeStamp;
- Avvio del Network Visualizer;
- Avvio del primo nodo;
- Avvio del secondo nodo;
- Avvio del terzo nodo;
- Avvio del quarto nodo;
- Confermare la prima transazione;
- Invio di un Jollar dal primo al secondo nodo;
- Confermare la seconda transazione;
- Invio di due Jollar dal primo al terzo nodo;
- Confermare la terza transazione;
- Invio di tre Jollar dal primo al quarto nodo;
- Richiesta al Network Visualiser di stampare la situazione della blockchain e della rete.

STRUMENTI E BIBLIOGRAFIA

STRUMENTI:

JOLIE 1.7.0: https://www.jolie-lang.org/index.html

SUBLIME TEXT: https://www.sublimetext.com/

GITLAB: https://about.gitlab.com/

GITHUB: https://github.com/

BIBLIOGRAFIA:

SPECIFICHE:

https://github.com/stefanopiozingaro/so_lab/blob/master/project/instructions.pdf

Gruppo LSO:	https://groups.google.com/forum/#!forum/infoman-so
GITHUB SPZ:	https://github.com/stefanopiozingaro/jolie
	https://github.com/stefanopiozingaro/so_lab
BITCOIN:	https://bitcoin.org/bitcoin.pdf
PRIMECOIN:	http://primecoin.io/bin/primecoin-paper.pdf
JOLIE:	https://jolielang.gitbook.io/docs