Sistemi Cooperativi e Reti Sociali

Lezione 1

Daniele Margiotta, Gabriele Prestifilippo, Luca Squadrone

16 marzo 2019

Indice

Capitolo 1

Introduzione

Il corso tratterà della struttura **blockchain**, verranno analizzati i vari tipi di **criptovalute** e verrà studiato il progetto **Hyperledger** (un progetto di blockchain open source avviato a dicembre 2015 dalla Fondazione Linux[2] per supportare lo sviluppo collaborativo di registro distribuito).
Nel corso si utilizzerà principalmente il linguaggio **Golang** per alcune proprietà del linguaggio che lo rendono particolarmente fruibile.
In questa dispensa si tratta di un problema preliminare di comunicazione tra macchine su una rete, il **C10K Problem**.

1.1 Blockchain

La Blockchain è un protocollo di comunicazione che identifica una tecnologia basata sulla logica del database distribuito (un database in cui i dati non sono memorizzati su un solo computer ma su più macchine collegate tra loro, chiamate nodi). La Blockchain è 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 identifia il blocco in modo univoco e che permette il collegamento con il blocco precedente tramite identificazione del blocco precedente.

1.1.1 I componenti basilari della Blockchain:

- Nodo: sono i partecipanti alla Blockchain e sono costituiti fisicamente dai server di ciascun partecipante;
- Transazione: è costituita dai dati che rappresentano i valori oggetto di "scambio" e che necessitano di essere verifciate, approvate e poi archiviate;
- Blocco: è rappresentato dal raggruppamento di un insieme di transazioni che sono unite per essere verificate, approvate e poi archiavite dai partecipanti alla Blockchain;
- Ledger: è il registro pubblico 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.

Ciascun blocco contiene dunque diverse transazioni e dispone di un Hash collocato nell'header. L'Hash registra tutte le informazioni relative al blocco e un Hash con le informazioni relative al blocco precedente che permette di creare la catena e di legare un blocco all'altro. La transazione contiene invece informazioni relative all'indirizzo pubblico del ricevente, le caratteristiche della transazione e la firma crittografica che garantisce della sicurezza e dell'autenticità della transazione. La Blockchain è organizzata per aggiornarsi automaticamente su ciascuno dei client che partecipano al network. Ogni operazione è ettuata deve essere confermata automaticamente da tutti i singoli nodi attraverso software di crittografia, che verificano un pacchetto di dati definiti a chiave privata o seme, che viene utilizzato per firmare le transazioni. Garantendo l'identità digitale di chi le ha autorizzate.

1.1.2 Le principali caratteristiche della Blockchain

- Affidabilità: la Blockchain è affidabile. Non essendo governata dal centro, ma dando a tutti i partecipanti diretti una parte di controllo dell'intera catena, la Blockchain diventa un sistema meno centralizzato, meno governabile, e allo stesso tempo molto più sicuro e affidabile, ad esempio da attacchi di malintenzionati. Se infatti soltanto uno dei nodi della catena subisce un attacco e si danneggia, tutti gli altri nodi del database distribuito continueranno comunque a essere attivi e operativi, saldando la catena e non perdendo in questo modo informazioni importanti.

- Trasparenza: le transazioni effettuate attraverso la Blockchain sono visibili a tutti i partecipanti, garantendo così trasparenza nelle operazioni.

- Convenienza: effettuare transazioni attraverso la Blockchain è conveniente per tutti i partecipanti, in quanto vengono meno interlocutori di terze parti, necessari in tutte le transazioni convenzionali che avvengono tra due o più parti (ovvero le banche e altri enti simili).

- Solidità: le informazioni già inserite nella Blockchain non possono essere modificate in alcun modo. In questo modo le informazioni contenute nella Blockchain sono tutte più solide e attendibili, proprio per il fatto che non si possono alterare e quindi restano così come sono state inserite la prima volta.

- Irrevocabilità: con la Blockchain è possibile effettuare transazioni irrevocabili, e allo stesso tempo più facilmente tracciabili. In questo modo si garantisce che le transazioni siano definitive, senza alcuna possibilità di essere modificate o annullate.

- Digitalità: con la Blockchain tutto diventa virtuale. Grazie alla digitalizzazione, gli ambiti applicativi di questa nuova tecnologia diventano tantissimi.

1.2 Linguaggio Golang

Il linguaggio Go, chiamato anche Golang, è un linguaggio di programmazione open source sviluppato da Google, rilasciato ufficialmente nel novembre del 2009.

Golang è un linguaggio particolarmente fruibile per le seguenti proprietà:

- Presenza dei puntatori, che sono una struttura a stretto contatto con il codice a livello macchina;
- Presenza delle closures;
- Possibilità di effettuare return multipli, permettendo una migliore flessibilità a livello di lettura dei return delle funzioni;
- Linguaggio compilato JIT (just in time) che porta ad una velocità di compilazione quasi a livello macchina;
- Linguaggio "ordinato" di facile lettura. Difatti, a differenza del C, il Golang può essere letto da sinistra a destra, rendendo molto più chiara la lettura.

Capitolo 2

C10K Problem

2.1 Socket

Un socket è un endpoint software che stabilisce la comunicazione bidirezionale tra un programma server e uno o più programmi client. Il socket associa il programma server a una porta hardware specifica sulla macchina in cui viene eseguito in modo tale che qualsiasi programma client in qualsiasi punto della rete con un socket associato a quella stessa porta possa comunicare con il programma server.
Un programma server in genere fornisce risorse a una rete di programmi client. I programmi client inviano richieste al programma server e il programma server risponde alla richiesta.

2.2 Descrizione C10k Problem

Il problema C10k è il problema dell'ottimizzazione dei socket di rete per gestire un numero elevato di client allo stesso tempo. Il nome C10k è un "numeronym" per gestire simultaneamente diecimila connessioni. Da notare che le connessioni simultanee non sono le stesse richieste per secondo, sebbene siano simili: la gestione di molte richieste al secondo richiede un throughput elevato (elaborandoli rapidamente), mentre un elevato numero di connessioni simultanee richiede una pianificazione efficiente delle connessioni. In altre parole, la gestione di molte richieste al secondo riguarda la velocità di gestione delle richieste, mentre un sistema in grado di gestire un numero elevato di connessioni simultanee non deve necessariamente essere un sistema veloce, solo uno in cui ogni richiesta restituirà una risposta deterministicamente entro un lasso di tempo finito (non necessariamente fisso).
Il problema dell'ottimizzazione del socket server è stato studiato perché è necessario prendere in considerazione una serie di fattori per consentire a un server Web di supportare molti client. Ciò può implicare una combinazione di limitazioni del sistema operativo e limitazioni del software del server Web. In base alla portata dei servizi da rendere disponibili e alle capacità del sistema operativo, nonché a considerazioni sull'hardware come le capacità di multielaborazione , può essere preferito un modello multi-threading o un modello a thread singolo . In concomitanza con questo aspetto, che implica considerazioni sulla gestione

della memoria (di solito relative al sistema operativo), le strategie implicite si riferiscono agli aspetti molto diversi della gestione degli I / O. Il termine è stato coniato nel 1999 da Dan Kegel, citando l' host FTP di Simtel , cdrom.com , che serviva 10.000 client contemporaneamente oltre 1 gigabit al secondo Ethernet in quell'anno. Il termine è stato utilizzato per il problema generale di un numero elevato di client, con numeri simili per un numero maggiore di connessioni, più recentemente C10M nel 2010.
Nei primi mesi del 2010 sono stati possibili milioni di connessioni su un singolo server 1U di commodity: oltre 2 milioni di connessioni (WhatsApp , 24 core, usando Erlang su FreeBSD), 10-12 milioni di connessioni (MigratoryData, 12 core, usando Java su Linux). Le applicazioni comuni di un numero molto elevato di connessioni includono server pub / secondari, chat, file server, server Web e reti definite dal software.
Questo problema trova vari ostacoli nella gestione della memoria e nella gestione dell' I/O.

Capitolo 3

Web Server

Un web server deve gestire diverse richieste da parte di clienti, che comunicano con esso tramite dei browser.
Si analizzano alcune delle possibili soluzioni del problema, partendo dal modello di base che ci consente di gestire una singola connessione.

3.1 Pseudo-codice

Server:

1. creare un socket tramite funzione socket();
2. collegare il socket ad una porta tramite la funzione bind();
3. impostare le code per le richieste di arrivo tramite la funzione listen();
4. attendere una richiesta di connessione tramite la funzione accept();
5. prendere un messaggio dal socket tramite la funzione read();
6. inviare un messaggio tramite la funzione write();
7. chiudere la connessione tramite la funzione close().

Client:

1. creare un socket tramite funzione socket();
2. inviare una richiesta di connessione a una porta specificata (il client ha bisogno di conoscere l'indirizzo del server) tramite la funzione connect();
3. inviare un messaggio tramite la funzione write();
4. prendere un messaggio dal socket tramite la funzione read();
5. chiudere la connessione tramite la funzione close().

Già questo semplice esempio porta alla luce varie problematiche della gestione di una connessione. Al momento della ricezione non è immediatamente nota la lunghezza di un messaggio, non è noto il tipo di messaggio che sto ricevendo, non è noto se il messaggio sia un pacchetto di un messaggio più grande o se il messaggio sia chunked. Nel caso più paradossale il messaggio potrebbe essere una richiesta di escalation verso i web socket.

3.2 Tipi connessioni multiple

Non potendo prevedere tutte variabili prima di ricevere il messaggio, per ricevere messaggi multipli l'unica opzione è ricorrere a tecniche di multiprocessing o multithreading, o ricorrere a soluzioni asincrone

Si analizzano i vari tipi di soluzioni:

- Asincrona, che può essere descritta da macchine a stati finiti. Il throughput è molto basso, e non consente la ricezione di due messaggi contemporaneamente;
- Multitasking: come fa Apache, che quando riceve una richiesta effettua una fork (dispendiosa in tempo e calcoli) in quanto duplica il processo per lasciare il processo padre sempre in ascolto e assegnare il job al processo figlio. Questo per'o 'e rischioso in quanto se il processo padre termina, tutti i figli cessano di operare;
- Multithread: esegue uno switching a livello utente (piu' leggero del multitasking), ma troppi thread sono pesanti.

3.2.1 Server Multithread

1. creare un socket tramite funzione socket();
2. configurare le impostazioni della struttura dell'indirizzo del server;
3. impostare il numero di porta, usando la funzione htons() per usare l'ordine dei byte corretto;
4. impostare l'indirizzo IP del server;
5. associare la struttura dell'indirizzo al socket;
6. ascoltare sul socket con ad esempio 40 richieste di connessione massima in coda;
7. while(true)
   (a) creare un nuovo socket per la connessione in ingresso tramite la funzione accept();
   (b) per ogni richiesta client creare un thread e assegnare la richiesta del client ad esso per l'elaborazione e quindi il thread principale può intrattenere la prossima richiesta.

3.2.2 Server Multitasking

1. creare un socket tramite funzione socket();
2. configurare le impostazioni della struttura dell'indirizzo del server;
3. impostare il numero di porta, usando la funzione htons() per usare l'ordine dei byte corretto;
4. impostare l'indirizzo IP del server;
5. associare la struttura dell'indirizzo al socket;
6. ascoltare sul socket con ad esempio 40 richieste di connessione massima in coda;
7. while(true)
   (a) Accettare la chiamata e creare un nuovo socket per la connessione in entrata usando la funzione fork();