JOLLAR PROJECT

Documentazione ed istruzioni

ABSTRACT
Un progetto di che consiste nella creazione di un sistema di blockchain rispettando la programmazione orientata ai servizi in Jolie sviluppato da un gruppo di 5 studenti dell'Università di Bologna.

B. Marafini, A. Freda, F. Ventruto, A. Fabbri, R. Costanzo
Sistemi Operativi

Jollar Project – Laboratorio Sistemi Operativi A.A. 2017-2018

Table of Contents

0. Introduzione

- Nome del gruppo: BO.RO.LE
- Indirizzo mail di riferimento: bruno.marafini@studio.unibo.it /brunomarafinidj@gmail.com
- Componenti del gruppo:
    - Marafini, Bruno, 0000731900
    - Ventruto, Francesca, 0000733061
    - Costanzo, Riccardo, 0000732909
    - Fabbri, Alessandro, 0000739011
    - Freda, Alessandro, 0000772546

1. Generalità del progetto

Il progetto si propone di creare un sistema di scambio elettronico decentralizzato, in cui gli utenti effettuano transazioni certificate dalla rete stessa, senza il bisogno di un organo centrale garante (come ad esempio una banca).

Ci siamo proposti di ricreare una **Blockchain**, affrontando non pochi problemi, e quindi una "catena di blocchi", che può essere semplificata come un processo in cui un insieme di soggetti (definiti nodi) condivide risorse informatiche (memoria, CPU, banda) oppure anche monete o qualsiasi altro oggetto fittizio per rendere disponibile alla comunità di utenti un database virtuale generalmente di tipo pubblico (ma esistono anche esempi di implementazioni private) ed in cui ogni partecipante ha una copia dei dati.

Entrando più nello specifico si spiega come è stata concepita l'implementazione di questa blockchain: abbiamo inteso quindi la blockchain come una tecnologia che permette la creazione e gestione di un grande database distribuito per la gestione di transazioni condivisibili tra più **nodi** di una rete.

Si tratta di un **database strutturato in Blocchi** (contenenti una transazione) che sono tra loro collegati in rete in modo che ogni transazione avviata sulla rete debba essere **validata** dalla rete stessa nell'analisi di ciascun singolo blocco.

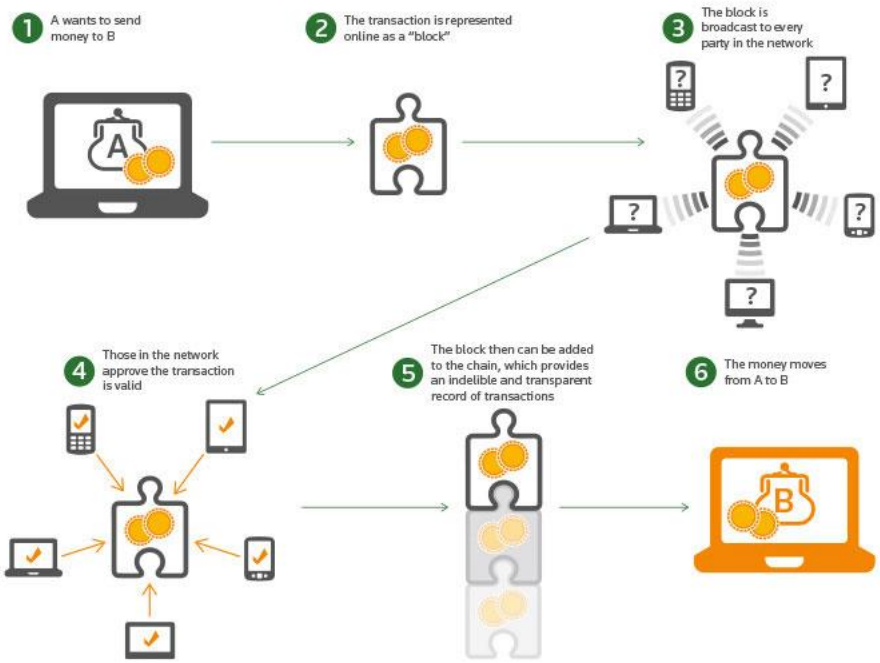
La Blockchain risulta cosi costituita da una **catena di blocchi che contengono ognuno una transazione.**

La soluzione per tutte le transazioni è affidata ai Nodi che sono chiamati a vedere, controllare ed approvare tutte le transazioni creando una rete che condivide su ciascun nodo l'archivio di tutta la blockchain e dunque tutti i blocchi con le relative transazioni.

Ciascun blocco è per l'appunto anche un archivio per ogni transazione eseguita nel sistema e per tutto lo storico di ciascuna transazione che, possono essere modificate solo con l'approvazione dei nodi della rete.

Nell'immagine sottostante vediamo nel concreto cosa siamo andati ad implementare, quindi cosa realmente sia la blockchain: ovvero una serie di **blocchi che archiviano un insieme di transazioni validate** e correlate da un **Marcatore Temporale** (Timestamp).

Ogni blocco include l'**hash** (una funzione algoritmica informatica non invertibile che mappa una stringa di lunghezza arbitraria in una stringa di lunghezza predefinita) che identifica il blocco in modo univoco e permette il collegamento con il blocco precedente tramite identificazione del blocco precedente, o meglio l'hash del blocco precedente.



Vediamo infine i **componenti basilari di una Blockchain:**

- **Nodo:** sono i partecipanti alla Blockchain, ovvero utenti, e sono costituiti fisicamente dai server di ciascun partecipante, ogni nodo ha un id univoco ed un hash univoco, oltre che una chiave privata che nessun altro nodo o la stessa rete può conoscere.
- **Transazione:** è costituita dai dati che rappresentano i valori oggetto di "scambio" e che necessitano di essere verificate, approvate e poi archiviate, inoltre ogni transazione ha un hash, il numero di jollar trasferiti, il nodo seller ed il nodo buyer;
- **Blocco:** è rappresentato dal raggruppamento di un insieme di transazioni (nel nostro caso invece da una ed una sola transazione per blocco, ogni blocco identificato da un id) che sono unite per essere verificate, approvate e poi archiviate dai partecipanti alla Blockchain;
- **Ledger:** è il registro pubblico (per noi un file .log) nel quale vengono "annotate" con la massima trasparenza e in modo immutabile tutte le transazioni effettuate in modo ordinato e sequenziale. Il Ledger è costituito dall'insieme dei blocchi che sono tra loro incatenati tramite una funzione di crittografia e grazie all'uso di hash;
- **Hash:** è un'operazione (non invertibile) che permette di mappare una stringa di testo e/o numerica di lunghezza variabile in una stringa unica ed univoca di lunghezza determinata. L'Hash identifica in modo univoco e sicuro ciascun blocco. Un hash non deve permettere di risalire al testo che lo ha generato.

Il progetto è sviluppato seguendo i principi della programmazione orientata ai (micro)servizi, in cui la comunicazione tra i processi è implementata nel linguaggio visto a lezione: Jolie.

2. Istruzioni per la DEMO

Si elencano delle semplici istruzioni per eseguire una demo corretta del sistema di blockchain (da Terminale) :

1. Avviare il Server Timestamp – sarà colui che detterà i tempi e stabilirà il momento esatto in cui una precisa transazione sarà stata effettuata – eseguendo il file *serverTimestamp.ol*
2. Avviare il Network Visualizer – sarà colui che con la sua relativa interfaccia stamperà informazioni relative ad ogni nodo/utente presente all'interno della blockchain insieme ad altre informazioni importanti ed utili – eseguendo il file *networkVisualizer.ol*
3. Avviare il primo nodo, quindi creare un genesis block con un relativo reward di 6 jollar – semplicemente eseguendo il file *nodo1.ol*
4. Avviare secondo, terzo e quarto nodo con conseguente *download* della blockchain – semplicemente eseguendo i file nell'ordine che segue, *nodo2.ol, nodo3.ol, nodo4.ol*
5. Inviare un jollar dal primo al secondo nodo con conseguente scrittura su un nuovo blocco e reward relativo – questa semplice procedura vi verrà richiesta all'avvio del *nodo1.ol* sotto forma di transazione – *"Vuoi eseguire una nuova transazione?"*
6. Lo stesso vale per l'invio di 2 jollar dal primo al terzo nodo e per l'invio di 3 jollar dal primo al quarto – la procedura è la stessa semplicemente occorrerà cambiare il numero di jollar da inviare ed il destinatario quando richiesto
7. Per quanto riguarda la richiesta al Network Visualizer di stampare la situazione relativa a jollar presenti nella rete ed ai relativi possessori con l'elenco delle transazioni avvenute basterà terminale le 4 transazione e rispondere si alla domanda: *"Vuoi visualizzare la situazione generale jollar/transazioni presenti nella rete?"*
8. Fine della demo.

3. Discussione sulle strategie di implementazione

Iniziamo con il discutere la nostra strategia di implementazione partendo dalla base dei servizi implementati e dei file contenuti all'interno della relativa cartella del progetto.
Come prima cosa abbiamo creato un file *blockchainInterface.iol* all'interno del quale abbiamo inserito i diversi *type* ed i vari servizi *One_way* oppure *Request_response* che fanno da tramite (ovvero permettono di creare la operation da utilizzare nei vari file per la gestione dei servizi) per quello che andiamo ad implementare all'interno dei **Nodi** e del **Server Timestamp**.
Ovviamente essendo una rete P2P (Peer-to-peer), ovvero un modello di architettura logica di rete informatica in cui i nodi non sono gerarchizzati unicamente sotto forma di **client** o **server** fissi, ma anche sotto forma di *nodi equivalenti o 'paritari'*, potendo quindi fungere allo stesso tempo da client o da server verso gli altri nodi terminali (**host**) della rete, i nodi fungeranno sia da client che da server rispettando uno schema di porte che allego in PDF di seguito.

Durante l'implementazione ed i vari ragionamenti non pochi sono stati i problemi:

1. Innanzitutto il concepimento di cosa realmente fosse una blockchain e di quali fossero i principali componenti e di come potessero essere implementati. Possiamo dire che questo non è stato un vero e proprio problema bensì una semplice discussione di gruppo che ha portato fuori una componente di conoscenza generale del progetto in se per se.
2. Sistema peer-to-peer: il problema principale è stato nel concepire come implementare una rete peer-to-peer in jolie. Inizialmente essendo un linguaggio a noi poco conosciuto e disponendo di poca documentazione online a parte quella sul sito del linguaggio. Principalmente riuscire a comprendere come, non essendoci un'architettura centralizzata, quindi non essendoci né client né server, potessimo far sì che i nodi facessero da client e da server allo stesso tempo. Questo problema è stato risolto implementando delle porte di input per i nodi server e delle porte di output per i nodi client di modo che i nodi potessero comunicare tra di loro.
3. Per quanto riguarda i messaggi broadcast non riuscivamo a comprendere come riuscire ad implementare un servizio che in parallelo facesse sì che un nodo potesse notificare gli altri per invitare a fare qualcosa oppure per notificarli di qualche avvenimento in particolare, visto che i nodi dispongono di porte di collegamento tra di loro, con protocollo *sodep*, ma comunque non si conoscono tra di loro. In jolie semplicemente implementando delle richieste separate dal simbolo | che sta a significare che sia l'operatore di destra che quello

di sinistra vengano eseguiti *concorrentemente, quindi in parallelo*; questo è un sistema molto utile quando, come in questo caso, si ha a che fare con molteplici servizi e si vogliono minimizzare i tempi di attesa gestendo molteplici comunicazioni in una sola volta.

4. Durante l'implementazione ci siamo resi conto che all'interno dell'interfaccia dove abbiamo dichiarato tutti i tipi (***Type***) era necessario implementare due Type per quanto riguardasse i blocchi; un tipo riguardante il **Blocco Genesis**, che si chiama tale in quanto è il primo blocco di una blockchain, ed è rappresentato solitamente con indice = 0 quindi blocco 0 anche se nelle prime versioni di **bitcoin** veniva considerato con indice = 1 mentre un altro tipo riguardante gli altri blocchi di nome **Block**. Il motivo per cui li abbiamo separati nella rappresentazione dell'albero dei Data Types è molto semplice: riguardando la struttura di un blocco all'interno di una blockchain vediamo che si compone di 3 dati, rispettivamente, **dati del blocco, hash del blocco (uguale a fingerprint come identificativo) ed hash del blocco precedente –** si può quindi facilmente intuire che il blocco genesis essendo il primo non può puntare ad un blocco precedente, e si compone quindi di attributi diversi rispetto ad un blocco qualsiasi (successivo) all'interno della blockchain.
5. Ulteriori problemi sono stati apportati dalla **Proof of work** della quale ne viene richiesta un'implementazione simile a quella utilizzata dalla moneta elettronica **Primecoin**: si tratta di un'implementazione di un sistema di Proof of work unico per la sua sicurezza e particolare per il suo sviluppo. Il tutto consiste nel cercare delle catene di numeri primi rispettivamente **Catene di Cunningham (1° e 2° Tipo) e Catene Bi-Twin** che poi dovranno essere validate tramite il **Piccolo Teorema di Fermat** che non fa altro che analizzare ogni singolo numero delle catene per verificare se effettivamente sia primo o meno. Questo sistema è stato designato di modo che il lavoro effettuato sia facilmente verificabile da parte di tutti i nodi all'interno della rete; ovviamente per far sì che tutto ciò sia possibile la dimensione dei numeri primi all'interno del sistema non potrà essere troppo elevata.
    a. Il primo problema si è verificato nel momento in cui cercavamo di capire a cosa si dovesse ricondurre la difficoltà di una catena, alla quale, in un primo momento, non avevamo fatto altro che associare la lunghezza della catena. Abbiamo successivamente visto all'interno di un documento di **Sunny King** (King, 2013) che per misurare la difficoltà di una catena occorre effettuare un semplice ragionamento: *teniamo conto che **k** rappresenti la **lunghezza** della catena. La*

catena sarà rappresentata da $p_0, p_1, \dots, p_{k-1}$. *Teniamo conto che* ***r*** *sia il* ***resto dell'applicazione del piccolo teorema di Fermat*** *del numero successivo all'interno della catena* p_k. ***Possiamo affermare che*** p_k / r ***sia l'operazione utilizzata per calcolare la difficoltà della catena.***

b. Il secondo problema è stato rappresentato dagli algoritmi utili a generare i diversi tipi di catena: studiandoci bene come fossero 'costruite' le catene di cunningham ci siamo resi conto che un **primo di Sophie Germain** è un **primo** p tale che *2p + 1* sia primo; se continuando a raddoppiare e sommare 1 per alcune volte si ottengono numeri primi, la sequenza risultante si dice *"catena di Cunningham di prima specie"* o semplicemente *"catena di Cunningham"* in onore di Allan Cunningham. Come abbiamo potuto apprendere da un documento di (Fiorentini), esiste una tabella che riporta i primi iniziali delle catene di Cunningham di prima specie di lunghezza almeno 3 minori di 1000, inoltre ve ne è un'altra che mostra il minimo primo in una catena di esattamente *n* termini.

Tabella 1 - Primi iniziali catene di Cunningham 1° Tipo

Primo iniziale	Lunghezza della catena
2	5
5	4
11	3
41	3
89	6
179	5
359	4
509	4
719	3
1019	3
1031	3
1229	4
1409	4
1451	3
1481	3
1511	3
1811	3
1889	3
1901	3
1931	3
2459	3
2699	4
2819	3
3449	3
3491	3
3539	4
3821	3
3911	3
5081	3
5399	3
5441	3
5849	3
6101	3
6131	3
6449	4
7079	3
7151	3
7349	3
7901	3
8969	3
9221	3

Tabella 2 - Minimo primo in una catena di esattamente n termini

n	Minimo primo
1	13
2	3
3	41
4	509
5	2
6	89
7	1122659
8	19099919
9	85864769
10	26089808579
11	665043081119
12	554688278429
13	4090932431513069
14	95405042230542329
15	113220800675069784839
16	810433818265726529159

c. Abbiamo notato un piccolo errore all'interno della documentazione all'interno della formula delle catene di cunningham; la formula contenuta all'interno della documentazione è la seguente: $p_i = 2^{i-1}p_i + (2^{i-1} + 1)$ mentre invece la formula **corretta** è la seguente: $p_i = 2^{i-1}p_i + (2^{i-1} - 1)$

6. Un altro problema che abbiamo avuto è stato il cercare di capire come implementare semafori all'interno del progetto; principalmente cercare di capire dove andare ad implementarli ma prima ancora, cosa il semaforo dovesse fare. Alla fine abbiamo deciso di utilizzare il semaforo per far sì che il primo nodo che finisca la proof-of-work mandi una notifica in broadcast agli altri nodi dicendo di stopparsi avendo lui stesso fatto la verifica delle catene, evitando che gli altri nodi la verifichino nuovamente. Il problema consisteva ancora prima nell'identificazione del 1 nodo che finisse la POW ed è stato risolto con l'inserimento di un semaforo all'interno del **Server Timestamp**.

Tabella 3 - Uso del semaforo

```
/*
 Servizio che determina quale nodo finisce per primo
 la proof of work
 */

[finePOW(nomeNodo)(nodoprimo){

 acquire@SemaphoreUtils(semaforo1pow)(res);

 //sezione critica
 nodoprimo = nomeNodo;

  //mando agli altri nodi il nome del nodo che ha finio per primo la POW
  fineSemaforo@PP14(nodoprimo)
  ;
  fineSemaforo@PP15(nodoprimo)
  ;
  fineSemaforo@PP16(nodoprimo)
  ;
  fineSemaforo@PP17(nodoprimo)
 ;

  release@SemaphoreUtils(semaforo1pow)(res)

}]
```

7. Infine abbiamo notato che all'interno della documentazione fornita dal Professore, all'interno del codice di partenza al lato del tipo Transaction vi era un * che sta a significare che ci possono essere infinite transazioni; mentre invece siccome sappiamo che ci può essere per questa implementazione **solamente una transazione per blocco** abbiamo semplicemente eliminato l'asterico (*).

Bibliography

Fiorentini, M. (s.d.). *bitman.* Tratto da bitman.name: http://www.bitman.name/math/article/1585

King, S. (2013, July 7). *Primecoin: Cryptocurrency with Prime Number Proof-of-Work.* Tratto da primecoin.io: http://primecoin.io/bin/primecoin-paper.pdf