

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

Per un Pugno di Jollar

giacomo.minello@studio.unibo.it

September 16, 2018

Abstract

Questo documento ha lo scopo di essere un report dettagliato sulle modalità di esecuzione e le scelte implementative della blockchain Jollar. Verranno indicati i contatti dei membri del gruppo, si procederà quindi ad una breve descrizione delle caratteristiche del progetto. Seguono le istruzioni ed i comandi di esecuzione per una demo che è stata preparata. Infine si discuteranno le strategie implementative con particolare riguardo all'architettura dei servizi.

Indice

1	Contatti	2
2	Descrizione del progetto	3
3	Istruzioni per la demo	5
4	Discussione delle strategie di implementazione	6
4.1	Organizzazione	6
4.2	Struttura del progetto	6
4.2.1	Struttura di un nodo	6
4.2.2	Struttura di main.ol	7
4.2.3	Struttura del Network Visualizer	8
4.2.4	Struttura della demo	8
4.3	Descrizione delle feature	8
4.3.1	Decentralizzazione	8
4.3.2	Service-oriented recursion	9

1 Contatti

Contatti del gruppo

- Minello Giacomo^a, Matr. 0000802402,
giacomo.minello@studio.unibo.it
minellogiacomo@gmail.com
- Aspromonte Marco, Matr. 0000806519
marco.aspromonte@studio.unibo.it
marco.aspromonte@gmail.com
- Menetto Davide, Matr. 0000768828
davide.menetto@studio.unibo.it
davide.menetto96@gmail.com
- Morselli Enrico, Matr. 0000806725
enrico.morselli@studio.unibo.it
- Memoli Alessandro, Matr. 0000806773
alessandro.memoli@studio.unibo.it
alessandromemoli@hotmail.it

^aReferente del gruppo

2 Descrizione del progetto

Il progetto assegnato prevede la creazione di una blockchain, ovvero il codice dei nodi che partecipano ad una rete P2P per lo scambio e la verifica di transazioni basate sulla cryptocurrency Jollar. Dopo la formazione del gruppo ci siamo riuniti per discutere su come affrontare la progettazione e lo sviluppo. Ci siamo posti come obbiettivo quello di creare una cryptocurrency funzionante o almeno del codice funzionale alla creazione di una criptocurrency che ricalchi quanto più possibile Bitcoin e Primecoin; questa influenza si può notare nel file `maininterface.ol`. A seguito di problemi di sviluppo in particolare dovuti ad alcuni limiti del linguaggio Jolie siamo giunti ad un compromesso implementativo. Il progetto, così come è implementato, prevede una rete *fully connected* di 4(+2) nodi, di conseguenza l'importanza degli algoritmi e gli accorgimenti per la diffusione della blockchain risulta praticamente nulla. Questo fatto tuttavia ci ha lasciato più spazio di azione per poterci concentrare sull'utilizzo di strutture dati più complete (es. `Block type`). Nel progetto sono state implementate transazioni di tipo UTXO piuttosto che un sistema *account-based* sia per una questione di aderenza al protocollo bitcoin, sia per una preferenza stilistica. L'utilizzo del linguaggio Jolie e la scarsa mole di documentazione ci ha rallentato notevolmente nella progettazione dell'architettura della rete, nella navigazione di strutture dati articolate ma soprattutto nell'implementazione di protezioni crittografiche adeguate e nelle parti *compute-intensive*. Riguardo al primo punto la scarsa documentazione ci ha più volte ingannato in fase progettuale: la soluzione ideale sarebbe stata una comunicazione di tipo *broadcast* non realizzabile dato l'utilizzo esclusivo del protocollo TCP. In questo caso, ovvero per noi programmatori poco esperti, quello che nel linguaggio Jolie è considerato un vantaggio, ovvero l'astrazione della comunicazione tra servizi, si è rivelato un impedimento che si sarebbe potuto evitare con una documentazione più ampia e matura. Abbiamo quindi ripiegato su una soluzione più spartana e non parallela: *dynamic binding* all'interno di un ciclo `for` che ci permettere di scorrere la lista dei nodi sulla rete. Questa soluzione è stata l'unica che si è dimostrata funzionale in quanto nemmeno l'utilizzo di altri costrutti architetturali messi a disposizione da Jolie (*aggregators* e *redirectors*) avrebbe portato ad un cambiamento nel comportamento effettivo del codice; si sarebbe aggiunto solo del *syntactic sugar* che non avrebbe portato nessun vantaggio sia a livello di comprensione del codice che come tempo di sviluppo. Per quanto riguarda l'utilizzo di strutture dati la scarsa documentazione unita alle numerose opzioni e metodologie per operare con le variabili ha richiesto del tempo aggiuntivo per la comprensione dell'argomento e la riscrittura di alcuni *snippets* di codice. Ciò che più ci ha sorpreso però è

stato l'embedding di servizi Java. Nella documentazione se da molto spazio ed è presente anche un post che ne spiega la procedura ma non siamo riusciti a far funzionare questa feature. Premettendo che l'errore potrebbe essere da parte nostra nella comprensione ciò sarebbe comunque indice di una documentazione non efficace. Per questo motivo nel progetto non è implementato un'algoritmo di crittografia quali ad esempio RSA o ECDSA disponibili in Java 9 ed anche perché come best practice è sconsigliato implementare il proprio algoritmo crittografico dato che il rischio di incorrere in errori è molto alto, dando un'errata sensazione di sicurezza. Allo stesso modo per quanto riguarda la proof-of-work, la mancanza di librerie matematiche adeguate e l'impossibilità di utilizzarne una fortina dal linguaggio Java ne ha limitato la rilevanza pratica e rendendone di fatto impossibile un'implementazione congrua con le specifiche del protocollo primecoin.

3 Istruzioni per la demo

La sequenza di esecuzione è la seguente:

1. avvio del Nodo 1;
`jolie node1.ol`
2. avvio del Nodo 2;
`jolie node2.ol`
3. avvio del Nodo 3;
`jolie node3.ol`
4. avvio del Nodo 4;
`jolie node4.ol`
5. avvio del Network Visualizer
`jolie networkVisualizer.ol`
6. avvio di DemoTX.ol
`jolie demoTX.ol`

4 Discussione delle strategie di implementazione

In questa sezione sono analizzate la struttura del progetto, le scelte e i vincoli implementativi e la descrizione specifica delle singole feature.

4.1 Organizzazione

Dover sviluppare un progetto orientato ai microservizi per ragioni intrinseche al linguaggio utilizzato ha reso necessario, ma soprattutto conveniente, optare per un'organizzazione in sub-team. Analizzando le specifiche abbiamo costituito tre sottogruppi che corrispondono ai tre principali set di funzioni della blockchain: rete, transazioni e blocchi. Per determinare la dimensione e il numero di questi sottogruppi ci siamo avvalsi della tecnica del "two-pizza-box team" ovvero "If you can't feed a team with two pizzas, it's too large."¹. Per ottenere un codice uniforme anche dal punto di vista stilistico ci siamo accordati per utilizzare la notazione camelCase. Il codice è stato ospitato su Gitlab, piattaforma in cui abbiamo settato una pipeline di build in modo da poter controllare in maniera agevole la correttezza della sintassi (purtroppo l'utilizzo di CI/CD non è banale quando si inizia a parlare di rete, quindi questi test sono stati svolti manualmente). Per evitare di incorrere in problemi di merge abbiamo deciso di gestire ogni modifica tramite altre forme di comunicazione (diretta, mail, chat, etc.) e incaricare un solo membro di caricare i commit sulla repository.

4.2 Struttura del progetto

4.2.1 Struttura di un nodo

Riguardo alla struttura del codice è bene iniziare analizzando la struttura di un nodo.

```
constants {  
  ROOT="socket://localhost:900",  
  CREATEGENESISBLOCK = true,  
  ID="1",  
  LOCATION="socket://localhost:9001"  
}  
include "main.ol"
```

Si osserva quindi che, contrariamente a quanto ci si aspetti, un file nodoN.ol contiene una piccola porzione di codice. Per massimizzare la code reusability e per agevolare la mantenibilità del codice abbiamo scelto di identificare

¹Siamo molto affamati

un nodo con quattro costanti (di cui solo tre caratteristiche di ogni nodo). Queste costanti sono in ordine:

- ROOT, costante comune a tutti i nodi, è la radice di una location ed è stata inserita per evitare di avere ogni indirizzo hardcoded nel codice.
- CREATEGENESISBLOCK, è una costante booleana con valore true solo per il primo nodo della rete, il suo scopo si può dedurre dal nome.
- ID, questo valore corrisponde al numero n del nodoN, viene usato per identificare univocamente un nodo.
- LOCATION, indica l'indirizzo dell'inputPort del nodo, è stata inserita solo perché il linguaggio Jolie non permette di dichiarare una location tramite l'operazione sulle stringhe ROOT+ID.

Ogni nodo importa il file "main.ol" che contiene il codice comune a tutti i nodi.

4.2.2 Struttura di main.ol

Il file main.ol di cui riporteremo qui solo alcuni estratti per ragioni di chiarezza e lunghezza si occupa di importare alcune librerie standard del linguaggio Jolie per poter esporre al programma alcuni servizi utili. Per ragioni di compattezza abbiamo scelto di impostare solo due porte per ogni nodo, una di output e una di input. Per fare ciò abbiamo deciso di considerare la location dell'inputPort come univoca per ogni nodo e dichiarare la location dell'output port tramite dynamic binding. Questo file comprende sia l'inizializzazione del nodo, sia il main ed anche un gran numero di servizi embeddati internamente. Nell'inizializzazione del nodo si inizializzano alcune variabili e ci si occupa di creare il primo blocco della blockchain se necessario oppure richiedere una lista dei nodi attivi. Nell'inizializzazione inoltre viene creata una coda per immagazzinare le transazioni e si fa iniziare una service-oriented-endless-recursion per processare eventuali transazioni nella coda. Ciò permette che solo un'istanza del servizio alla volta sia in esecuzione. I servizi embeddati internamente hanno lo scopo di essere utilizzati come funzioni, unico inconveniente nell'usarle è l'impossibilità di accedere alle variabili globali del nodo. Tuttavia ciò non è per forza un difetto: è sempre preferibile limitare il più possibile l'accesso alle informazioni (information hiding) ed è favorevole alla concorrenza (se posso accedere ad una copia del dato occupo un semaforo per un periodo minore). Nel main sono presenti le input choice che permettono di rispondere alle richieste che arrivano alla porta di input (richieste che possono provenire sia da un nodo esterno che da un servizio embeddato internamente).

4.2.3 Struttura del Network Visualizer

il file `networkVisualizer.ol` contiene le istruzioni per richiedere ai nodi alcuni dati così come indicato nelle specifiche del progetto. Il suo funzionamento è semplice: mandiamo una richiesta ai nodi che abbiamo avviato e ne viene stampata la risposta.

4.2.4 Struttura della demo

Il file `demoTX.ol` contiene le istruzioni per far eseguire una piccola demo seguendo le specifiche indicate nel progetto. Per fare ciò abbiamo previsto l'uso di una struttura dati molto più semplice rispetto ad una transazione così come rappresentata nella blockchain:

```
type TxValue: void {  
  .value: long //in Jollaroshi 1Jollar=100,000,000 Jollaroshi  
  .location: string  
}
```

Questi dati stanno ad indicare il valore della transazione, in Jollaroshi², e l'identificativo del ricevente. Questi dati vengono comunicati al nodo che deve eseguire la transazione in modo che si occupi di formare una transazione con la struttura dati prevista. Infine si comunica al Network Visualizer il comando per far eseguire la richiesta dei dati ai nodi.

4.3 Descrizione delle feature

4.3.1 Decentralizzazione

Quello che ormai sarà saltato all'occhio è la mancanza del `networkTimestampServer`. Consapevoli di poter facilmente rimpiazzare questo punto di centralizzazione della rete abbiamo deciso di optare per un approccio distribuito. Abbiamo quindi immaginato un semplice "protocollo" che prevede per ogni nodo il calcolo del tempo medio della rete e la disponibilità a fornire ad altri nodi il proprio dato temporale. Allo stesso modo avremmo potuto sostituire il `networkVisualizer` e `demoTX` incorporandolo come servizio embeddato internamente ad ogni nodo, scelta che avrebbe reso il nodo più completo dal punto di vista formale se si guarda ad un full-node della rete bitcoin. Per evitare di condensare troppo codice in un unico file e per agevolare la manutenibilità e lo sviluppo abbiamo deciso di dimostrarne la fattibilità ma di non procedere con l'idea.

²L'equivalente di un Satoshi per il Bitcoin

4.3.2 Service-oriented recursion