Università degli Studi di Napoli "Parthenope"
DIPARTIMENTO DI SCIENZE E TECNOLOGIE

Corso di laurea in Corso di Laurea in Informatica

Tesi di Laurea Triennale

PROGETTAZIONE E SVILUPPO DI UN SISTEMA DISTRIBUITO PER L'ANALISI DELLE TRANSAZIONI BITCOIN

Relatore
prof. Alessio Ferone

Laureando
Antonio Riccardi
mat. 0124000411

Anno accademico 2018 - 2019

Ai miei genitori.

Ringraziamenti

TODO
Grazie a soreta.

Sommario

La seguente tesi ha come obiettivo la realizzazione di un'applicazione distribuita in grado di fare analisi di transazioni provenienti dalla blockchain di Bitcoin.
Il nocciolo della questione affrontata è quello di studiare la struttura di ogni singola piattaforma distribuita e di implementare componenti in grado di ricavare informazioni utili per l'analisi delle transazioni. In particolare, ci si soffermerà sulle principali tecnologie informatiche che permettono di costruire sistemi distribuiti, come Spark ed Hadoop, e sulle strutture dati che facilitano l'immagazzinamento delle informazioni come Neo4j.
Nella prima parte della tesi si procede con l'introduzione alla tecnologia alla base della moneta virtuale Bitcoin, facendo luce su cos'è la Blockchain e quali vantaggi/svantaggi presenta rispetto ad altri sistemi che gestiscono transazioni monetarie. A seguire, sono presi in esame le principali soluzioni di analisi di transazioni già esistenti.
Il secondo capitolo dell'elaborato mostra una panoramica del disegno dell'architettura del sistema realizzato, mettendo in luce i principali componenti del sistema. A tal fine è presente un'ampia e dettagliata descrizione di ogni singolo componente del sistema specificando i vantaggi ottenuti dal proprio utilizzo.
Il terzo capitolo descrive come è stata fatta l'implementazione del sistema, mostrando porzioni di codice e screen dell'applicazione che ci aiutano nella comprensione di come funziona il sistema.
Infine nell'ultimo capitolo vengono trattate le considerazioni finali del lavoro svolto soffermandosi anche sui possibili sviluppi.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Capitolo 1

Bitcoin: Fonte di bigdata

Nel lontano 2009 uno pseudonimo Satoshi Nakamoto pubblicò il manifesto *"Bitcoin: A Peer-to-Peer Electronic Cash System"*[13] sancendo la nascita di una nuova moneta digitale chiamata Bitcoin, che fa del suo punto di forza la libertà e la sicurezza. Nel corso degli anni il valore della moneta è accresciuto sino a raggiungere picchi di 19290 dollari nel 17 Dicembre 2018[4]. Di pari passo, anche il numero di transazioni giornaliere sulla rete bitcoin ha raggiunto cifre esorbitanti riuscendo a validare 5 transazioni al secondo[3], questa enorme mole di dati è possibile processarla solo con sistemi distribuiti che gestiscono i BigData. Non a caso l'elaborato di tesi, utilizza un sistema distribuito creato ad hoc per i bigdata come Spark ed Hadoop. I quali permettono l'elaborazione parallela e distribuita di grosse quantità di dati.
Bitcoin, oltre all'impatto economico, ha avuto un importante ruolo nel campo della ricerca, donandoci un sistema che gestisce le transazioni tra due parti in un network peer-to-peer, diversamente dai sistemi tradizionali, denominato Blockchain.

1.1 Blockchain

La blockchain come dice il nome è una catena di blocchi che implementano un database aperto e distribuito, atto a memorizzare le transazioni tra due parti in modo sicuro, verificabile e permanente. In altre parole, la blockchain rappresenta il libro contabile (o libro mastro), ossia il registro sul quale sono riportati tutti gli scambi tra le parti. Questo libro mastro è distribuito (Distributed Ledger) replicato e sincronizzato tra tutti i membri di una rete. In questo database vengono registrate le transazioni (come lo scambio di beni o informazioni) tra i partecipanti alla rete.
I dati non sono memorizzati su un solo computer ma su più macchine collegate tra loro attraverso una rete peer-to-peer [1.1] sottoforma di blocchi.
Ciascun nodo è autorizzato ad aggiornare e gestire il libro contabile distribuito in modo indipendente, ma sotto il controllo consensuale degli altri nodi. Infatti, gli aggiornamenti non sono più gestiti, come accadeva tradizionalmente, sotto il controllo rigoroso di un'autorità centrale, ma sono invece creati e caricati da ciascun nodo in modo appunto indipendente. In questo modo ogni partecipante è in grado di processare e controllare ogni

transazione ma, nello stesso tempo ogni singola transazione, essendo gestita in autonomia, deve essere verificata, crittografata e approvata dalla maggioranza dei partecipanti alla rete.

Figura 1.1: Rete distribuita di nodi paritari.

Ogni blocco contenuto nella blochchain archivia un insieme di transazioni validate correlate da un Marcatore Temporale (Timestamp) ed un hash (una stringa alfanumerica) che identifica il blocco in modo univoco e che permette il collegamento con il blocco precedente[1.2].

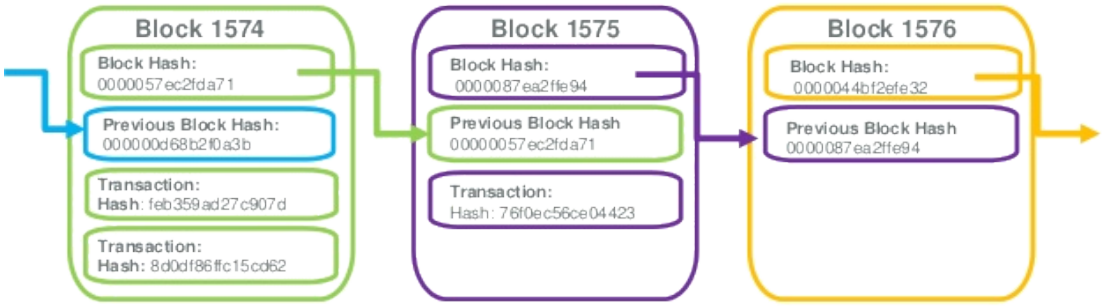


Figura 1.2: Catena di blocchi nella blockchain.

Perché un nuovo blocco di transazioni sia aggiunto alla Blockchain è necessario appunto che sia controllato, validato e crittografato. Solo con questo passaggio può poi diventare attivo ed essere aggiunto alla Blockchain. Per effettuare questo passaggio è necessario che ogni volta che viene composto un blocco venga risolto un complesso problema matematico che richiede un cospicuo impegno anche in termini di potenza e di capacità elaborativa. Questa operazione viene definita come “Mining” ed è svolta dai “Miner”. La risoluzione del problema è un processo irreversibile per cui il blocco valido aggiunto alla

catena diviene immutabile.
In conclusione, questa tecnologia può essere utilizzata in tutti gli ambiti in cui è necessaria una relazione tra più persone o gruppi. Ad esempio può garantire il corretto scambio di titoli e azioni o sostituire un atto notarile, perché ogni transazione viene sorvegliata da una rete di nodi che ne garantiscono la correttezza senza l'ausilio di intermediari.
Bitcoin sfrutta questa tecnologia per gestire le transazioni finanziarie in modo sicuro e del tutto autonomo.

1.2 Bitcoin

Bitcoin (codice: BTC o XBT) è una criptovaluta e un sistema di pagamento mondiale creato nel 2009 da un anonimo inventore, noto con lo pseudonimo di Satoshi Nakamoto, che sviluppò un'idea da lui stesso presentata su Internet a fine 2008. Per convenzione se il termine Bitcoin è utilizzato con l'iniziale maiuscola si riferisce alla tecnologia e alla rete, mentre se minuscola (bitcoin) si riferisce alla valuta in sé.[10]
Il progetto Bitcoin è nato con l'intento di risolvere i problemi di fiducia, trasparenza e responsabilità tra due parti nello scambio di denaro per beni e servizi su Internet, senza l'utilizzo di intermediari. Bitcoin, infatti, rappresenta la prima rete di pagamento che utilizza una tecnologia distribuita peer-to-peer per operare senza un'autorità centrale: la gestione delle transazioni e l'emissione di denaro sono effettuati collettivamente dalla rete.
La rete Bitcoin utilizzando la tecnologia blockchain, è una catena di blocchi concatenati tra di loro, infatti ogni blocco contiene il riferimento al hash del blocco precedente. Questo sistema è immutabile perchè se si volesse modificare un blocco si dovrebbero modificare tutti i blocchi successivi ad esso, per fare ciò si dovrebbe creare una catena di blocchi che è più lunga di quella esistente. Ovviamente, la distribuzione e la natura delle tempistiche del processo rendono praticamente impossibile che ciò accada.
Inotre, utilizzando un sistema distribuito, permette di tenere traccia di tutti i trasferimenti in modo da evitare il problema del double spending (spendere due volte). Tutti gli utenti sono a conoscenza di ciò che accade e pertanto non c'è bisogno di una autorità centrale che gestisca le transazioni.
In aggiunta, Bitcoin consente il possesso e il trasferimento anonimo delle monete, ed infatti i dati necessari per usufruire dei propri bitcoin sono salvati in uno o più personal computer sotto forma di digital wallet (portafoglio). I bitcoin sono trasferiti tramite Internet a chiunque disponga di un indirizzo bitcoin. La struttura peer-to-peer della rete e l'assenza di un ente centrale rende impossibile a qualsiasi autorità il blocco dei trasferimenti, il sequestro dei bitcoin senza il possesso delle relative chiavi o la svalutazione dovuta all'immissione di nuova moneta. Per questo motivo i bitcoin sono la moneta più utilizzata nel Deep Web.

Ricapitolando, le principali caratteristiche su cui si basa questo sistema di enorme successo sono le seguenti:

- **Nessuna autorità centrale**: non dipende da nessuna terza parte privata o ente governativo, e, pertanto, il valore dei BTC è liberamente contrattato sul mercato.

Si tratta quindi di un sistema decentralizzato;

- **Irreversibile e non falsificabile**: una volta che una transazione è stata effettuata ed è inclusa nella blockchain, non può più essere annullata, nemmeno dal mittente;
- **Anonimo**: chiunque può scaricare il software ed iniziare ad effettuare transazioni, senza registrarsi, senza comunicare dati personali e senza svelare la propria identità.
- **Sistema distribuito su rete Peer-to-peer (rete paritaria o paritetica)**: qualsiasi nodo è in grado di avviare e completare una transazione in modo autonomo.
- **Inflazione determinata a priori**: L'emissione di nuovi BTC è determinata dall'algoritmo stesso del programma, e non può essere modificata.

Per capire meglio questi punti, verrà illustrata in modo dettagliato una transazione tra due utenti Alice e Bob che vogliono scambiarsi bitcoin.

Figura 1.3: Logo Bitcoin.

1.2.1 Come avviene una transazione

Un esepio pratico di scambio di moneta aiuterà nel capire come funziona il sistema. I nostri attori saranno Alice (pagante) e Bob (ricevente).

I passi da eseguire sono:

1. Bob comanda alla sua applicazione (wallet) su PC o smartphone di creare un indirizzo. Il software restituisce una sequenza alfanumerica da 26 a 35 caratteri. Questo è un esempio: 12gXGyXWkvyDAjVKZHyGGstVYyXJ6ZjgqV
2. Bob copia l'indirizzo e lo mostra al mittente tramite qualunque mezzo: via mail, messaggio, QR code, pubblicato su un sito internet, scritto su carta, dettato al telefono etc.
3. Alice inserisce nel suo software l'indirizzo di Bob e la quantità di Bitcoin da inviare, inoltre specifica l'ammontare della commissione da pagare al minatore (detto "miner") per convalidare la transazione. Ad oggi i più comuni software stabiliscono automaticamente una commisione fissa a 0,00001 bitcoin.
4. Bob vede immediatamente sul suo software la transazione avvenuta, ma prima di considerare il pagamento effettuato attende che la transazione sia inserita nella blockchain.
5. In un massimo di 10 minuti circa la transazione viene inserita in un blocco della blockchain da parte del miner, che ha convenienza a inserirla perché in questo modo ottiene la commissione pagata da Alice.

6. Bob può sentirsi sicuro che la transazione è confermata con l'aumentare di blocchi che vengono aggiunti alla blockchain conseguentemente a quello che contiene la transazione.

7. Bob avrà nel proprio portafoglio i bitcoin inviati da Alice per spenderli in una nuova transazione.

Figura 1.4: Come avviene una transazione Bitcoin.

1.2.2 Come avviene una transazione dal punto di vista tecnico

La stessa transazione affrontata con un occhio più tecnico, soffermando l'attenzione sulla parte di creazione delle chiavi e della conferma ed aggiunta del blocco.

I passi da eseguire sono:

1. Tramite il software che si occupa del wallet, Bob genera in modo del tutto casuale una chiave privata, che viene salvata sul suo computer.

2. La chiave privata viene convertita in una chiave pubblica tramite un procedimento matematico. Comunemente è il software di Bob a generare automaticamente la chiavi quando Bob chiede all'applicazione di creare un indirizzo da comunicare al mittente. In realtà Bob potrebbe utilizzare una chiave privata facilmente ricordabile, come "sum qui sum", e ricavare Public key e indirizzo da questa. Il procedimento matematico si basa su un algoritmo chiamato "Elliptic Curve Digital Signature Algorithm"[7] che utilizza la curva ellittica.
   È teoricamente possibile ma statisticamente impraticabile scoprire la chiave privata partendo da quella pubblica: poiché il procedimento matematico applicato è unidirezionale, il processo inverso per indovinare la chiave privata richiederebbe una quantità di tentativi e una potenza di calcolo talmente enorme da essere al di là di ogni possibilità

3. La chiave pubblica viene a sua volta crittografata e accorciata tramite un hash. Possiamo chiamare la nuova chiave pubblica Public Key Hash. La chiave pubblica originaria invece è detta Full Public Key.

4. L'hash della chiave pubblica viene convertito in una riga di massimo 35 caratteri (per comodità pratica), che costituisce l'indirizzo del portafoglio di Bob. Per esempio: 12gXGyXWkvyDAjVKZHyGGstVYyXJ6ZjgqV.

5. Bob spedisce l'indirizzo ad Alice.

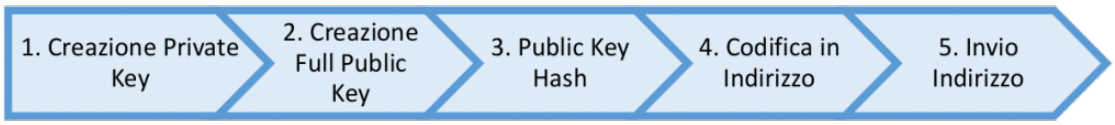


Figura 1.5: Azioni del ricevente.

6. Il software di Alice decodifica immediatamente l'indirizzo in una normale Public Key Hash
7. Alice crea la transazione. Si può pensare la transazione come un codice che contiene diverse informazioni, ciascuna rappresentabile come una stringa composta da molti caratteri:
   - l'input: uno o più output di una transazione precedente fatta nei confronti di Alice, da cui ella attinge i bitcoin che «spedisce» nel nuovo output
   - l'output: la quantità di bitcoin spediti. Possono esserci più output per ogni transazione, ciascuno identificato con un ID specifico
   - l'istruzione per la firma (la "signature script"): ovvero le le istruzioni che Bob dovrà fornire per convalidare la transazione, dimostrando di essere il possessore del nuovo output. È proprio per la creazione dello script che il software di Alice ha bisogno del Public Key Hash fornito da Bob. Le informazioni necessarie per validare la firma sono due, entrambe già in possesso di Bob: la full Public Key e la Private Key, che dovranno combaciare col Public Key Hash specificato da Alice nello script.

   Queste informazioni vengono processate insieme nella creazione di un unico hash chiamato txid (transaction identifier)

Figura 1.6: Prima parte. Azioni del pagante.

8. Tutti i bitcoin che Alice ha a disposizione su un particolare input vengono "spediti" nella transazione. Infatti nella transazione è coinvolta sempre l'intera quantità di bitcoin presenti nell'input anche se Bob ne ha richiesti molti meno. Se Alice dispone di un input di 100 bitcoin e ne trasferisce 20 a Bob, l'input è sempre trasferito nella sua interezza di 100 bitcoin. In questo caso avrà due output diversi, uno di 80 bitcoin (al lordo della commissione per il miner) che tornano al portafoglio di Alice (il change output), l'altro di 20 bitcoin che vanno all'indirizzo di Bob. L'unico caso di transazione che abbia un solo input e un solo output è quello in cui l'input corrisponde esattamente all'ammontare richiesto da chi riceve i bitcoin. Spesso le

transazioni hanno più output, e quindi i bitcoin trasferiti vanno ad indirizzi con diverse chiavi pubbliche e private.

9. Alice trasmette via internet al software di tutti gli altri nodi tutte le informazioni relative alla transazione. I nodi sono rappresentati da tutti coloro che hanno il software Bitcoin Core sui propri pc/dispositivi o numerosi altri software che permettono di "collegarsi" al network Bitcoin.

Figura 1.7: Seconda parte. Azioni del pagante.

10. I minatori inseriscono le transazioni ancora non confermate nella blockchain.
Per inserire le transazioni all'interno della blockchain il miner deve creare un nuovo blocco, processo che richiede una quantità di calcolo molto elevata e dunque una spesa in energia elettrica e strumenti. Un miner ha interesse a inserire quante più transazioni nel blocco che vuole creare poiché guadagnerà tutte le commissioni pagate su ciascuna transazione. Se una transazione non include alcuna commissione, il miner non ha alcun interesse economico nell'inserirla nel blocco.

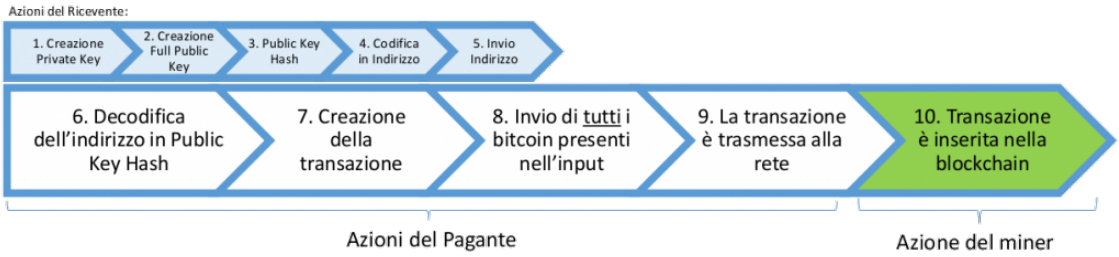


Figura 1.8: Azione dei miner.

Per inserire le transazioni nel blocco, i minatori partono dagli id delle transazioni (txid), ciascuno dei quali rappresenta l'hash di tutte le informazioni inerenti una singola transazione (passo 7).
I txid sono accoppiati due a due, creando un hash per ogni coppia di transazioni. Ogni hash viene poi accoppiato con un altro hash, creando un hash figlio dei due hash precedenti, e così via finché non si arriva a un unico hash. In caso di numeri dispari un hash viene processato con una sua copia identica. Questo procedimento può essere rappresentato come un albero, il cosiddetto Merkle tree [1.9], dove le foglie sono le transazioni txid, i rami (biforcuti) gli hash intermedi e la radice l'hash finale, prodotto di tutti gli altri hash: la Merkle root. L'hash finale è come l'ultimo di una stirpe e porta con sé il "DNA" di tutti gli hash precedenti[8].
Grazie alla struttura del Merkle tree, non è necessario conoscere tutte le transazioni

incluse in un blocco per verificare che una singola transazione ne faccia parte, è invece sufficiente seguire un particolare ramo che collega una foglia (una transazione) alla merkle root.

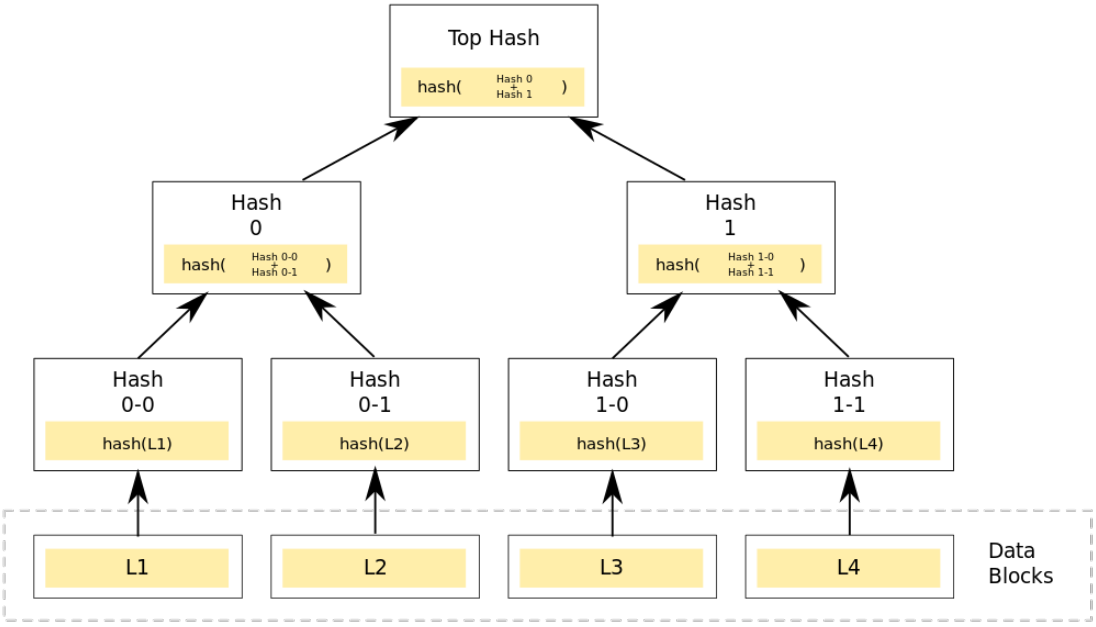


Figura 1.9: Merkle tree.

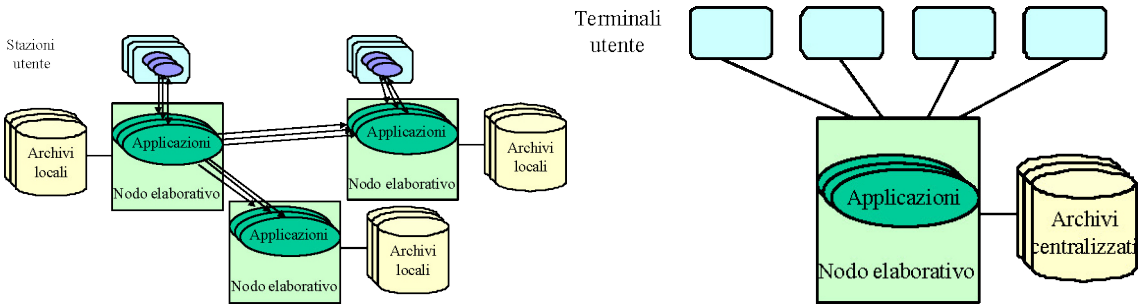
11. Bob ora vuole spendere i suoi nuovi bitcoin in una nuova transazione, il destinatario è Charlie. Bob segue la stessa procedura che ha seguito Alice, creando una transazione specificando output, signature script (per cui gli serve il public key hash di Charlie), timestamp e versione del software.
    Bob però deve dimostrare di essere il possessore dei bitcoin che invia a Charlie, ovvero i bitcoin presenti nell'input della transazione. Tale input è anche l'output della transazione fra Alice e Bob. Quest'ultimo per dimostrare di possedere l'ouput di quella transazione deve porre la sua firma (signature). Bob inserisce quindi la sua Full Public Key, verificando che corrisponde al Public Key Hash dato in precedenza ad Alice, e la sua Private Key, che rappresenta la conferma che Bob solo è la persona che ha originato inizialmente quella Public Key. Infatti seppur sia teoricamente possibile, è per motivi statistici «infattibile» scoprire la Private Key partendo dalla Public Key. Il procedimento di firma è del tutto automatizzato dal software Bitcoin[16].

1.3 Analisi dati su sitemi distribuiti

Il numero di utenti che ogni giorno spende bitcoin è in costante crescita. La tecnologia Bitcoin quindi è costretta a lavorare con volumi di dati sempre crescente. Per questo motivo, le applicazioni che fanno analisi devono adattarsi scegliendo sistemi consoni a queste moli di dati: i Sistemi distribuiti.
Esistono più definizioni (più o meno equivalenti fra loro) di un sistema distribuito, fra cui:

- "Un sistema distribuito è una porzione di software che assicura che un insieme di calcolatori appaiano come un unico sistema coerente agli utenti del sistema stesso" (Maarten van Steen, 2016).[1]
- "Un sistema distribuito consiste di un'insieme di calcolatori autonomi, connessi fra loro tramite una rete e un middleware di distribuzione, che permette ai computer di coordinare le loro attività e di condividere le risorse del sistema, in modo che gli utenti percepiscano il sistema come un unico servizio integrato di calcolo" (Wolfgang Emmerich, 1997).[5]
- "Un sistema distribuito è un sistema in cui il fallimento di un computer che non sapevi neppure esistere può rendere il tuo computer inutilizzabile" (Leslie Lamport, 1987).[12]

Sintetizzando, un sistema distribuito è un insieme di processori indipendenti (con proprie risorse hardware/software) interconnessi da una rete di comunicazione, che cooperano per condividere alcune delle risorse ovunque distribuite ed eseguire algoritmi parallelamente. Questi sistemi riescono ad apparire all'utente come un singolo sistema, permettendo di avere una certa estrazione dall'harware dei singoli nodi.
I sistemi distribuiti [1.10a] si contrappongo ai sistemi centralizzati, nella quale tutto il lavoro è eseguito da un solo calcolatore [1.10b]. Un esempio di sistema distribuito è la rete Internet stessa, che si estende a livello mondiale comprendendo risorse fisicamente molto distanti tra loro, in cui processi con funzioni diverse e connessi da reti di vario tipo si scambiano messaggi informativi basati su disparati protocolli di comunicazione.



(a) Sistema distribuito (b) Sistema centralizzato

Figura 1.10: Differenza tra sistema distribuito e centralizzato.

L'applicazione creata è eseguita su un sistema distribuito questo perché abbiamo bisogno di tempi di risposta bassi ed alta affidabilità dei dati. Per questo motivo il codice viene partizionato in piccoli problemi ed eseguiti dai nodi del nostro cluster. Il processo di partizione dei dati viene automatizzato dal framework Spark [2.3.2] che si occupa della gestione, invio e recupero dei dati tra i vari nodi.

1.3.1 Caratteristiche di un sistema distribuito

Un sistema distribuito si definisce tale se rispetta alcune caratteristiche come:

- **Remoto**: le componenti di un sistema distribuito devono poter essere trattate allo stesso modo sia che siano in locale che in remoto.
- **Concorrenza**: è possibile eseguire contemporaneamente due o più istruzioni su macchine differenti.
- **Malfunzionamenti parziali**: Ogni componente del sistema può smettere di funzionare correttamente indipendentemente dalle altre componenti del sistema; questo non deve compromettere le funzionalità dell'intero sistema. I fallimenti che possono affliggere i processi possono essere di varia natura.
- **Eterogeneità**: un sistema distribuito è eterogeneo per tecnologia sia hardware che software. Si realizza in tutti i contesti come rete di comunicazione, protocollo di rete, linguaggi di programmazione, applicazioni, etc.
- **Autonomia**: un sistema distribuito non ha un singolo punto dal quale può essere controllato, coordinato e gestito. La collaborazione va ottenuta inviando messaggi tra le varie componenti del sistema e gestita tramite politiche di condivisione e di accesso che devono essere rigorosamente seguite.
- **Evoluzione**: un sistema distribuito può cambiare sostanzialmente durante la sua vita, sia perché cambia l'ambiente sia perché cambia la tecnologia utilizzata. L'obiettivo è quello di assecondare questi cambiamenti senza costi eccessivi.[11]

1.3.2 Vantaggi e Svantaggi

I sistemi distribuiti hanno dei pro e dei contro che sono da prendere in considerazione quando si vogliono adottare soluzioni di questo tipo.
I vantaggi nell'utilizzo dei sistemi sono:

- **Connettività e collaborazione**: possibilità di condividere risorse hardware e software (compresi dati e applicazioni)
- **Prestazioni e scalabilità**: la possibilità di aggiungere risorse fornisce la capacità di migliorare le prestazioni e sostenere un carico che aumenta (scalabilità orizzontale)
- **Tolleranza ai guasti**: grazie alla possibilità di replicare risorse e dati

- **Apertura**: l'uso di protocolli standard aperti favorisce l'interoperabilità di hardware e software di fornitori diversi
- **Economicità**: i sistemi distribuiti offrono spesso (ma non sempre) un miglior rapporto prezzo/qualità che i sistemi centralizzati basati su mainframe

Gli svantaggi invece sono:

- **Complessità**: i sistemi distribuiti sono più complessi di quelli centralizzati e quindi risultano più difficili da capire, inoltre, lo sviluppo delle applicazioni deve essere implementato ad hoc.
- **Sicurezza**: l'accessibilità in rete pone problematiche di sicurezza
- **Gestibilità**: è necessario uno sforzo maggiore per la gestione del sistema operativo e delle applicazioni

1.4 Stato d'arte

Capitolo 2

Overview e progettazione di sistema

In questo capitolo viene messo in luce lo scopo dell'elaborato realizzato con particolare attenzione alle scelte progettuali adottate e alle ricerche effettuate per implementare l'argomento trattato, onde presentare un quadro generale completo. Viene, inoltre, descritta l'architettura del software, sottolineando ed approfondendo le sue principali componenti.

2.1 Scopo del progetto

Lo scopo principale del lavoro di tesi è quello di creare un sistema distribuito che permetta la visualizzazione real time, la storicizzazione e l'analisi delle transazioni provenienti dalla blockchain di bitcoin. L'obiettivo, dunque, è quello di riuscire a creare un sistema che gestisca grandi quantità di dati in un ambiente distribuito, garantendo affidabilità e consistenza dei dati anche in caso di guasti.
Il sistema si rivolge ad un pubblico che vuole fare analisi delle transazioni processate dalla rete bitcoin. L'utente infatti, può controllare in real time le ultime transazioni elaborate, monitorare l'intera rete blockchain per risalire a tutta la catena di transazioni, oppure controllare gli hash (indirizzi) che hanno avuto maggior punteggio di PageRank.
Le ultime transazioni elaborate, hanno una serie di informazioni di dettaglio come il blocco di appartenenza, l'hash che ha generato quella transazione, il timestamp ed i destinatari dei bitcoin. Mentre per quanto riguarda il monitoraggiio dell'intera blockchain, l'utente può navigare con l'utilizzo del proprio mouse, in un grafo orientato rappresentante la storia di tutte le transazioni. Inoltre, il fruitore può controllare i nodi con il maggior punteggio di PageRank e localizzarlo all'interno del grafo.

2.2 Architettura del progetto

Le principali scelte di progetto sono state prese coerentemente con lo stato dell'arte e si è tentato di non introdurre nuova complessità al panorama esistente, ricorrendo a tecnologie, protocolli e standard già esistenti ed affermati, senza definirne di nuovi. Quindi, l'architettura dovrà essere sufficientemente generale in modo da poter garantire

nuovi sviluppi ed evoluzioni future e da non comportare l'esclusione a priori di determinate soluzioni e tecnologie.
Il sistema, quindi, si divide logicamente in due moduli:

- **Il Sistema distribuito (back-end)**: E' la parte non visibile all'utente. Si occupa del recupero dei dati dalla blockchain di bitcoin, della storicizzazione e della pubblicazione sui topic di kafka. Interamente scritto in Java, comprende Bitcoind, Spark, Hadoop, Neo4j, Zookeeper.
- **Webapp (front-end)**: E' la parte visibile all'utente finale. Si occupa della rappresentazione grafica delle transazioni. Scritto in principalmente in Javascript, utilizza la potenza di NodeJS per creare l'interfaccia grafica. Integra nel proprio ecosistema Express Handlebars, WebSocket, MaterialCSS e D3js.

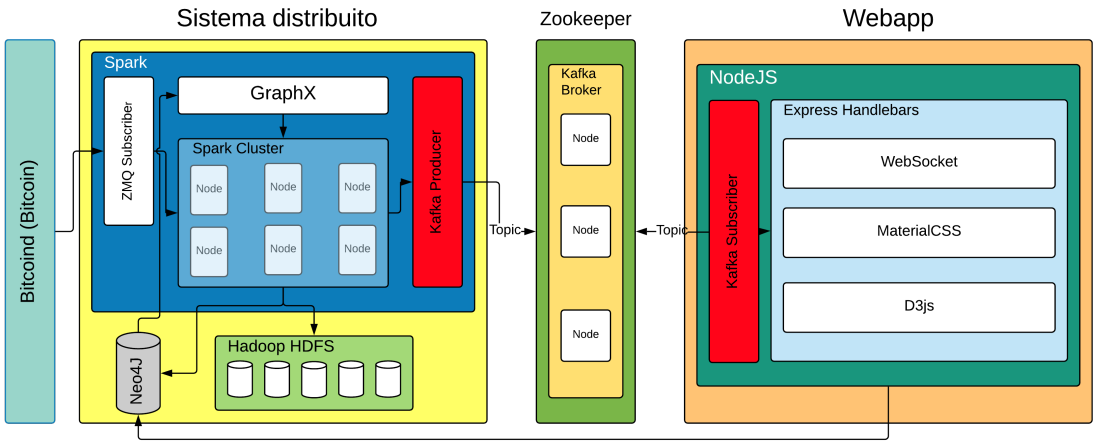


Figura 2.1: Architettura completa.

Per la costruzione di questa infrastruttura, la prima grande sfida, è stata quella di trovare un framework o un tool che permettesse di creare e programmare su di un sistema distribuito, senza complicarci la vita. Facendo ricerche sul web, la tecnologia che più si accostava meglio al mio problema è stata Apache Spark [2.3.2]. Questo strumento riesce a garantire a pieno i vincoli che ci siamo imposti, regalandoci la possibilità di creare un sistema distribuito su vari nodi impostando opportuni file di configurazione.
Risolto il problema infrastrutturale si è proceduto all'analisi dei singoli sottoproblemi. L'applicazione, per testare il carico di lavoro sul sistema, fa uso dei blocchi grezzi provenienti dal software nativo del progetto Bitcoin: Bitcoind [2.3.1]. Bitcoind è un demone che invia blocchi o transazioni (a seconda di come lo si imposta) su di una coda di tipo publisher-subscriber tramite protocollo ZeroMQ [2.3.1.1]. I dati, quindi, sono prelevati da Bitcoind grazie all'implementazione di un connettore creato ad hoc.
Ottenuti i blocchi dalla coda, è nata l'esigenza di conservare i dati ottenuti così da poterli processare ed analizzare. Fortunatamente, Spark offre una nativa collaborazione con il FileSystem distribuito Hadoop [2.3.3], permettendomi di tenerli salvati su una memoria

di massa distribuita.
Oltre ad Hadoop, i dati sono stati immagazzinati in Neo4j [2.3.4]. Un database NoSQL che permette il salvataggio dei dati sottoforma di grafo, cosi da poter gestire facilmente i collegamenti tra le varie transazioni.
L'ultimo step, è stato quello di fare analisi delle transazioni, trovando i nodi con il maggior PageRank [**??**]. Anche in questo caso Spark è venuto in contro grazie al modulo GraphX, contenuto nel framework, il quale contiene algoritmi (come il PageRank) già sviluppati per l'analisi sui grafi.
Una volta che il sistema distribuito è completo, non resta che mostrare i risultati ottenuti. Le scelte nel campo del front-end sono migliaia ma per semplicità ed una forte attitudine ai sistemi real-time si è preferito usare NodeJS [2.4.1]. NodeJS ha dei moduli che permettono l'accesso a Kafka, il tramite tra la parte di back-end e front-end. Quindi, con NodeJS è stato creato un sito web consentendo agli utenti dal proprio browser, di visualizzare lo stato delle transazioni, i valori del PageRank e le transazioni che arrivano in real time.

2.3 Sistema distribuito

Come si accennava il precedenza, per lo sviluppo dell'applicazione è stato necessario ricorrere ad un sistema distribuito. Questo sistema, perno di tutta la trattazione, è invisibile all'utente ma di fondamentale importanza. Infatti, è colui che si occupa del processamento dei dati provenienti da Bitcoind, garantendo che nessun guasto infici sul risultato.
Il sistema elabora su più macchine distribuite per ottimizzare i tempi di risposta. In particolare, utilizza un "cluster Spark" (insieme di macchine) per ridistribuire il carico di lavoro equamente sui diversi nodi. Cosi facendo, ha la possibilità di elaborare grandi quantità di dati in real-time. Inoltre, utilizza la tecnica della replicazione dei dati così che, in caso di rottura di uno dei nodi, si possano recuperare le informazioni perse.
Terminata l'elaborazione dei dati, il cluster invia i risultati ai sistemi dediti al salvataggio e alla pubblicazione. Due, sono quelli dedicati alla storicizzazione permanente dei dati: Neo4j e Hadoop. Mentre per quanto riguarda la pubblicazione, i dati saranno inviati a Kafka (tramite un producer) che li renderà disponibili per essere fruiti. La scelta dei framework Hadoop e Kafka, è data dal fatto che sono entrambi capaci di lavorare in ambienti distribuiti. Diversa invece è la scelta di Neo4j, l'utilizzo di tale sistema, infatti, è dovuto al fatto che si adatta al meglio con l'idea di transazione, salvando i dati come grafi.

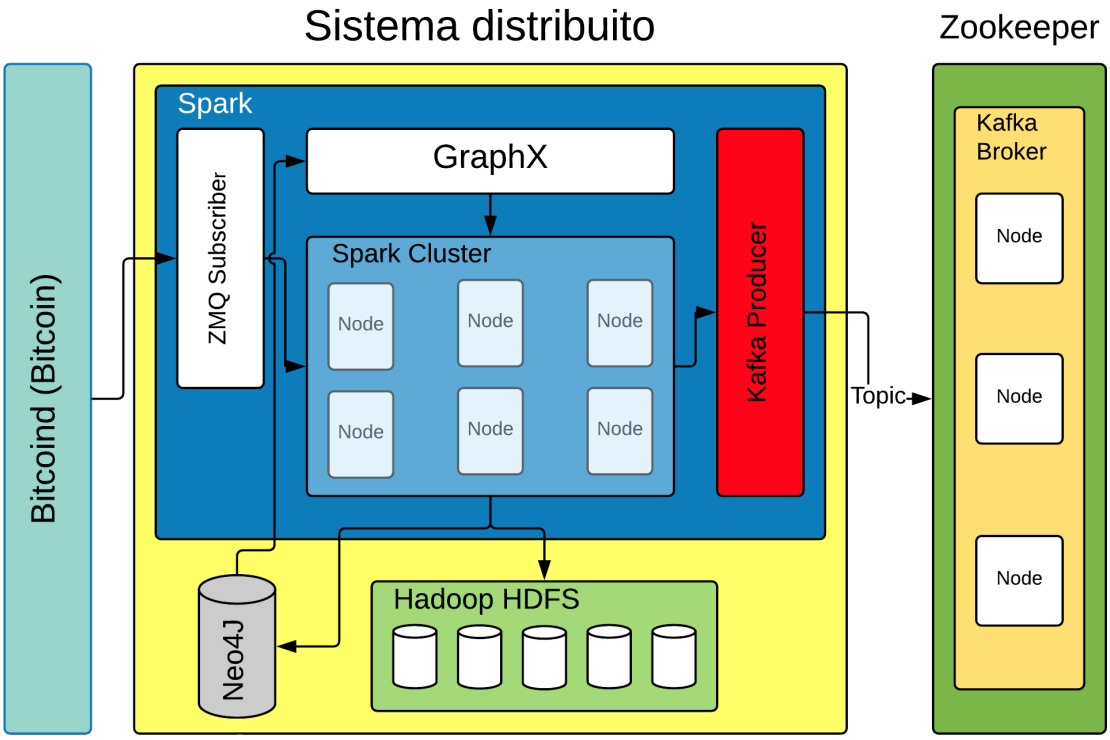


Figura 2.2: Architettura in dettaglio del sistema distribuito.

2.3.1 Bitcoind

Il primo componetene del sistema distribuito ha il compito di fornire i dati da elaborare. Questa funzione è svolta dal demone Bitcoind.
Bitcoind, formalmente, è un software che implementa il protocollo Bitcoin per l'utilizzo delle remote procedure call (RPC). Esso è anche il secondo client Bitcoin nella storia del network [6]. Per sua natura, è eseguito come processo in background quindi l'utente per interagire con esso ha bisogno di farlo tramite una interfaccia da riga di comando chiamata *bitcoin-cli*. Il demone, inoltre, funge da nodo della rete Bitcoin, infatti si sincronizza con la blockchain, verifica le transazioni ed invia blocchi. Esiste una versione anche con interfaccia grafica del demone chiamata *Bitcoin-QT o Bitcoin Core*, ma per lo scopo dell'elaborato si è preferito utilizzare la versione lite per limitare l'utilizzo di risorse.
La versione demone, inoltre, ha il vantaggio di creare una coda ZeroMQ per la comunicazione con applicazioni esterne. Il sistema distribuito, utilizza questa funzione per recuperare i blocchi in formato grezzo (sequenza di byte) ogni qualvolta sono validati dalla blockchain.

2.3.1.1 ZeroMQ

ZeroMQ (anche conosciuto come ØMQ, 0MQ, o zmq) è una libreria di messagistica asincrona ad alte prestazioni, destinata all'uso in applicazioni distribuite o concorrenti. Fornisce code di messaggi, ma a differenza dei middleware orientati ai messaggi, il sistema ZeroMQ può essere eseguito senza un broker di messaggi dedicato [9]. La libreria, inoltre, funziona molto bene grazie al suo modello interno di threading, che può superare le tradizionali applicazioni TCP in termini di throughput utilizzando una tecnica di batching automatico dei messaggi [17]. Il suo modello I/O asincrono offre la possibilità di creare applicazioni multicore scalabili, costruite come attività asincrone di elaborazione dei messaggi. Inoltre, la comunità di sviluppatori, ha creato una quantità enorme di wrapper per la libreria, così da renderla funzionante sulla maggior parte dei sistemi operativi. All'interno del sistema distribuito, infatti, è stato utilizzato il wrapper *jzmq* che permette l'utilizzo della libreria tramite Java API.
ZMQ, può avere diverse modalità di invio di messaggi atomici:

- Request-reply: Connette un insieme di clienti ad un insieme di servizi. Questo è una Remote Procedure Call (RPC).
- Publish-subscribe: Connette un insieme di produttori ad un insieme di consumatori.
- Push-pull (pipeline): Connette i nodi in un pattern fan-out/fan-in che può avere più passaggi e cicli. Distribuisce in maniera parallela i messaggi.
- Exclusive pair: Collega due socket in maniera esclusiva.

Bitcoind utilizza una socket di tipo Publish-subscribe. Il publisher, in questo caso bitcoind, può spedire un messaggio a molti consumers attraverso un canale virtuale chiamato topic. Uno stesso topic può essere condiviso da più subscriber (client). Per ricevere i messaggi, i client devono "sottoscriversi" al topic creato da bitcoind. Ovviamente, questo implica che c'è relazione temporale tra i publisher e i subscibers, nel senso che, un client che si sottoscrive ad un topic può consumare solamente i messaggi pubblicati dopo la sua sottoscrizione. Qualsiasi messaggio spedito sul topic, viene consegnato a tutti i consumer sottoscritti, ciascuno dei quali riceve una copia identica di ciascun messaggio inviato. I messaggi quindi, vengono automaticamente inviati in broadcast ai consumer, senza che questi ne abbiano fatto esplicita richiesta.
Nel caso del sistema distribuito, il client che si sottoscrive al topic *pubrawblock* di bitcoind è il connettore-subscriber implementato all'interno di Spark.

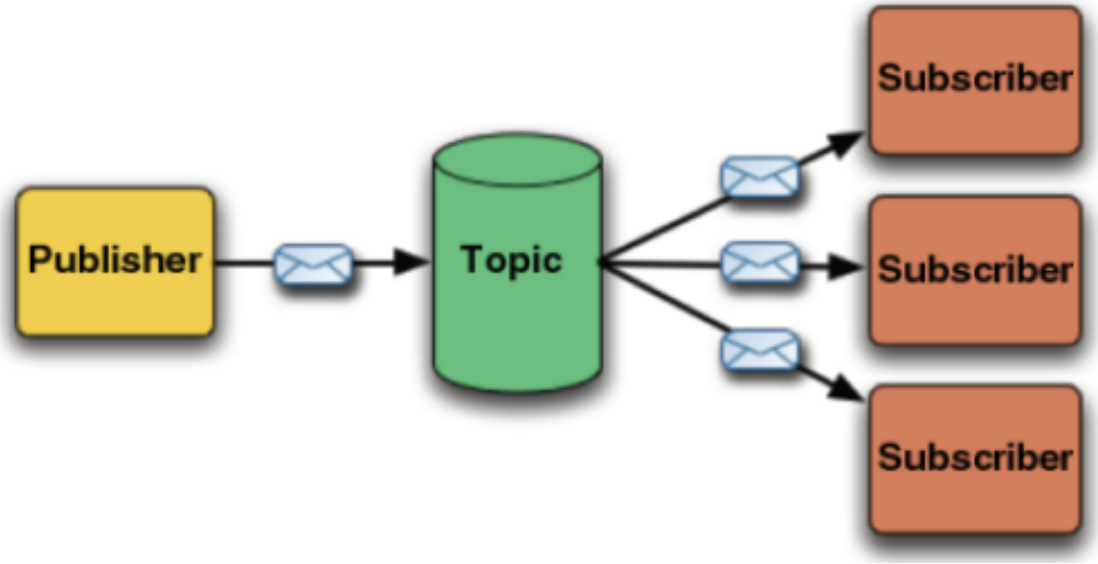


Figura 2.3: Messaggio inviato con la modalità Publish-Subscribe.

2.3.2 Apache Spark

Apache Spark è un framework di calcolo del cluster per l'elaborazione di dati su larga scala, progettato e implementato nel 2010 da un gruppo di ricercatori dell'Università di Berkeley a San Francisco [15]. Questo progetto nasce dall'esigenza di migliorare le prestazioni dei sistemi distribuiti "MapReduce". Per questo si sviluppa il concetto di Resilient Distributed Dataset (RDD), che è la teoria alla base del sistema Spark. Un RDD rappresenta un set di dati che è suddiviso in partizioni (Una tabella chiave-valore suddivisa in tante sotto-tabelle o un file diviso in tanti segmenti). Un RDD ha la proprietà di essere immutabile, cioè una volta creato non può essere cambiato se non creandone un altro. La creazione di un RDD avviene a partire dai dati su disco (presi da HDFS) o da altre fonti di dati. Una volta creato, un RDD può restare in memoria oppure può essere materializzato su disco. Ogni RDD è descritto da un set completo di metadati che consentono la ricostruzione di una delle sue partizioni in caso di fault. Spark nasce come un sistema per creare e gestire job di analisi basati su trasformazioni di RDD. Dato che gli RDD nascono e vivono in memoria, l'esecuzione di lavori iterativi, o che trasformano più volte un set di dati, sono immensamente più rapide di una sequenza di MapReduce; questo perchè il disco non viene mai (o quasi mai) impiegato nell'elaborazione.
Come detto in precedenza, Spark non utilizza MapReduce come motore di esecuzione; invece, utilizza il proprio runtime distribuito (DAG) per l'esecuzione di jobs su un cluster. Quando viene invocata un'azione su un RDD, viene creato un "job". Un Directed Acyclic Graph o DAG è un grafo aciclico in cui ogni nodo è una partizione di RDD e ogni vertice è una trasformazione. A differenza di MapReduce, il motore DAG di Spark può processare pipeline arbitrarie di operatori e tradurle in un unico "job" per l'utente.
Spark, infine, sta dimostrando di essere una buona piattaforma su cui costruire strumenti di analisi, infatti ha moduli per il Machine learning (MLlib), Elaborazione grafica

(GraphX), Elaborazione di stream (Spark Streaming) ed SQL (Spark SQL) [15].

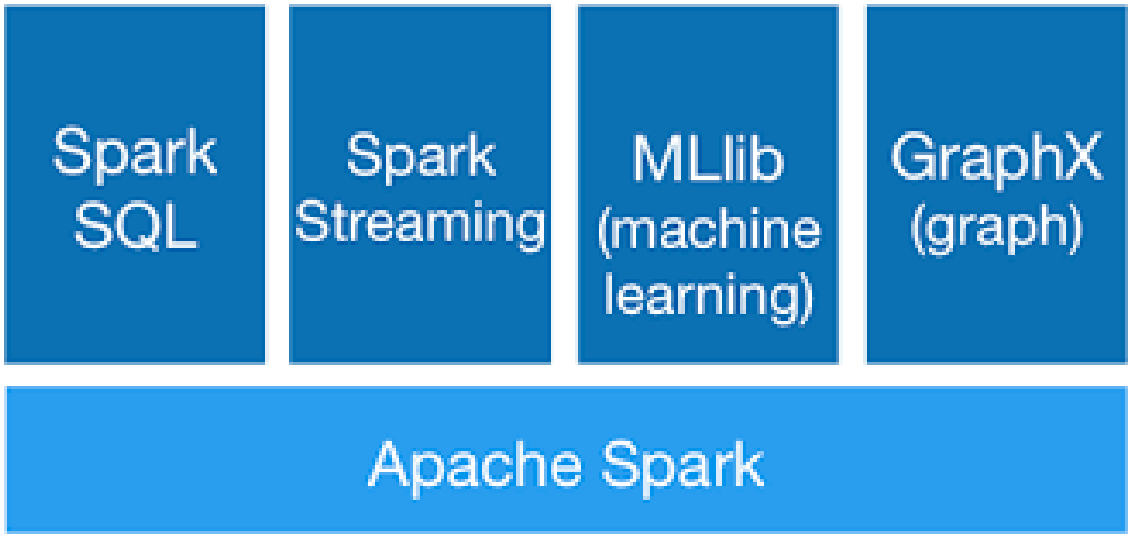


Figura 2.4: Infrastruttura Spark.

Nel sistema distribuito, poichè c'è l'esigenza di recuperare i dati in real time dalla coda di bitcoind, viene utilizzato Spark Streaming. Questo modulo è un'estensione dell'API Spark di base che consente l'elaborazione streaming, scalabile, ad alto throughput e con tolleranza agli errori dei flussi di dati in tempo reale. I dati, che possono provenire da diverse fonti, sono elaborati utilizzando algoritmi complessi espressi con funzioni di alto livello come *map*, *reduce*, *join* e *window*. I dati processati, infine possono essere inviati a filesystem (Hadoop) o database (Neo4j) per il salvataggio oppure ad altri moduli di Spark, dediti all'analisi, tipo machine learning (MLlib) o graph processing (GraphX).
Internamente, Spark Streaming riceve streams di dati di input e li divide in batch, che vengono quindi elaborati dal motore Spark per generare il flusso finale di risultati. Per consentire il facile utilizzo di questi dati, fornisce un'astrazione di alto livello chiamata stream discretizzato o DStream, che rappresenta un flusso continuo di dati. E' possibile creare Dstreams da flussi di dati di input da sorgenti come Kafka, Flume e Kinesis o applicando operazioni di alto livello su altri Dstreams [14]. Internamente, un Dstream è rappresentato come una sequenza di RDD sulla quale possono essere effettuate le operazioni descritte in precedenza.

Figura 2.5: Come vengono gestiti i dati in Spark Streaming.

Nel sistema distribuito, per fare analisi dei dati provenienti dall'elaborazione di Spark, è stato scelto il modulo interno Graphx.

2.3.3 Hadoop HDFS

2.3.4 Neo4j

2.3.5 Zookeeper

2.3.5.1 Kafka

2.4 WebApp

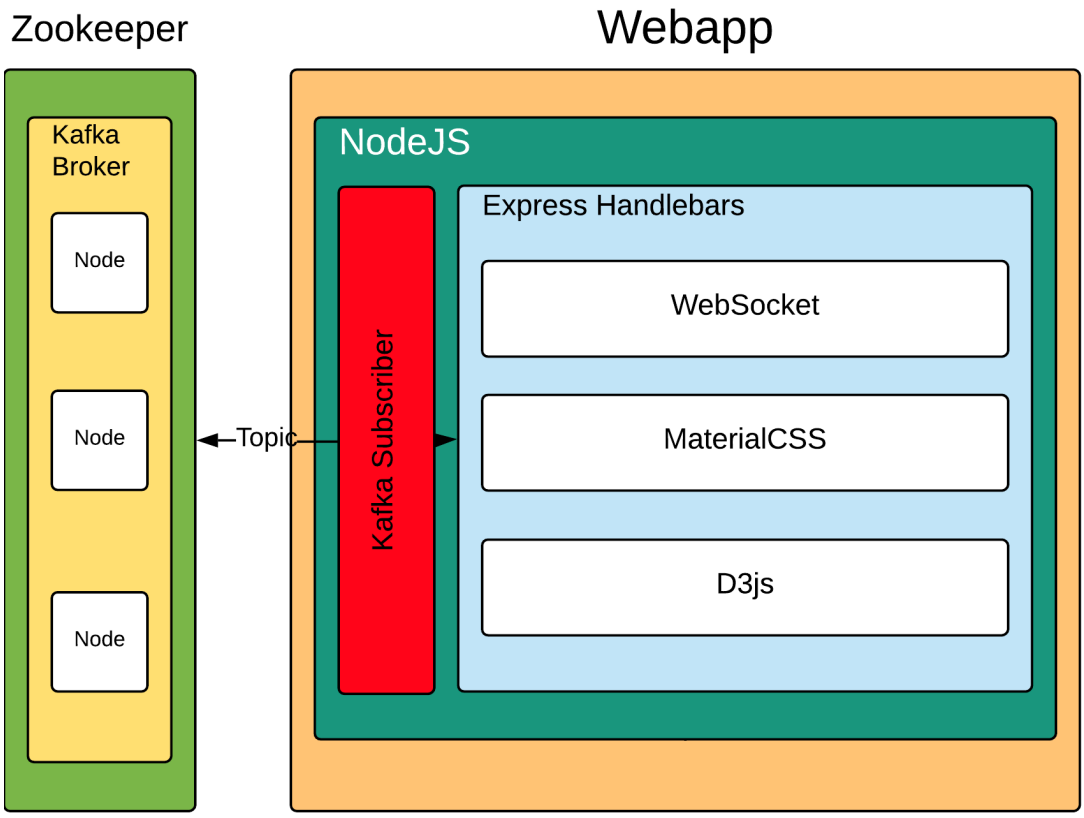


Figura 2.6: Architettura in dettaglio della webapp.

2.4.1 NodeJS

2.4.1.1 Express Handlebars

2.4.1.2 WebSocket

2.4.1.3 MaterialCSS

2.4.1.4 D3js

Capitolo 3

Implementazione

3.1 Diagramma delle classi

3.2 Codice

3.3 Github

Capitolo 4

Conclusioni e sviluppi futuri

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut

porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Bibliografia

[1] Maarten van Steen e Andrew S. Tanenbaum. «A brief introduction to distributed systems». In: (2008). URL: https://www.cs.vu.nl/~ast/Publications/Papers/computing-2016.pdf.

[2] Apache. «Apache Software Foundation. Official Website Apache Spark». In: *URL http://spark.apache.org* ().

[3] Blockchain.com. *Chart bitcoin transactions*. URL: https://www.blockchain.com/it/charts/n-transactions?timespan=all (visitato il 2018).

[4] Blockchain.com. *Chart value of Bitcoin*. URL: https://www.blockchain.com/charts/market-price?timespan=all (visitato il 2018).

[5] Wolfgang Emmerich. «Distributed System Principled». In: (1997). URL: http://www0.cs.ucl.ac.uk/staff/ucacwxe/lectures/ds98-99/dsee3.pdf.

[6] en.bitcoinwiki.org. *Bitcoind*. URL: https://en.bitcoinwiki.org/wiki/Bitcoind (visitato il 2018).

[7] en.bitcoinwiki.org. *ECDSA*. URL: https://en.bitcoinwiki.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm (visitato il 2018).

[8] en.wikipedia.org. *Merkle Tree*. URL: https://en.wikipedia.org/wiki/Merkle_tree (visitato il 2018).

[9] en.wikipedia.org. *ZeroMQ*. URL: https://en.wikipedia.org/wiki/ZeroMQ (visitato il 2018).

[10] it.wikipedia.org. *Bitcoin*. URL: https://it.wikipedia.org/wiki/Bitcoin (visitato il 2018).

[11] it.wikipedia.org. *Sistema Distribuito*. URL: https://it.wikipedia.org/wiki/Sistema_distribuito#Caratteristiche (visitato il 2018).

[12] Leslie Lamport. «Distribution». In: (1997). URL: https://www.microsoft.com/en-us/research/uploads/prod/2016/12/Distribution.pdf.

[13] Satoshi Nakamoto. «Bitcoin: A Peer-to-Peer Electronic Cash System». In: (2009). URL: https://bitcoin.org/bitcoin.pdf.

[14] spark.apache.org. *Spark Streaming*. URL: https://spark.apache.org/docs/latest/streaming-programming-guide.html (visitato il 2018).

[15] Tom White. *Hadoop: The definitive guide*. "O'Reilly Media, Inc.", 2012.

[16] www.albertodeluigi.com. *Come avviene una transazione bitcoin*. URL: http://www.albertodeluigi.com/index/bitcoin/transazione-bitcoin#panel-108-0-0-1 (visitato il 2018).

[17] zeromq.org. *How come ØMQ has higher throughput than TCP although it's built on top of TCP?* URL: http://zeromq.org/area:faq#toc6 (visitato il 2018).