UNIVERSITÀ DEGLI STUDI DI PERUGIA
Dipartimento di Matematica e Informatica

CORSO DI LAUREA TRIENNALE IN INFORAMTICA

Tesi di Laurea

Blockchain: gestione di Big Data

Laureando
Francesco Moca

Relatore/i
Prof. Stefano Bistarelli

Anno Accademico 2014 / 2015

Alla mia famiglia e ai miei amici . . .

Introduzione

Bitcoin è un'innovativa moneta elettronica nata nel 2009 che ha come caratteristica principale l'indipendenza da governi e banche. La tecnologia che permette ciò è chiamata *blockchain*, un registro pubblico e condiviso che contiene lo storico di tutte le transazioni avvenute dall'invenzione della valuta da parte del sedicente Satoshi Nakamoto.

Questo registro può essere visto come un immenso database distribuito in cui ogni giorno mediante una rete peer to peer vengono aggiunte, all'interno di strutture chiamate blocchi, oltre 200 mila transazioni.

Bitcoin è allo stesso tempo trasparente e anonimo. Infatti nonostante le transazioni siano pubbliche, la privacy dei possessori è garantita da un identificativo crittografico che nasconde la loro identità.[1]

L'anonimato garantito dal protocollo Bitcoin, pur avendo certamente utilizzi legali, spesso viene usato come alternativa al denaro per l'acquisto di merci o servizi illegali. Quindi studiando la blockchain è possibile osservare il flusso dei bitcoin in un certo periodo, scoprire transazioni interessanti e pattern.

Ad esempio si può usare la blockchain per seguire una transazione "macchiata" ovvero una transazione dove l'identità del possessore di bitcoin è conosciuta. Per evitare ciò sono nati servizi di riciclaggio

che mischiano i Bitcoin per ottenere Bitcoin "puliti", chiamati *Mixing Services*.[1] Studiando i pattern delle transazioni può essere possibile ricostruire il percorso che fanno questi bitcoin e capirne l'origine e la destinazione.

Tuttavia nel corso degli anni la blockchain ha raggiunto numeri notevoli, al momento contiene oltre 115 milioni di transazioni che hanno coinvolto più di 135 milioni di indirizzi bitcoin univoci![2]

Lo studio della blockchain è quindi considerabile a tutti gli effetti un'analisi di Big Data, ovvero un insieme di dati molto grandi che richiedono analisi particolari. Negli ultimi anni i classici database relazionali si sono rilevati inadatti a gestire queste grosse quantità di dati, ed è quindi nata una nuova tipologia di base di dati detta non relazionale (o NoSQL), più veloce e più scalabile.

Visto che il nostro progetto si basa sull'analizzare lo scambio di valuta useremo un particolare sottogruppo di database NoSQL: i graph database.

La tesi è suddivisa in cinque capitoli. Nel primo verrà spiegato in dettaglio il funzionamento della moneta Bitcoin, cosa è una transazione tra più indirizzi bitcoin e soprattutto la struttura dati Blockchain.

Nel secondo capitolo si parlerà dei big data e quali sono le principali caratteristiche del processo di analisi.

Il terzo si focalizzerà nelle nuove tipologie di database NoSQL e soprattutto GraphDB e quali sono le principali differenze e vantaggi rispetto alle basi di dati relazionali.

Nel quarto capitolo si analizzeranno gli strumenti e le tecnologie scelte

[1] https://en.bitcoin.it/wiki/Mixing_service

[2] http://blockr.io/trivia/blockchain (ultima consultazione: 23/03/2015)

per la realizzazione del nostro applicativo. Infine nell'ultimo capitolo si descriverà la nostra applicazione, in particolare sulla parte backend a partire dall'esportazione dei dati della blockchain nel nostro database fino alla creazione di API per rendere di facile accesso alcune query di interesse.

Indice

Introduzione **i**

1 Bitcoin **1**

1.1 Panoramica sui Bitcoin 3

1.2 Chiavi, indirizzi, wallet 4

1.2.1 Come creare un portafoglio 5

1.3 Transazioni 9

1.3.1 Un esempio di transazione 13

1.3.2 Il linguaggio Script Bitcoin: Casi Particolari 17

1.4 Blockchain 20

1.4.1 Struttura della blockchain 22

1.4.2 Proof of Work: Mining di Bitcoin 25

1.5 Riepilogo e Conclusioni 35

2 Big Data **38**

2.1 Caratteristiche 39

2.2 Data analysis 40

2.2.1 Analisi sulla esperienza utente 42

2.3 Data Management e Conclusioni 43

3 Database NoSQL e GraphDB **45**

3.1 Il passato: Database Relazionali 45

3.1.1 Caratteristiche 46

3.2 Differenze e vantaggi dei database NoSQL 47

3.3 Tipi ed esempi di Database NoSQL 48

3.4 Grafi e GraphDB 50

3.4.1 Cos'è un grafo. 50

3.4.2 Database a grafo 51

3.5 Conclusioni 52

4 Strumenti usati **53**

4.1 Parte Back end 53

4.1.1 OrientDB 53

4.1.2 PHP 55

4.1.3 JSON 56

4.1.4 Bitcore 56

4.2 Node.js 58

4.3 Parte Front End 59

4.3.1 HTML 59

4.3.2 CSS 59

4.3.3 Javascript 60

4.3.4 Ajax 60

5 Web Application **62**

5.1 Descrizione del progetto 62

5.2 Struttura Dati Scelta 62

5.2.1 Transaction 63

5.2.2 Address 64

5.2.3 ValueTx 65
5.2.4 InvalidOutputScript 66
5.2.5 Riepilogo 66
5.3 Inserimento Dati 66
5.3.1 Script GetBlock 66
5.4 Parte Back end 68
5.4.1 Riepilogo Database 68
5.4.2 Scarica Blocchi 69
5.4.3 Aggiungi nomi ad indirizzi 70
5.4.4 Svuota database 71
5.5 Query interessanti 72
5.5.1 Cosa sono le isole 72
5.5.2 Elenco Isole 74
5.5.3 Visita di una singola Isola 76
5.5.4 Transazioni di un indirizzo 79
5.6 Riepilogo 81

Conclusioni **83**

Bibliografia **85**

Capitolo 1

Bitcoin

Il primo novembre del 2008 in una mailing list riguardante la crittografia qualcuno, con lo pseudonimo Satoshi Nakamoto, fece un annuncio che avrebbe fatto la storia: [1]

> I've been working on a new electronic cash system that's fully peer-to-peer, with no trusted third party.
>
> The paper is available at: http://www.bitcoin.org/bitcoin.pdf [2]

In questo documento Nakamoto presenta un sistema di contanti elettronico completamente decentralizzato che non dipende da nessuna autorità centrale né per l'emissione di valuta né per l'invio di denaro.
La principale novità è l'introduzione di un sistema distribuito *proof of work* che ogni circa 10 minuti tramite una sorta di "competizione" globale raggiunge il consenso sulla validità delle transazioni.
Questo elegantemente risolve il problema del *double spending*, ovvero la possibilità di spendere più volte la stessa quantità di denaro, che prima del 2008 era risolvibile solamente mediante un'entità centrale fidata.

[1] https://www.mail-archive.com/cryptography@metzdowd.com/msg09959.html

Ad esempio PayPal, sicuramente uno dei più popolari sistemi di pagamento online, per verificare la validità di un invio di denaro possiede un registro contabile centralizzato nei loro sistemi che tiene traccia delle transazioni eseguite e dei saldi di ogni conto. [2]
Questo registro invece nella tecnologia Bitcoin è distribuito e copiato in tutti in tutti i nodi della rete e ogni 10 minuti si aggiorna con le nuove transazioni eseguite.
Ogni aggiornamento viene detto blocco ed è strettamente legato al precedente formando la cosidetta *blockchain*, ossia lo storico completo delle transazioni avvenute dal 2009 ad oggi nella rete Bitcoin.

Infatti nel gennaio del 2009 questa rete distribuita è partita basandosi inizialmente sulla prima implementazione del client pubblicata da Nakamoto e poi, grazie alla natura *open source* del progetto, tramite il contributo di tanti altri sviluppatori volontari.
Nel 2011 Satoshi Nakamoto si è ritirato dal pubblico scomparendo nello stesso modo misterioso in cui si era presentato, tuttavia la crescita del Bitcoin non si è certamente arrestata, ad oggi la capitalizzazione della moneta è stimata oltre i 6 miliardi di dollari. [2]
Questa rivoluzionaria invenzione ha dunque creato tanto fermento sia nel campo informatico che in quello economico, ed è certamente necessario comprendere appieno la rivoluzione iniziata da Nakamoto per capire ciò che avverrà nei prossimi anni.

[2] https://coinmarketcap.com/ aggiornato al 24/03/2016

1.1 Panoramica sui Bitcoin

Prima di analizzare il protocollo Bitcoin in una maniera più dettagliata, credo sia utile visualizzare graficamente l'intero ecosistema che ruota intorno alla moneta.

In particolare, vediamo come la sua struttura è composta da *utenti* che possiedono dei portafogli digitali o *wallet* contenenti delle chiavi crittografiche, con le quali firmare delle transazioni trasmesse in broadcast sulla rete e, in ultimo, nodi chiamati *miner* che hanno lo scopo di aggiornare, attraverso un processo detto mining, la struttura dati pubblica e distribuita chiamata *blockchain* mantenendo così il sistema funzionante senza la necessità di una terza parte centralizzata fidata.

Figura 1.1: L'ecosistema della tecnologia Bitcoin [3]

1.2 Chiavi, indirizzi, wallet

Una delle sfide che bisogna affrontare per definire una valuta digitale è definire un modo per traferire la proprietà delle monete da un soggetto ad un altro.
Satoshi Nakamoto ha proposto di usare la crittografia asimmetrica, che si basa su una coppia di chiavi, una pubblica, che deve essere distribuita e una privata, appunto personale e segreta.
Ogni utente deve possedere almeno una coppia di queste chiavi che consentono in ogni momento di provare la legittima proprietà di un determinato numero di bitcoin allo scopo di poterli successivamente spendere e trasferire ad un nuovo proprietario. Le chiavi in questione sono solitamente generate e salvate da un software che prende il nome di "portafoglio" digitale o *wallet*.

Praticamente per poter ricevere bitcoin, un utente deve fornire la propria chiave pubblica a colui che li invia, analogamente a come fornisce il proprio indirizzo email per riceve una mail.
Viceversa, per poter inviare bitcoin che fanno riferimento ad un determinato portafoglio occorre provare di esserne realmente i proprietari, attraverso la chiave privata associata alla chiave pubblica del wallet.
In realtà, come vedremo in seguito nella parte dedicata alle transazioni, la chiave pubblica e privata fanno riferimento non ai singoli portafogli elettronici ma a delle transazioni in ingresso bloccate, attraverso l'ausilio della crittografia, a favore di una determinata chiave pubblica.

1.2.1 Come creare un portafoglio

Per iniziare a usare Bitcoin bisogna innanzitutto creare un wallet, essendo una tecnologia opensource ci sono tantissime implementazioni del protocollo che si possono raggruppare in tre tipologie.

- Full Client [Figura 1.2]

  È un nodo della rete "completo" che memorizza l'intera storia delle transazioni, gestisce il portafoglio dell'utente e quindi le chiavi crittografiche ad esso associate ed infine può inviare e ricevere in maniera diretta transazioni nella rete bitcoin. Facendo sempre un paragone con la posta elettronica è come server mail che si occupa di tutti gli aspetti del protocollo di posta elettronica.

- Light Client

  È una soluzione più leggera rispetto ad un full client perché per funzionare non ha bisogno di scaricare l'intera blockchain. In locale sono presenti le chiavi necessarie per effettuare trasferimenti di BTC ma per interagire con la rete bitcoin fa affidamento a nodi di terze parti. Questo è molto simile ad un client mail installato sul proprio pc che per poter inviare e ricevere mail si connette a un server esterno.

- Web Client

  È una soluzione completamente centralizzata che memorizza l'intero portafoglio su un server posseduto da una terza parte. In questo modo il proprio wallet è accessibile da un qualsiasi browser. Continuando il confronto è come usare una web mail.

La scelta del client dipende dal tipo di controllo che l'utente vuole del proprio portafoglio. Installare un full client sicuramente offre il più alto livello di controllo ma richiede di scaricare e tenere constantemente aggiornata la blockchain e non permette l'uso dei propri bitcoin lontano da casa.
Inoltre l'utente deve essere in grado di mantenere sicuro il proprio sistema (sempre più malware bersagliano i wallet bitcoin[3]). Questa soluzione è quindi adatta soprattutto agli utenti più esperti.

Ma anche le soluzioni più alla portata alla massa non sono esenti da rischi, affidare il controllo del proprio portafoglio a terze parti può portare in caso di compromissione del server alla perdita dei propri bitcoin. [4]

Ricapitolando mediante un client possiamo generare un indirizzo personale rappresentato da una chiave pubblica legata crittograficamente ad una privata.

Conoscendo gli algoritmi su cui si basa creazione delle chiavi nulla vieta di generare in maniera "manuale" la propria coppia personale.

Una chiave privata è creata a partire da un numero causale di 256 bit al quale viene successivamente applicata la codifica base 58. Questo significa che abbiamo a disposizione 2^{256} possibili chiavi private e cioè 2^{256} = 115.792.089.237.316.195.423.570.985.008.687.907.853.269.984.665.640.564.039.457.584.007.913.129.639.936 *chiavi private possibili*

[3]http://www.forbes.com/sites/andygreenberg/2014/02/26/nearly-150-breeds-of-bitcoin-stealing-malware-in-the-wild-researchers-say/

[4]Recentemente un noto sito di exchange Cryptsy ha annunciato di esser stato derubato di oltre 13'000 bitcoin, quasi 5 milioni di euro. (http://blog.cryptsy.com/post/137323646202/announcement)

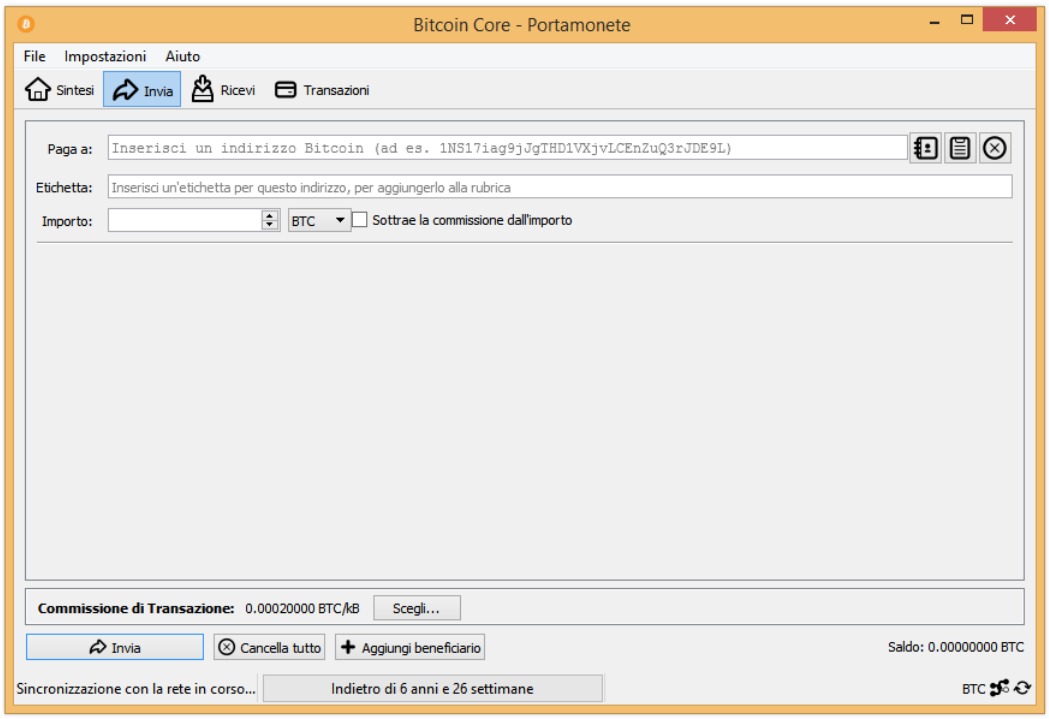

Figura 1.2: Il client bitcoin ufficiale nella schermata di invio.

Una volta scelta una chiave privata attraverso l'algoritmo Elliptic Curve DSA si ottiene la corrispettiva chiave pubblica di 512 bit.

Per ridurre lo spazio nella blockchain occupato dalle chiavi pubbliche questa viene a sua volta compressa a 160 bit utilizzando le funzioni di hash SHA256 e RIPEM-160 e infinde codificata in ASCII.
Il risultato è una coppia di chiavi del tipo:

CHIAVE PUBBLICA 144MLkNs6gZGEzHVwbNgMoxYciBWRsSq7Q

CHIAVE PRIVATA 2QMT37UQs4KhZjgbYlWdfP44hsoTHkgAr77mEEt GxMypKa22V2s

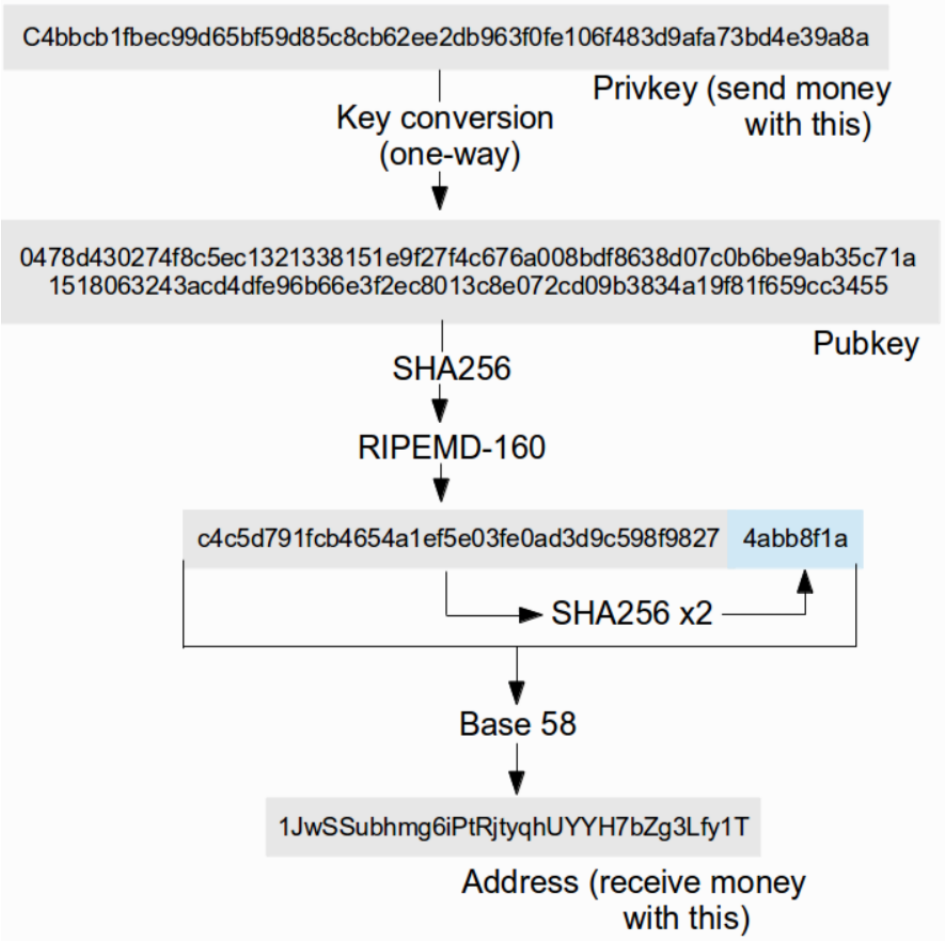

Figura 1.3: Processo di generazione della coppia di chiavi Bitcoin. [3]

Ognuno di noi può permettersi di possedere all'incirca 215.000.000 .000.000.000.000.000.000.000.000.000 indirizzi[5]. L'elevato ordine di grandezza di questi numeri ci garantisce, a livello matematico, che la probabilità che due utenti generino una coppia di chiavi uguali sia praticamente nulla assicurando così l'unicità degli indirizzi che di volta in volta vengono generati.

[5]https://en.bitcoin.it/wiki/Weaknesses#Generating_tons_of_addresses

Possiamo perciò sostenere che quando generiamo una chiave privata siamo praticamente certi che questa non sia mai stata generata in precedenza e, a meno che io non la riveli, non verrà mai casualmente creata in futuro. Concludiamo questa parte riguardante gli indirizzi evidenziando che a differenza di un conto bancario , il proprietario di wallet Bitcoin, non ha bisogno di svelare i propri dati personali per accedere al sistema. Infatti abbiamo visto come per poter iniziare a ricevere ed inviare moneta è sufficiente creare una coppia di chiavi crittografiche.

Le identità degli utilizzatori sono celate dietro le stringhe alfanumeriche dell'indirizzo pubblico, perciò, fin tanto che non è possibile collegare direttamente un wallet all'identità del suo possessore, ad esempio nel caso in cui quest'ultimo riveli informazioni personali durante uno scambio, il Bitcoin è una tecnologia che garantisce un livello alto di anonimato. Tuttavia come abbiamo visto e come vedremo più in dettaglio nella prossima sezione lo storico delle transazioni è completamente pubblico e consultabile da chiunque e in qualsiasi momento.

1.3 Transazioni

Le transazioni sono la parte più importante della cryptovaluta Bitcoin. L'intero protocollo è stato progettato per garantire che le transazioni possano essere create, diffuse nella rete, convalidate e infine aggiunte allo storico delle transazioni: la blockchain. Una transazione è il meccanismo tramite il quale viene ceduta la proprietà di una quantità di Bitcoin ad un altro indirizzo.

Il nuovo proprietario potrà a sua volta spendere i Bitcoin ricevuti

creando un'altra transazione che assegni parte o l'intero valore di questi Bitcoin al proprietario di un'altra chiave pubblica e così via creando praticamente una catena di proprietà. La proprietà delle monete da parte di un certo indirizzo si basa quindi solamente sulla base delle precedenti transazioni che lo hanno coinvolto.

Ad esempio, Alice, per inviare 5฿ [6] a Bob, deve far riferimento ad altre transazioni che ha ricevuto in ingresso che ammontino ad un totale di almeno 5฿. Queste transazioni sono chiamate transazioni di input. Ogni transazione contiene una prova di proprietà sotto-forma di firma digitale ed quindi spendere Bitcoin consiste nel segnare il valore di una transazione in input per la quale si ha la chiave privata corrispondente ed inviarla ad un nuovo proprietario identificato da un indirizzo pubblico.

La figura 1.4 mostra come funziona una catena di transazioni .

Nell'esempio Alice vuole pagare Bob per un totale di 0.02฿. Alice quindi deve avere almeno una transazione in input, del valore pari o maggiore del valore di Bitcoin che deve trasferire a Bob, ricevuti da una transazione precedente che nel nostro caso è rappresentata dalla numero 20.

La transazione 20 ha in effetti trasferito la proprietà di una transazione del valore di 0.152฿ da Joe ad Alice proprio perché Joe ha firmato la transazione aggiungendo la chiave pubblica di Alice. Questi 0.152฿ sono perciò vincolati alla chiave pubblica di Alice fin tanto che non si fornisca prova di essere a conoscenza della corrispettiva chiave priva-

[6] D'ora in poi useremo il simbolo ฿ per indicare la valuta spesa, allo stesso modo di come viene utilizzato il simbolo $ per indicare i dollari. https://en.bitcoin.it/wiki/Bitcoin_symbol

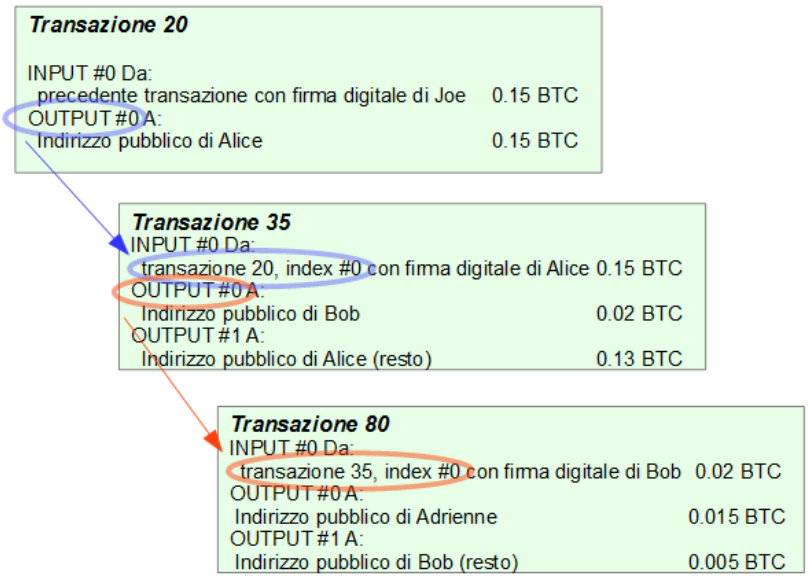

Figura 1.4: Esempio di catena di transazioni. [3]

ta.

La prova che Alice è a conoscenza della chiave privata avviene nel momento in cui Alice stessa, attraverso la transazione 35, decide di pagare Bob. Alice fornisce la chiave privata che sblocca la precedente transazione ricevuta in input da Joe (Transazione 35 input #0), dando così prova alla rete che è effettivamente la proprietaria dei 0.15฿ ricevuti.

Successivamente decide che 0.022฿ saranno destinati a Bob attraverso la transazione 35 output#0 mentre la rimanente parte 0.132฿ sarà restituita indietro ad Alice stessa sotto forma di resto attraverso la transazione 35 output#1. Bob a questo punto ha la possibilità di riscattare una transazione contenente 0.022฿ spendibile a sua volta attraverso con la transazione 80. Tutte queste operazioni sono trasparenti all'utente quando utilizza un portafoglio Bitcoin.

Nel software di un wallet è presente un algoritmo che seleziona gli

appropriati input e output in base all'importo da inviare indicato dall'utente. Se vogliamo però andare più in dettaglio una transazione è rappresentata digitalmente in formato binario con la struttura della tabella 1.1.

Dimensione	Campo	Descrizione
4 byte	Versione	Indica la versione del procollo Bitcoin
1–9 byte	Input	Contatore degli input
Variable	Inputs	Uno o più input legati a una transazione
1–9 byte	Output	Contatore degli output
Variable	Output	Uno o più output legati a una transazione
4 bytes	Locktime	Timestamp Unix o numero blocco

Tabella 1.1: Struttura di una transazione.[4]

1.3.1 Un esempio di transazione

Ecco una transazione di esempio realmente contenuta nella blockchain:

010000000184f1ca8667e8bef16ce206ee07ca06a9e9b83b2bd0d132ac3f9d15bef7ea8
db1000000008a47304402204d4734f8ede9c6b4c41cd38909057d79b652ae264717e90fd2
468ae8f1c6aae202200e24d93b2d9a9c8d48264202fe864df7b3b4094c266e5152fc09fe337
fd550e9014104ffc83a1835d9780d591c1ad98128b14afda69e0921ccfd139365f938b59564
aea68c3dc2fd2b2e989482ed9f1fa397a077b5f6c61abb05d6b97cfc9b4cd4fa2effffffff02239
0f001000000001976a914f0704ce2b364c12c4a82faa428f0c16db33c150f88ac8d8d040400
0000001976a9148655298c2fac774d85084e67e9b3c7ea7473bd2288ac00000000

Una transazione nella blockchain è immagazzinata in formato esadecimale.[7]
Ovviamente questo formato è poco human-readable e per questo lo stesso client bitcoin possiede delle API (Application_programming_interface)[8] per tradurre parti della blockchain in formato JSON(JavaScript Object Notation)[9] che ha il vantaggio di essere facilmente manipolabile da qualsiasi linguaggio di programmazione e al tempo stesso è chiaramente leggibile dall'uomo.

Ecco un formato più chiaro di una transazione:

[7] https://blockchain.info/tx/21ca709e800c216d73bc53b07e8a6ec8bdf3dfc6673400b4eb32c37fe50490e7?format=hex
Nella blockchain è rappresentata in formato binario, ma ovviamente ho usato il formato esadecimale per ragioni di spazio.

[8] https://en.wikipedia.org/wiki/Application_programming_interface

[9] (JavaScript Object Notation)

```
{txid:"21ca709e800c216d73bc53b07e8a6ec8bdf3dfc6673400b4eb32c37fe50490e7",
version: 1,
locktime: 0,
vin:
[{txid:
    "b18deaf7be159d3fac32d1d02b3bb8e9a906ca07ee06e26cf1bee86786caf184",
vout: 0,
scriptSig: {hex:
    "47304402204d4734f8ede9c6b4c41cd38909057d79b652ae264717e90fd246
8ae8f1c6aae202200e24d93b2d9a9c8d48264202fe864df7b3b4094c266e5152fc09fe337fd550e9
014104ffc83a1835d9780d591c1ad98128b14afda69e0921ccfd139365f938b59564aea68c3dc2f
d2b2e989482ed9f1fa397a077b5f6c61abb05d6b97cfc9b4cd4fa2e"},
n: 0,
addr: "1NvKhobLdzd4Cxp7UVa9Tw9nj5iRA4XuES",
value: 1}],
vout:
[{value: "0.32542755",
n: 0,
scriptPubKey: {asm: "OP_DUP OP_HASH160
    f0704ce2b364c12c4a82faa428f0c16db33c150f
OP_EQUALVERIFY OP_CHECKSIG",
type: "pubkeyhash",
addresses: ["1NvKhobLdzd4Cxp7UVa9Tw9nj5iRA4XuES"]}},
{value: "0.67407245",
n: 1,
scriptPubKey: {asm: "OP_DUP OP_HASH160
    8655298c2fac774d85084e67e9b3c7ea7473bd22
OP_EQUALVERIFY OP_CHECKSIG",
type: "pubkeyhash",
addresses: ["1DFHZtM7mbicJ442xC9G7HrSUDZWbUubzV"]}}
fees: 0.0005}
```

Linea 1 - è l'hash (identificatore univoco) della stessa transazione espressa in esadecimale.

Linea 2 - indica la versione del protocollo Bitcoin.

Linea 3 - è un timestamp o uno specifico numero di blocco che di fatto congela la transazione fino al tempo desiderato. Quando questo tempo scade la richiesta verrà raccolta dagli altri nodi della rete per poter essere confermata. Prima della scadenza la transazione rimane in uno stato di sospensione e può essere cancellata o modificata in uno o più parametri in qualsiasi momento.

Linee 4—13 - indicano la transazione di input con la quale specifichiamo da dove i soldi vengono e la rivendicazione di una transazione di output precedente. In particolare le linee 5 e 6 ci dice che la transazione di input fa riferimento ad una precedente transazione di output con hash "b18deaf. . . " e ci indica che deve essere preso in particolare l'output numero zero. La linea 7 contiene la firma della persona che invia i soldi creata a partire dalla transazione di output, seguita dalla corrispondente chiave pubblica.

La combinazione della firma con la chiave pubblica permetteranno di verificare che la transazione di input è effettivamente rivendicata dal legittimo proprietario. Inoltre la firma permetterà anche di garantire l'integrità della transazione inviata.

La linea 12 infine indica l'indirizzo dell'input e la linea 13 il valore che ha spedito, in questo caso 1฿.

Linee 14—28 - definiscono i due output della transazione, il primo definisce l'invio di 0.32542755฿ allo stesso indirizzo di partenza (è

come ottenere il resto in una transazione bitcoin) mentre l'altro consiste nella cessione di 0.67407245₿. Molto importanti sono le righe 17-18 e 23-24 che consistono in espressioni scritte nel linguaggio di Scripting Bitcoin che indicano con quali modalità sarà possibile sbloccare e quindi spendere gli output di questa transazione.

Una transazione rappresentando il trasferimento di una proprietà di un determinato valore di bitcoin da una coppia di chiavi ad un'altra. Lo script indica in che modo i bitcoin associati sono bloccati e perciò le modalità con cui dovranno essere rivendicati. Ad esempio, se Alice invia dei soldi a Bob attraverso lo script dovrà indicare una condizione di lock che può essere sbloccata soltanto da Bob che conosce la soluzione della condizione programmata. La maggiorparte delle transazioni processate nella rete bitcoin sono della forma Pay-to-Public-Key-Hash, conosciuta come transazione P2PKH, e contengono lo script di lock

```
“OP_DUP OP_HASH160 8655298c2fac774d85084e67e9b3c7ea7473bd2224
OP_EQUALVERIFY OP_CHECKSIG”.
```

La parte centrale è l'indirizzo in formato a 160 bit a cui trasferire i BTC , mentre le istruzioni sono dei comandi che verranno eseguiti per verificare, nel momento in cui questa transazione di output verrà rivendicata, che le firme crittografiche corrispondano.

In particolare per poter sbloccare questa transazione P2PKH di output, e quindi liberare i bitcoin inclusi in essa, si dovrà presentare la firma digitale della transazione creata con la chiave privata corrispondente alla chiave pubblica sopraindicata.

Questo linguaggio di script è sorprendemente complesso con 80 differenti possibili funzioni che permettono di formulare condizioni di rivendicazione di transazioni più complesse di cui vedremo qualche esempio nella prossima sezione.

Linea 27 - è la commissione pagata sulla transazione, per ora possiamo dire che più è alta e più velocemente la nostra transazione sarà confermata dai miner.

1.3.2 Il linguaggio Script Bitcoin: Casi Particolari

Il linguaggio di scripting Bitcoin è interamente basato su uno *stack* (o pila) (stack-based language) su cui si possono aggiungere (*push*) o eliminare (*pop*) elementi dalla cima della struttura dati. È volutamente limitato per ragioni di sicurezza: non è turing completo (non sono presenti loop) ed è stateless ovvero non vengono mantenuti stati prima o dopo la sua esecuzione. Queste caratteristiche sono dovute al fatto che ogni nodo bitcoin deve eseguire ogni script di ogni transazione per verificarne la validità, e quindi è indispensabile che il risultato sia facilmente prevedibile.[10]

A parte le transazioni di tipo P2PKH ci sono molte combinazioni di cui noi vedremo tre casi particolari.

Multi-Signature

Gli script di tipo multi-signature sono anche detti script "M di N" poichè sono composti da un elenco di N chiavi pubbliche e di un numero M che indica quante chiavi sono richieste per sbloccare la transazione.

[10] https://en.bitcoin.it/wiki/Script

Ad esempio in un schema "2 di 3" sono presenti 3 chiavi pubbliche e ne sono necessarie almeno due per poter sbloccare l'importo di output inviato. Il codice usato ha questa struttura:
"M <Public Key 1> <Public Key 2> . . . <Public Key N> N OP_CHECKMULTISIG"

Pay-to-Script-Hash

Pur essendo molto potenti gli output che fanno uso di multi-signature sono spesso difficili da creare e successivamente da spendere, per semplificare queste problematiche e avere comunque uno script molto personalizzabile sono nati gli script Pay-to-Script-Hash (P2SH).
L'output è semplicemente bloccato da un impronta digitale, un hash crittografico.
Quando una transazione tenta di spendere questo UTXO deve contenere lo script che passato ad una funzione di hashing combaci con l'hash indicato dell'output.
Il codice usato ha questa struttura:
"OP_HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e OP_EQUAL"

Output inspendibili: messaggi nelle transazioni

Usando l'istruzione OP_RETURN si indica immediatamente che lo script non è valido garantendo che non esiste alcuna chiave per poterlo sbloccare, quindi l'output viene rimosso dalla lista degli UTXO (Unspent Transaction Output, ovvero un output che può essere speso in una nuova transazione) [11] e eventuali valori associati a questa istruzione sono dati come commissione al miner che ha convalidato il blocco.
L'uso più comune di questa istruzione è quello di lasciare dati (fino a

[11] https://bitcoin.org/en/glossary/unspent-transaction-output

40 byte) legati ad una transazione all'interno della blockchain, con il vantaggio che questi messaggi sono associati ad un timestamp, sono immutabili e accessibili da ovunque.
Si possono quindi usare questi script per lasciare veri propri messaggi che possono essere contratti vincolanti (ad esempio è possibile provare la paternità di un opera in una determinata data [12]) ma anche contenuti meno seri.
Ad esempio nella transazione "8bae12b5f4c088d940733dcd1455efc6 a3a69cf9340e17a981286d3778615684" traducendo il primo output "OP_RETURN 636861726c6579206c6f766573206865696469" in codifica ascii diventa "charley loves heidi".[13]
La blockchain può essere usata persino per messaggi d'amore!

Arrivati a questo punto Satoshi Nakamoto aveva già a disposizione un modello teorico funzionante per scambiarsi le proprietà di una moneta digitale. Ma esempi di moneta digitale già esistevano da almeno un decennio prima del Bitcoin come ad esempio HashCash o Ecash. [7] Allora in cosa è diverso il bitcoin da altri tipi di monete digitali come quelle appena menzionate? La vera e straordinaria innovazione alla base del Bitcoin è che è stata costruita per essere una tecnologia completamente decentralizzata.
Gli schemi di moneta elettronica precedentemente implementati avevano bisogno che una qualche autorità centrale tenesse traccia di tutte le transazioni e, perciò, degli scambi di moneta che intercorrevano per tenere traccia dei saldi di ogni conto e le persone che lo possedevano.

[12] https://bitcore.io/guides/i-made-this/

[13] http://bitcoin.stackexchange.com/questions/29554/explanation-of-what-an-op-return-transaction-looks-like

In poche parole se Alice voleva scambiarsi del denaro con Bob doveva passare per Eva come intermediario dello scambio.
Satoshi Nakamoto voleva che questa struttura centralizzata venisse meno e dare così la possibilità ad Alice e Bob di scambiarsi il denaro in maniera diretta o se vogliamo peer-to-peer.
Voleva una moneta digitale emessa e trasferita nella rete senza l'utilizzo di una terza parte centrale ma con la rete stessa in grado di accertare e convalidare e accordarsi sulle transazioni trasmesse e quindi delle proprietà del bitcoin.
Ma per far sì questo doveva risolvere due problemi informatici che si pensava irrisolvibili: il problema del double spending e dei generali bizantini. Vedremo come nel prossimo capitolo.

1.4 Blockchain

Nel mondo digitale se possediamo un qualsivoglia token digitale come ad esempio un file musicale mp3 non c'è nulla che impedisca di effettuare una copia perfetta del file e ridistribuirla ad un numero potenzialmente infinito di persone.
Capiamo bene che questa possibilità di copiare i bit in maniera perfetta era un problema anche per tutti coloro che volevano creare una moneta digitale. Se Alice poteva fare una copia perfetta del proprio cd preferito perché allora non fare anche una copia dei 5 bitcoin che possedeva nel proprio wallet e spenderli almeno due volte?! Questo appena illustrato è il problema del double spending ossia la possibilità di spendere due volte la stessa moneta e per decadi la comunità scientifica in particolare matematici ed informatici non avevano idea di come risolverlo.

Prima del Bitcoin si pensava che l'unica maniera possibile per evitare la copia di una moneta digitale era quella di avere un'entità centrale che disponendo della fiducia di tutti poteva tenere traccia di chi aveva quale copia del token.
Una banca, Paypal, Visa o Western Union di fatto funzionano così; questi enti o società di servizi finanziari basicamente tengono traccia dei conti di ciascun utente in un database centrale, che può essere visto come un grande "libro mastro", e nel momento in cui richiediamo di effettuare un pagamento ad un altro soggetto controllano se possediamo la somma in questione e solo allora autorizzano il trasferimento di denaro prevenendo cosi ogni tentativo di spendere dei soldi che eventualmente non si possiedono.
Satoshi Nakatomo ha risolto il problema pensando di distribuire a tutti i nodi della rete una copia aggiornata del libro mastro delle transazioni che venga interrogata ogni qual volta Alice decida di inviare dei bitcoin a Bob.
Nel Bitcoin perciò questo registro non si trova nei server di Paypal o di una qualche banca ma è distribuito e copiato in tutti i nodi della rete ed è aggiornato ogni 10 minuti con le transazioni che sono nel frattempo avvenute.
Ogni aggiornamento viene chiamato blocco e ognuno di questi blocchi è legato crittograficamente l'uno con l'altro in maniera tale d'avere una vera e propria catena chiamata appunto blockchain che riassume l'intera storia delle transazioni che sono avvenute dal gennaio 2009 ad oggi nella rete bitcoin.
Catena che se ripercorsa a ritroso ci condurrà al primo blocco Bitcoin mai creato chiamato blocco di genesi creato da Satoshi Nakamoto.

È bene capire che il concetto di blockchain è differente dalla catena di transazioni descritta nella precedente sezione. Mentre la catena di transazioni tiene traccia di come le proprietà dei Bitcoin cambiano, la blockchain è invece una catena di blocchi, cioè un insieme di più transazioni, utilizzata per ordinare temporalmente le transazioni che avvengano nella rete.

1.4.1 Struttura della blockchain

La blockchain (figura 1.5) è perciò una struttura dati, contenente la lista dei blocchi delle transazioni legati tra loro attraverso una funzione hash. Infatti ogni blocco nella catena è identificato unicamente con un hash, generato utilizzando l'algoritmo crittografico SHA256, che si trova nell' header del blocco stesso.
Ogni blocco contiene anche l'hash del blocco precedente chiamato blocco padre. Quest'ultima caratteristica comporta inevitabilmente che l'hash di un blocco figlio è strettamente legato all'hash del blocco padre.
Perciò, l'identità, sotto forma di hash, di un blocco figlio è modificata nel caso in cui l'identità del padre cambia. Quindi, nel caso in cui si abbia un'alterazione di un blocco, dovuta ad esempio ad una modifica effettuata anche in una sola delle sue transazioni interne, questa genera automaticamente l'invalidazione dell'hash del blocco figlio creando un effetto a cascata lungo tutta la catena.
Questo vuol dire che un blocco con molte generazioni di blocchi a seguire non è modificabile a meno che non vengono modificati gli hash di tutti i blocchi che lo seguono. E siccome questo ricalcolo, come vedre-

mo nel dettaglio nel prossimo capitolo, richiede un costo computazionale enorme, la blockchain e i dati al suo interno sono immodificabili e permanenti. Questa caratteristica rappresenta uno degli aspetti fondamentali della sicurezza del Bitcoin.

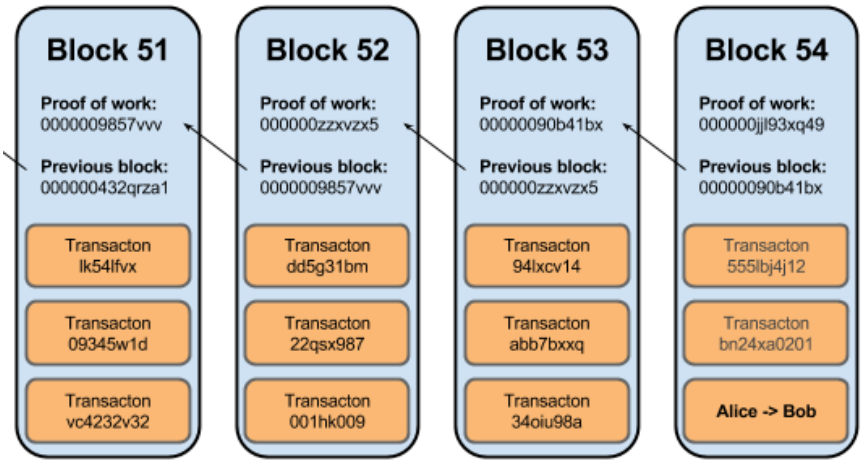

Figura 1.5: Blockchain. [4]

Entrando più nel dettaglio della struttura di un blocco questa è formata da un header contenente dei metadati seguita da tutta la lista delle transazioni che formano il grosso della sua dimensione. L'header del blocco, in giallo nella figura 1.6, contiene dei campi che descrivono il blocco stesso.

Il primo campo è la versione del protocollo seguito dall'hash del precedente blocco in ordine temporale inserito nella blockchain, collegamento che ci assicura che i blocchi formino una catena inalterabile.

Da questa proprietà è possibile definire l'*altezza* di un blocco come la distanza del blocco dal blocco di genesi.

Il campo seguente è il merkle root, un hash speciale di tutte le transa-

zioni nel blocco che è utile per individuare se una transazione è inclusa o no in un blocco. Un merkle tree, infatti, è costruito in maniera ricorsiva facendo gli hash dei suoi nodi a due a due.
Quando di N elementi, nel nostro caso di N transazioni, viene fatto l'hash e vengono inseriti in un merkle tree è possibile stabilire in un tempo 2*log(n) se un determinato elemento ne fa parte oppure no, rendendo questa struttura dati molto efficiente.
Infine abbiamo un campo timestamp approssimativo del blocco e i campi bits e nonce che sono utilizzati nel processo di mining e saranno discussi nel prossimo paragrafo.
Nel corpo di ogni blocco avremo la lista di tutte le transazioni inserite in quel blocco più una transazione speciale chiamata coinbase che è la transazione che contiene i nuovi bitcoin creati dal sistema.

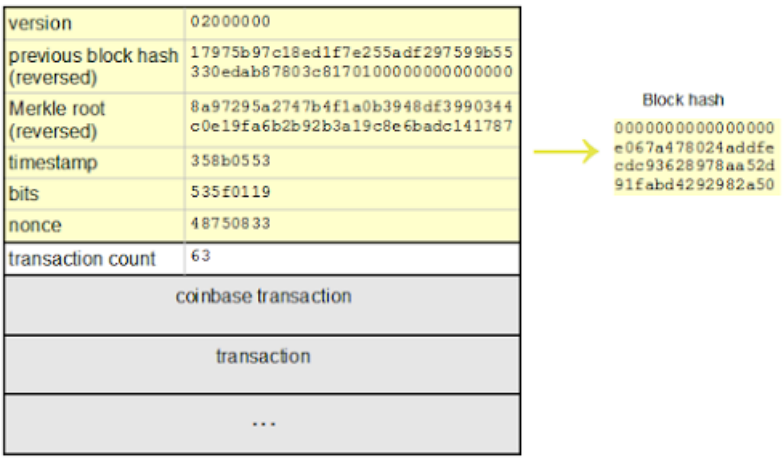

Figura 1.6: Dettaglio di un singolo blocco. [3]

In questo capitolo abbiamo studiato la blockchain, il libro mastro pubblico di tutte le transazioni, che viene visto da tutti i partecipanti alla rete Bitcoin come il database autorizzato delle proprietà della mo-

neta.
Ma il fatto che la copia della blockchain venga distribuita non basta per rassicurarci che la rete è in accordo su un'unica "verità" universale riguardo allo stato della blockchain.
Chi garantisce che non circolino diverse versioni della blockchain? Come possiamo sapere quale di queste sia quella "giusta"? Cercando di rispondere a queste domande capiamo bene come, per permettere al Bitcoin di mantenere il suo carattere decentralizzato, Satoshi Nakatomo avrebbe dovuto creare un modo per consentire ad ogni nodo della rete, sulla base delle informazioni che viaggiano attraverso un canale insicuro, di arrivare alle medesime conclusioni di tutti gli altri nodi e cioè avere una medesima copia della blockchain che rappresenti un'unica verità sullo stato del sistema in un determinato istante di tempo.
Praticamente avrebbe dovuto creare un algoritmo di consenso distribuito in una rete insicura e asincrona senza la presenza di un'entità centrale fidata.
La soluzione ingegnosa di Satoshi al problema appena esposto utilizza un sistema di proof of work che prende nome di Bitcoin Mining e che analizzeremo nella prossima sezione.

1.4.2 Proof of Work: Mining di Bitcoin

Il sistema Bitcoin ordina tutte le transazioni raggruppandole in gruppi chiamati blocchi i quali sono collegati gli uni agli altri crittograficamente formando cosi una catena chiamata blockchain.
Se le transazioni non confermate sono inviate in broadcast alla rete ogni nodo può collezionare le transazioni in un proprio blocco e inviarlo agli altri nodi come proprio suggerimento di quale sarà il successivo blocco

che dovrà far parte della blockchain.
Ma il fatto che ogni nodo può creare la propria versione ci suggerisce che ben presto nella rete ci saranno molte versioni dei blocchi da quali scegliere. Come allora trovare un accordo? Come stabiliamo quale blocco sarà quello corretto e quale no? Non possiamo semplicemente far affidamento all'ordine degli arrivi dei blocchi in quanto la rete è asincrona.
Questo problema di accordo in rete è analogo al problema dei Generali Bizantini formulato per la prima volta da Marshall Pease, Robert Shostak e Leslie Lamport [4]. Il problema è il seguente: un gruppo di Generali bizantini sono accampati, ciascuno con il proprio esercito al seguito, intorno ad una città nemica.
Essi, per poter attaccare con successo la città rivale, devono trovare una strategia comune di attacco potendo comunicare soltanto a distanza attraverso l'invio di informazioni trasportate da messaggeri; non possono quindi incontrarsi ad esempio in un singolo luogo e decidere ma di contro posso inviare quanti messaggi vogliono, con la frequenza che desiderano e in diversi istanti di tempo.
I messaggeri nel tentativo di trasportare il messaggio possono fallire nel loro intento e allo stesso tempo non è da escludere la possibilità che uno o più di uno di questi generali possa essere alleato con il nemico e tenterà di confondere le idee a tutti gli altri e di far saltare gli accordi d'attacco.
La risoluzione del problema è quella di trovare un algoritmo che assicuri che i generali leali possano trovare una strategia comune di attacco per distruggere la città nemica. Fino al 2008 si pensava che questo problema potesse essere risolvibile soltanto quando più dei due terzi

dei generali fossero fedeli alla causa. [5]
Bastava perciò che un terzo dei generali fosse alleato con il nemico per avere la certezza che i generali onesti non potessero avere la possibilità di trovare un accordo e perciò sferrare un attacco vincente alla città nemica. Bitcoin risolve il seguente problema inserendo un costo da pagare obbligatoriamente per ogni invio di un blocco pena il rifiuto del blocco stesso. Il costo che viene aggiunto è un costo computazionale chiamato "proof of work" ed è il calcolo di una particolare funzione hash. Il concetto di proof of work non è nuovo nell'informatica.
Nel 1997, un crittografo inglese chiamato Adam Back, per risolvere il problema dello spam nelle caselle di posta elettronica, propose per la prima volta un approccio in cui l'idea di base è che un messaggio di mail doveva contenere una prova di risoluzione di un problema matematico computazionalmente difficile (proof-of-work), relazionato al contenuto del messaggio stesso. [6]
Una caratteristica chiave di questo problema è la sua asimmetria: il lavoro deve essere moderatamente complesso, ma fattibile, dal lato richiedente, ma facile da controllare per il fornitore del servizio in maniera tale che una mail non contenente questa prova sarebbe stata scartata dal server e-mail.
Così facendo si rende antieconomico per uno spammer mandare messaggi in serie perché, aggiungendo un costo ad ogni invio, si crea un deterrente efficace sotto forma di spese nel costo dell'energia elettrica sprecata da un computer nel risolvere la proof-of-work.
Ma di contro, un utente ordinario del servizio di posta è ben disposto a spendere del lavoro computazionale perché, per un singolo messaggio di posta, questo risulta accettabile. Nonostante la tecnica anti spam

proposta da Adam Black era geniale, in realtà non funzionò perché gli spammers oramai utilizzano delle botnets per inviare mail.

Ma l'idea dietro Hashcash fu presa e usata per il Bitcoin. La proof of work implementata nel bitcoin è la seguente: l'hash di un blocco deve essere ottenuto incrementando un numero nonce nel blocco finché non si trovi un valore di hash che abbia un numero di zeri iniziali prestabilito. Vediamo un esempio calcolando l'hash di un blocco e modificando di volta in volta il nonce:

```
1 Hash256(Dati blocco, 0) =
      82d45fcc6844102013f5dd8df9adfc8c9f3494b34f25dafa22c4128a60913b77
2 Hash256(Dati blocco, 1) =
      aabe272a833fbb749e06b456b5ddd1b7ce5e84ac03004c824e5c865164619e9b
3 Hash256(Dati blocco, 2) =
      5ef127ba2406c0f6e9337214dcd5af4716958d893481ba3613da5a327e82fe19
4 Hash256(Dati blocco, 3) =
      90ee57eeee0451dce1bd8f9ce81479bf616a50478a777cd1d68b1afb98f995eb
5 Hash256(Dati blocco, 4) =
      55e6d130a7840655e5f07894009b19b7450158061864c803611f0acd91770f8f
6 Hash256(Dati blocco, 5) =
      4c307fda293a9f984e92d07ceae0a84d63c2f7a21e236601c2e38c9efad738c2
7 Hash256(Dati blocco, 6) =
      fb04cd046c79a858a066e7e7f1246b7f6292e8658ea841e271385fe40ea3787a
8 ...
9 Hash256(Dati blocco, 24) =
      09678b086b27c1ad34627f68f58cd00d95fbb3bda0f7aad1da46c50594c50ce9
```

Ogni stringa in ingresso produce un hash completamente diverso. Il numero dopo la virgola è il nostro nonce che è utilizzato per cambiare l'output della funzione hash. Se la proof-of-work fosse quella di trovare un hash di un blocco con un zero iniziale vediamo come al 24-esimo tentativo troveremo la soluzione. L'output è totalmente imprevedibile e l'unica maniera per trovare la soluzione è quello di provare tutte le possibili combinazioni. Non possiamo fare diversamente! E' un po' come provare ad indovinare le combinazioni di un lucchetto. Occorreranno tanti tentativi prima di trovare la combinazione giusta.
Quindi una volta che un nodo ha costruito il proprio blocco sa che può inviarlo solo se avrà la fortuna di trovare prima di tutti gli altri nodi la proof-of-work altrimenti la sua proposta sarà respinta dalla rete.
Perciò non gli resta altro che iniziare a calcolare il suo hash e vedere se l'output è quello richiesto oppure no. Se l'output non è quello giusto cambierà il nonce e ritenterà nuovamente. Se invece è fortunato a trovare la soluzione giusta lo comunica alla rete ed invia il blocco in broadcast.
A questo punto gli altri nodi verificando la correttezza dei dati contenuti nel blocco (quindi anche la validità delle transazioni inserite) e la proof-of-work aggiornano la blockchain con il blocco appena confermato.
Questo perché tentare di produrre un blocco alternativo a quello già inviato in broadcast è rischioso in quanto si dovrebbe convincere abbastanza nodi a costruire una catena alternativa con la possibilità di sprecare soltanto tempo computazionale e perciò soldi.
Accettando invece la catena più lunga si crea un consenso naturale ed ogni nodo sa che se sarà abbastanza fortunato a generare il prossimo

blocco valido esso sarà accettato per la medesima ragione con cui loro hanno accettato l'ultimo disponibile.
Con questo sistema la rete Bitcoin riesce a trovare sempre un consenso perché la totale causalità della proof-of-work ci assicura che difficilmente avremo due nodi che simultaneamente troveranno la soluzione. Occasionalmente questo però può accadere portando la blockchain a dividersi in due rami (blockchain fork) [Figura 1.7].
Se due nodi inviano simultaneamente in broadcast differenti versioni del successivo blocco validato, qualche nodo riceverà prima un blocco rispetto l'altro.
In questo caso loro sono forzati a lavorare ad uno dei due blocchi che hanno ricevuto ma allo stesso tempo salvano l'altro ramo della blockchain nel caso diventasse la catena più lunga.
La condizione di "pareggio" nella lunghezza sarà interrotta quando il successivo blocco verrò validato e uno dei due rami diventerà il più lungo; quindi, i nodi che lavoravano nella catena incorretta per loro convenienza effettueranno una switch nell'altro ramo, ora più lungo e perciò considerato più favorevole.
Il consenso è così ripristinato!

Inoltre quei nodi che agiranno disonestamente non solo avranno il loro blocco respinto ma avranno sprecato lo sforzo necessario per produrre una proof-of-work valida in termini di costi di elettricità.
Quindi meglio per loro costruire un blocco con transazioni al suo interno valide.
Ora nel nostro esempio per trovare un hash con un solo zero iniziale abbiamo impiegato solamente 24 tentativi.

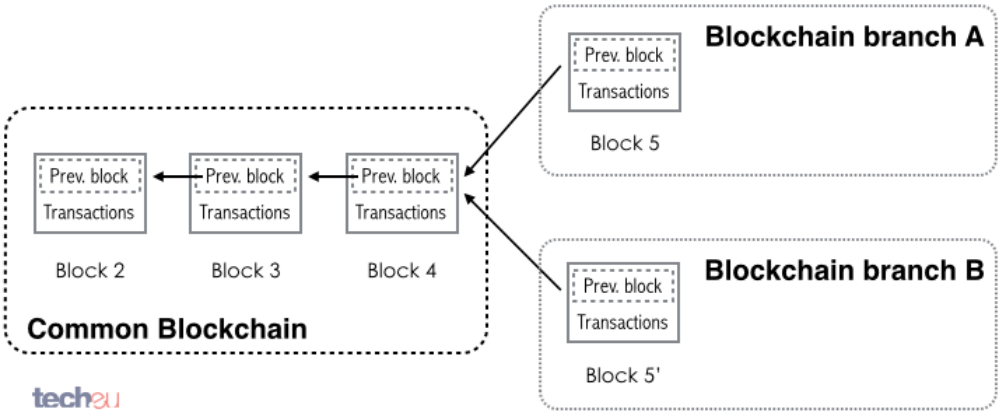

Figura 1.7: Esempio di fork della blockchain.

Ma il lavoro computazionale richiesto per risolvere questo problema matematico è esponenziale nel numero di bits a zero richiesti. Quindi capiamo bene come il numero di zeri influenza il numero di tentativi medi che dobbiamo effettuare per trovare una soluzione della proof of-work. La difficoltà nella rete Bitcoin e quindi il numero di zeri iniziali sono allora calibrati in maniera tale che un blocco è validato ogni 10 minuti circa.

Quindi più la rete Bitcoin ha maggiore capacità computazionale complessiva più la difficoltà nel trovare la soluzione alla proof of work deve aumentare al fine di avere una soluzione ad un blocco mediamente ogni dieci minuti.

Perciò ogni 2016 blocchi validati, tutti i nodi sono forzati a riaggiustare la difficoltà della proof-of-work sulla base del tempo che ci è voluto per trovare gli ultimi 2016 blocchi rapportata al tempo che vogliamo per i prossimi 2016 blocchi ossia 2016 x 10 = 20160 minuti.

L'equazione scritta nel codice sorgente è la seguente:

Nuova difficoltà = Vecchia difficoltà * (Tempo impiegato per gli ultimi 2016 blocchi / 20160 minuti)

Al momento in cui scrivo è stato validato il blocco n°404407 e la capacità complessiva di tutti i nodi della rete Bitcoin è di 1,209,353,167.6 GH/s![14]

Produrre una modifica in un blocco già inserito nella blockchain significa sfidare i 1209 PH/s della rete Bitcoin.

Un attaccante dovrebbe sopravanzare la capacità della rete per un tempo abbastanza lungo da rimpiazzare un blocco nella catena, calcolare il suo hash e cercare di costruire una blockchain alternativa più lunga dell'attuale in maniera tale da convincere tutti gli altri nodi ad effettuare lo switch nella sua catena. Il sistema di proof-of-work ci garantisce che le transazioni in un blocco sono protette da una gara matematica che oppone un attaccante contro il resto dell'intera rete.

Un computer singolo impiegherebbe diversi anni (nel caso più fortunato) prima di trovare la proof-of-work, perciò la possibilità individuale di risolvere un blocco prima del resto della rete che impiega all'incirca dieci minuti è praticamente nulla.

Tutto questo processo che abbiamo appena descritto prende il nome di Bitcoin mining e i nodi che partecipano nel risolvere la proof-of-work di un blocco prendono il nome di miner.

Il nome miner (minatore, attività estrattiva dei bitcoin) non è causale.

Infatti se il mining è il processo attraverso il quale si verificano e si

[14] 1 GH/s = 1,000,000,000 hash calcolati al secondo, http://bitcoin.stackexchange.com/questions/9219/what-is-the-difference-between-kh-s-mh-s-and-gh-s

mettono in sicurezza le transazioni avvenute nella rete la ricompensa di tale lavoro è invece l'immissione e creazione di nuova moneta nel sistema. Infatti per avere possibilità di successo il Bitcoin deve incentivare i minatori a mettere a disposizione della tecnologia la propria capacità computazionale.

Senza questo incentivo chi spenderebbe tempo e denaro solamente per validare le transazioni di altri nodi? La soluzione a questo problema è di ricompensare quei i nodi che aiutano a validare nuovi blocchi. In particolare il nodo che risolve la proof of work e quindi scopre un nuovo blocco valido viene ricompensato con la ricezione di nuovi bitcoin. Ogni quattro anni la ricompensa si dimezza fino a quando nel 2040 nessun bitcoin verrà mai più prodotto arrivando al numero massimo di 21 milioni [Figura 1.8].

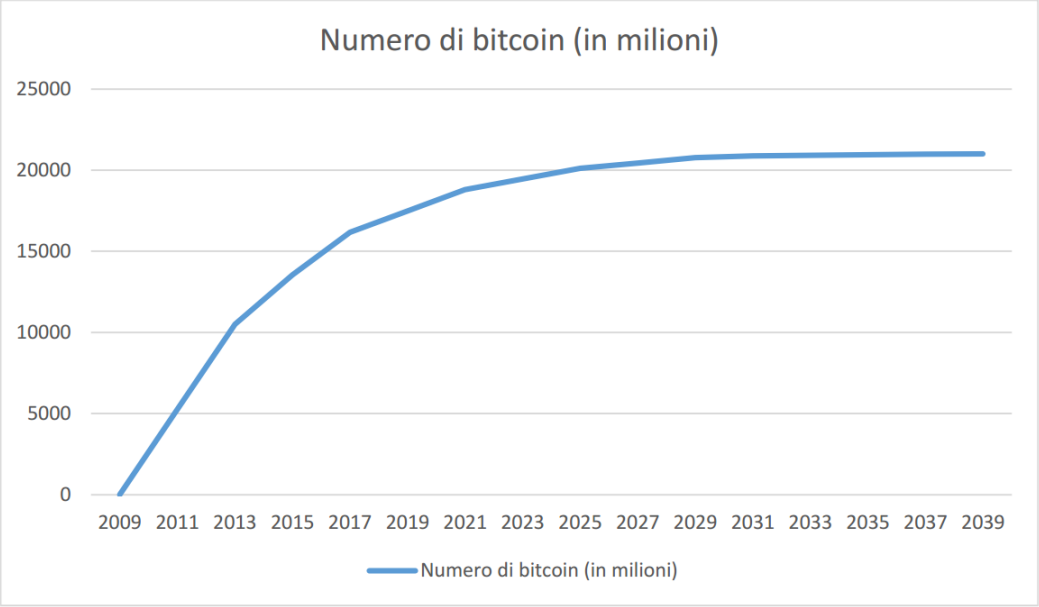

Figura 1.8: Numero di bitcoin sul mercato al passare degli anni.

Ma una volta che la ricompensa verrà a mancare quale sarà l'incentivo che avranno i nodi minatori per validare nuovi blocchi?
Oltre alla ricompensa per la soluzione del blocco, i minatori ricevono anche delle commissioni di transazione che vengono incluse in maniera opzionale da chi invia una transazione.
In futuro in un ambiente con scarsità di bitcoin i nodi anziché finanziarsi con la creazione di nuovi bitcoin trarranno profitto dalle commissioni che probabilmente aumenteranno.

I moderni sistemi di pagamento elettronico sono costruiti sulla base di un modello di fiducia che viene riconosciuto a terze parti centrali di processare in maniera sicura lo scambio di denaro tra soggetti. Gli ultimi sviluppi hanno visto la creazione di una moneta digitale, il Bitcoin, che ha combinato la creazione di una nuova moneta con un modello distribuito di pagamento dove i soggetti possono scambiarsi denaro senza intermediari.
Sebbene l'aspetto monetario del Bitcoin ha attratto la principale attenzione, il libro mastro o registro distribuito delle transazioni chiamato blockchain, che alla base del sistema di pagamento, e il modo con cui i nodi della rete prima si accordano e poi lo aggiornano sono i veri e significativi aspetti innovativi.
Riepiloghiamo, perciò, tutti gli step che permettono a questo sistema di funzionare [Figura 1.9] .

1.5 Riepilogo e Conclusioni

Figura 1.9: Numero di bitcoin sul mercato al passare degli anni.

1. CREAZIONE DELLA TRANSAZIONE

   Alice che vuole pagare 1 BTC a Bob crea un nuova trasazione che contiene: un riferimento (transazione di input) alla transazione precedente attraverso la quale ha ricevuto i bitcoin, l'indirizzo o chiave pubblica di Bob, la somma di bitcoin da inviare.

2. FIRMA DELLA TRANSAZIONE

   Una volta che il messaggio è stato creato Alice lo firma digitalmente per dimostrare alla rete che è in controllo dell'indirizzo di pagamento e cioè della transazione di input precedentemente indicata. In questo modo soddisfa la condizione di rivendicazione della transazione di output non spesa a cui la transazione di input fa riferimento.

3. INVIO IN BROADCAST DELLA TRANSAZIONE

   Alice invia alla rete il messaggio per permetterne la sua verifica con successiva validazione nella blockchain.

4. VERIFICA DELLE TRANSAZIONI (MINING)

   I nodi minatori in ascolto sulla rete ricevono tra le altre anche la transazione di Alice e la inseriscono in nuovo blocco da verificare. Validare un blocco comporta comprovare che tutte le transazioni siano ben formate e valide e questo principalmente significa verificare che esse fanno riferimento a transazioni di output non ancora spese e che le firme digitali inserite siano corrette.

   Quindi, i nodi minatori, iniziano a competere l'uno con l'altro per comunicare il nuovo blocco da loro formato cercando di essere i primi nella rete a trovare la sua corretta proof-of-work.

5. UN NODO MINATORE TROVA LA SOLUZIONE ALLA PROOF-OF-WORK DEL PROPRIO BLOCCO

   Un nodo minatore, prima di tutti gli altri, individua la proof-of-work del blocco in cui la transazione di Alice è inserita, l'invia in broadcast alla rete e se, dopo verifica degli altri miners, tutto è corretto, riceve la ricompensa in bitcoin prevista dal sistema. Gli altri nodi minatori "sconfitti" non possono fare altro che aggiungere il blocco alla copia della blockchain che possiedono e ritornare al passo 4.

   Bob a questo punto ha ricevuto il bitcoin inviato da Alice. Ricevere il bitcoin significa avere a disposizione la soluzione allo script di lock inserito nella transazione di output non spesa ricevuta da Alice.

La tecnologia realizzata da Satoshi Nakatomo, riepilogata in questo paragrafo conclusivo, è una metodologia per raggiungere un consenso distribuito su larga scala.
Ciò significa che per la prima volta nella storia, un numero vasto di persone sono capaci attraverso la rete di raggiungere un accordo sullo stato di una grande quantità di dati arbitraria senza l'ausilio di un'autorità centrale.
Studiare questo insieme di dati perciò è molto interessante, e per fare ciò occorre costruire uno schema per poter studiare questo costante flusso di moneta attraverso dei grafi. Tuttavia non è semplice raccogliere questi dati, ogni giorno vengono creati 144 nuovi blocchi con oltre 200000 transazioni aggiunte alla blockchain.[15]

Al momento ci sono circa 406000 blocchi e oltre 117 milioni di transazioni[16], per poter analizzare questa grande mole di dati occorre analizzare la blockchain come un *Big data*: questo sarà l'argomento dei prossimi capitoli.

[15] https://blockchain.info/stats (ultima consultazione 7/4/2016)
[16] http://blockr.io/trivia/blockchain (ultima consultazione 7/4/2016)

Capitolo 2

Big Data

Big data è un termine usato per indicare insiemi di dati così grandi o complessi da rendere inadeguate le tradizionali tecniche di *data processing*.

Sfide che coinvolgono i big data sono l'analisi, la raccolta, la ricerca, la condivisione, l'immagazzinamento, la visualizzazione e la capacità di fare *query*[1].

In questo capitolo ci concentreremo soprattutto sulla parte riguardante l'analisi dei dati ed il successivo immagazzinamento in una base di dati per successivamente effetturare query. I *data set* stanno crescendo rapidamente soprattutto per l'incremento del ruolo della tecnologia nella vita di tutti i giorni.

Ci sono sempre più sensori, fotocamere, microfoni e i social network che portano alla generazione di una quantità sempre più grande: la capacità pro-capite di immagazzinare informazioni è raddoppiata ogni 40 mesi dagli anni 80 [7]; nel 2012 ogni giorno venivano creati 2.5 exabyte

[1] Il termine query viene utilizzato per indicare l'interrogazione da parte di un utente di un database

di dati. [2]

Perciò è fondamentale per le aziende e per la ricerca gestire e analizzare i big data a proprio vantaggio.

2.1 Caratteristiche

I Big data possono essere descritti secondo le seguenti caratteristiche: [8]

1. Volume

   La quantità di dati generati e immagazzinati. La dimensione dei dati determina il valore e la potenziale comprensione dell'insieme di dati, e se possono essere considerati big data oppure no. La blockchain occupa circa 70 gb ed è la sua dimensione cresce linearmente. [3]

2. Varietà

   Il tipo e la natura dei dati. Questo aiuta gli analisti a capire come effettivamente usare gli insieme di dati per comprenderli.

3. Velocità

   Si intende la velocità con la quale i dati sono generati e processati. Ogni giorno vengono aggiungi circa 144 blocchi alla blockchain!

4. Variabilità

   L'inconsistenza dei dati che può rendere difficoltoso il processo di

[2] http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html, 1 exabyte è uguale a un miliardo di gigabyte

[3] https://blockchain.info/charts/blocks-size

gestione e analisi dei dati. Non sempre i dati delle transazioni sono consistenti, ad esempio una transazione può contenere output non spendibili oppure solamente output di resto.[4]

5. Veridicità

   La qualità dei dati catturati può variare enormemente influenzando la possibilità di fare analisi corrette.

I dati devono essere processati con strumenti avanzati per ricavare informazioni con un significato. Questo processo è detto *data analysis*.

2.2 Data analysis

L'analisi dei dati è un processo che, partendo dai dati presi dalla fonte (*raw data*), li trasforma e inserisce in un modello con l'obiettivo di scoprire informazioni utili, suggerire conclusioni e supportare decisioni.

Possiamo pensare a delle fasi comuni (schematizzate nella figura) sempre presenti nel processo di data analysis. Queste fasi devono esser considerati iterativi, ovvero i feedback delle fasi successive possono risultare in addizionale lavoro nelle fasi precedenti.[9]

1. Requisiti dei dati

   I dati necessari come input per l'analisi sono specificati come requisiti per coloro che effettueranno l'analisi e successivamente per gli utenti che useranno il prodotto finito per l'analisi.

   Il tipo di entità che verrà collezionata è chiamata unità sperimentale (ad esempio una persona, o un gruppi di persone).

[4] Ovvero valuta che torna al proprietario.

[5] https://en.wikipedia.org/wiki/File:Data_visualization_process_v1.png

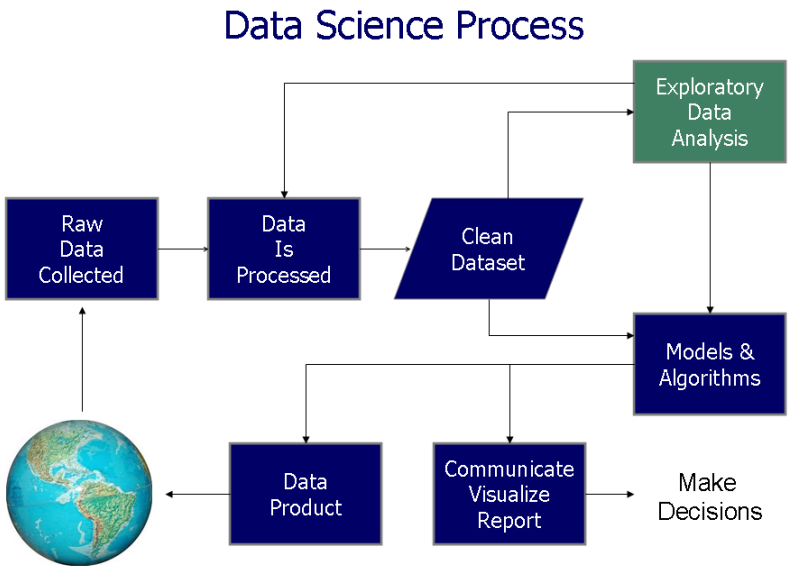

Figura 2.1: Processo di analisi dei dati[5]

Una principio per scegliere queste unità è detto principio MECE (Mutually Exclusive and Collectively Exhaustive) che sceglie le unità in modo tali che siano esclusive tra loro e sommate le une con le altre riformino l'insieme di partenza.[10]

2. Collezione dei dati

   I dati possono essere collezionati da numerose fonti, da sensori, da interviste, parsing di siti web o da testi.

3. Data processing

   Infine i dati vengono processati e organizzati per l'analisi, ad esempio questo può voler dire inserire dati in una tabella (o in una base di dati) per facilitare le analisi successive.

4. Data cleaning

I dati raccolti possono essere incompleti, contenere duplicati o contenere errori. Il processo di pulizia dei dati si occupa di eleminare i dati in eccesso che potrebbero rendere l'analisi meno precisa.

5. Analisi iniziale

   I dati iniziano ad essere studiati con diversi algoritmi o modelli, potrebbe essere necessario tornare alla fase di pulizia o di raccolta in seguito a feedback non eccellenti. Una prima fase di visualizzazione (*Data visualization*) può anche essere usata per ottenere una migliore comprensione visiva riguarda al contenuto dei dati.

6. Data Product

   Un *Data product* è un applicativo che prende input e genera output basati sul processo di analisi sopraindicato.

2.2.1 Analisi sulla esperienza utente

Visto che l'obiettivo finale dell'analisi dei dati è quello di renderli comprensibili all'utente finale può essere utili ascoltare i feedback dell'utente sia per quanto riguarda la fasi dell'analisi che per quanto riguarda le fasi della visualizzazione (e di conseguenza dell'interrogazione al dataset).
Si possono raggruppare le necessità dell'utente in tre categorie di attività: ottenere unità della popolazione, trovare alcuni data point specifici[6] e organizzarli.[11]

Ottenere unità: Data una popolazione, trovare unità specifiche e relativi attributi.

[6]Per data point si intende un attributo specifico dell'unità

Filtri: Possibilità di filtrare i risultati ottenuti ponendo alcune condizioni sugli attributi.

Calcolare valori derivati: Dato un set di dati, trovare una rappresentazione numerica aggregata di questi dati. Per esempio la media, o la somma di più unità.

Trovare estremi: Trovare i casi limite. Per esempio quali sono le unità più grandi secondo qualche attributo.

Determinare intervalli: Dato un insieme di dati e un data point che ci interessa trovare l'intervallo in cui sono i compresi i valori dell'attributo di interesse.

Data visualization

Una volta analizzati i dati bisogna pensare a delle tecniche per aiutare a comunicare i risultati in modo chiaro ed efficace al pubblico. Nella visualizzazione dei dati spesso si usano tabelle, grafici per aiutare a comunicare alcune informazioni chiave analizzate.

2.3 Data Management e Conclusioni

Come abbiamo visto precedentemente una importante fase nell'analisi dei dati è il data processing, ovvero l'organizzazione di dati in strutture per ulteriori analisi. Nel campo dei big data i database relazionali non sono più ritenuti sufficienti per ragioni di scalabilità e di velocità.

Quindi è sempre maggiore l'impiego di basi di dati di tipo NoSQL[7], che spesso rinunciano ad affidabilità in cambio di prestazioni e scalabilità.

Nel prossimo capitolo discuteremo dei principale vantaggi nell'uso di un database non relazionale e in dettaglio quello che poi andremo a implementare nella nostra applicazione, un *graph database*.

[7] Database di nuova generazioni con queste caratteristiche fondamentali: non relazionali, distribuiti, open source, scalabili orizzontalmente. http://nosql-database.org/

Capitolo 3

Database NoSQL e GraphDB

Un database NoSQL[1] fornisce un meccanismo per lo storage e il reperimento dei dati senza usare relazioni di tabelle come nei database relazionali.

Questi tipo di database pur essendo già stati pensati negli anni ’70 hanno avuto un’esplosione di popolarità a partire dagli anni 2000 con la nascita di enormi compagnie come Facebook, Gooogle e Amazon.[12] Sono sempre più usati nel campo dei big data e nelle applicazioni web *real-time*. A volte sono detti "Not Only SQL" per enfatizzare che possono comunque supportare linguaggi di query simili all’SQL.[2]

3.1 Il passato: Database Relazionali

Uno dei più grandi contributi che l’informatica offre al genere umano è la memorizzazione di dati in maniera persistente.

[1] Espressione usata originalmente per indicare database "non SQL" o "non relazionali

[2] http://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL

Quotidianamente, immense quantità di informazioni vengono affidate a tecnologie che ne garantiscono la conservazione duratura ed un recupero efficiente che ne permetta l'analisi.
Da anni, questo ruolo viene interpretato molto bene da un una tipologia di database efficiente ed affidabile: i database relazionali.

3.1.1 Caratteristiche

Tra i DBMS[3] esistenti la tipologia più comune è quello dei cosiddetti RDBMS (Relational DBMS), ispirati dalla Teoria Relazionale.
Ad oggi è ancora il tipo di database più comune per moltissime casistice. Un database relazionale è costituito da tabelle, ognuna delle quali è composta da righe identificate da un codice univoco denominato chiave. Le tabelle che compongono il database non sono del tutto indipendenti tra loro ma relazionate da legami logici[4].
Una caratteristica fondamentale dei database relazionali è l'operazione di join [Figura 3.1] dove due tabelle sono messe in relazione tra loro tramite dei campi in comune.

[3] Database Management System o Sistema di gestione di basi di dati
[4] www.html.it/guide/guidamysql/

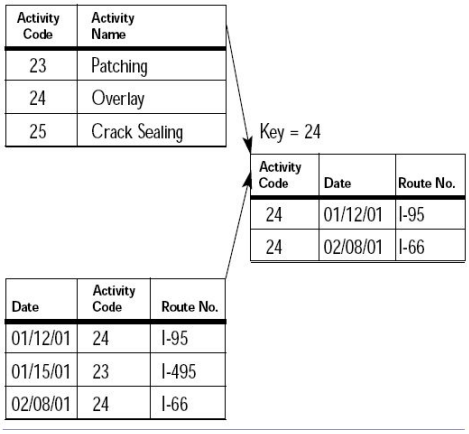

Figura 3.1: Esempio di Join

3.2 Differenze e vantaggi dei database NoSQL

Tuttavia questi database di tipo relazionale, pur rimanendo validi quando si devono gestire quantità di dati limitata, sono considerati inefficienti e non adatti quando si deve lavorare con grandi quantità di dati.

Molti DBMS che seguono il paradigma NoSQL rinunciano alla coerenza, come viene intesa del teorema CAP[5] a favore della disponibilità, della tolleranza di partizione e soprattutto della *velocità*.[13]

Questi database la sostituiscono con una "coerenza eventuale", dove i cambiamenti al database sono propagati agli altri "prima o poi". Le query perciò possono a volte non ritornare i dati aggiornati oppure

[5]Coerenza,Disponibilità, Tolleranza di partizione. Secondo il teorema di Brewer è impossibile ottenere tutte e tre in un sistema distribuito.

esserci delle perdite di dati in scrittura. Per risolvere questo inconveniente spesso si fa uso di tecniche *WAL* (write-ahead logging) dove tutte le modifiche sono scritte in un registro (log) prima di essere applicate.

Questo approccio facilità il design (non essendo limitati da una struttura a chiavi esterne) e soprattutto permette di scalare orizzontalmente (ovvero aggiungere più nodi a un sistema distribuito) caratteristica molto apprezzata a compagnie che devono lavorare con quantità enormi di dati.

Le strutture dati dei database NoSQL (che sono differenti a seconda del tipo di database scelto) come vedremo permettono di effettuare alcuni tipi di operazioni in modo molto più veloce e flessibile rispetto al corrispettivo relazionale.

3.3 Tipi ed esempi di Database NoSQL

In questa sezione elencheremo brevemente i vari tipi di database per poi focalizzarci in particolare quelli che useremo.

Chiave-Valore :

Un database di tipo Chiave-Valore immagazzina i dati in un array associativo (conosciuto anche come mappa o dizionario).

I dati sono rappresentati come una collezione di coppie chiave-valore in modo tale che ogni possibile chiave compre al massimo una volta nella collezione.

Questo modello può essere potenziato mantenendo le chiave in ordine lessicografico. In questo modo si possono ottenere range di chiavi in modo molto efficiente. [14]

Spesso questi database mantengono tutti i dati in memoria per ot-

tenere un efficienza ancora maggiore.

Documento :

Questo database si basa sul concetto di *"documento"*. Usualmente questi documenti contengono i dati in alcune codifiche standard, ad esempio XML o JSON.

I documenti sono organizzati nel database attraverso una chiave unica che li identifica.

Un'altra caratteristica tipica è quella di offrire delle API o un linguaggio di interrogazione dei documenti in base ai loro contenuti.

Collezioni :

Le collezioni sono raccolte di documenti analogamente a come in un database relazionale le tabelle sono raccolte di righe.

Tuttavia la differenza principale è che ogni riga in una tabella ha sempre la stessa sequenza di attributi mentre in una collezione i documenti possono anche essere completamente diversi.

Grafo :

È un tipo di database adatto a degli insieme di elementi con un numero finito di relazioni tra loro.

Nella prossima sezione analizzeremo le caratteristiche tipiche di questo tipo di database.

3.4 Grafi e GraphDB

3.4.1 Cos'è un grafo.

Un grafo può essere definito come un insieme di elementi detti nodi (o vertici) che possono essere collegati tra loro da linee chiamate archi (o lati).
Formalmente si può dire che un grafo è una coppia ordinata G(V,E) di insiemi, con V insieme di nodi e E insieme di archi.
Un grafo può essere di due tipologie [Figura 3.2]:

- Orientato:
  Gli archi hanno una propria direzione e un verso, solitamente indicato da una freccia. Questi grafi solitamente si utilizzano se si vuole determinare un certo percorso all'interno del grafo.

- Non Orientato:
  Gli archi non hanno un verso prestabilito, solitamente questi grafi servono se ci interessa solo determinare le connessioni riguardanti i nodi.

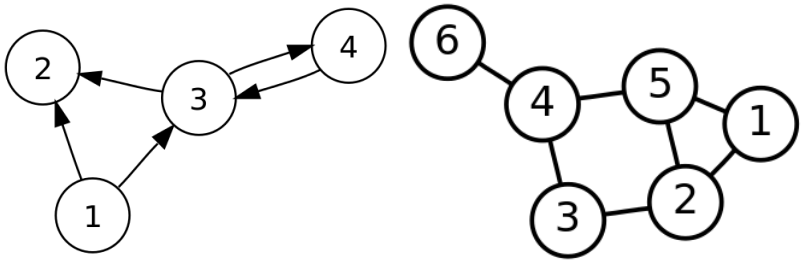

Figura 3.2: Differenze tra grafo orientato e non orientato

3.4.2 Database a grafo

Un database a grafo è un database che usa nodi e archi per rappresentare e archiviare i dati.
In termini generali possono essere considerati simili ai database chiave-valore con l'aggiunta del concetto delle relazioni tramite gli archi. Gli archi permettono un sistema di relazioni molto libero al contrario del modello relazionale dove le relazioni dovevano essere definite all'interno delle tabelle stesse. Quindi si possono formare gerarchie di nodi molto complesse che tuttavia possono essere attraversate molto velocemente permettendo prestazioni superiori alle join tipiche dei database relazionali.
Solitamente i nodi rappresentano le entità di cui si vuole tener traccia, come ad esempio persone e gruppi; i nodi possono avere proprietà come ad esempio l'età ed il nome. Anche gli archi possono avere proprietà che caratterizzano la relazione, ad esempio due persone possono esser collegate tra loro per il fatto di conoscersi, e nell'arco si potrebbe indicare da quanto tempo si conoscono. [Figura 3.3].

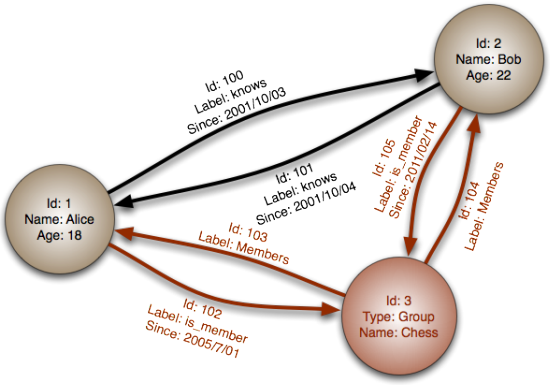

Figura 3.3: Esempio di relazioni in un Graph DB.

3.5 Conclusioni

Le alternative ai database relazionali esistono da molti anni ma solamente negli ultimi anni sta aumentando l'interesse per le tecnologie NoSQL. Sempre più aziende come Google, Amazon, Facebook e Twitter hanno iniziato ad usare soluzioni non relazionali nei loro prodotti.
Cosa spinge queste aziende a cambiare il loro modo di gestire dati? Principalmente la grande mole di dati che si sono ritrovate a gestire,i cosidetti Big Data.

Migrare verso un database NoSQL solitamente presenta dei grossi vantaggi:

- Prestazioni
- Scalabilità
- Maggiore Flessibilità

Nel prossimo capitolo illustreremo quale database è stato scelto per la nostra applicazione.

Capitolo 4

Strumenti usati

Per poter realizzare il progetto di analizzare lo storico delle transazioni Bitcoin abbiamo realizzato una *Web Application* attraverso l'uso del database OrientDB, PHP, Bitcore per la parte backend e HTML5, CSS3 e Javascript per la parte frontend. [1]

4.1 Parte Back end

4.1.1 OrientDB

OrientDB è un database ibrido, infatti possiede sia le caratteristiche di un database a grafo che quelle di uno basato sui documenti. È un progetto italiano scritto in Java sviluppato 6 anni fa da Luca Garulli. Ad oggi è il secondo Graph Database per utilizzo dietro al colosso Neo4j. [2]

[1] Il front end è la parte di un software che gestisce l'interazione con l'utente mentre il back end è la parte che elabora i dati generati dal front end. https://it.wikipedia.org/wiki/Front-end_e_back-end

[2] http://db-engines.com/en/ranking/graph+dbms

È stato scelto principalmente per la sua velocità: In lettura può leggere oltre 400'000 record al secondo, e inserirne oltre 120'000 [3]. Per attraversare le relazioni invece di usare costose join usa riferimenti molto veloci ad altri record, in pochi millisecondi si possono attraversare interi grafi. Supporta diversi linguaggi di query tra cui uno che è basato su SQL che ha semplificato molto da parte nostra l'apprendimento.

Fondamentale per la nostra applicazione era la possibilità di contenere un grosso numero di elementi. Un database di OrientDB può contenere fino a 2^{78}-1 record ma soprattutto la velocità di visita di un grafo è costante, sia se occorre attraversare pochi nodi che alcuni miliardi! Questo è fondamentale se si deve lavorare con Big Data!

Un'alternativa che era stata valutata precedentemente era Neo4j ma è stato abbandonato soprattutto per la mancanza di alcune funzionalità della versione gratuita e per evitare di dover imparare un nuovo linguaggio di query (Neo4j si basa su Cypher [Figura 4.1].

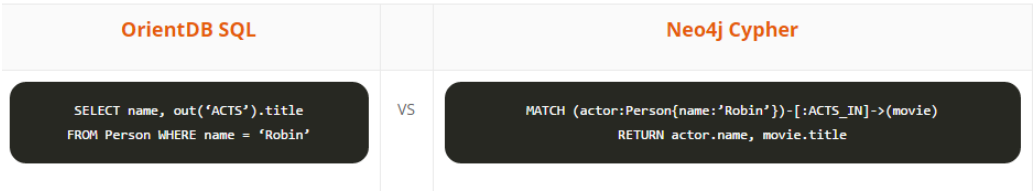

Figura 4.1: Differenza della stessa query fra Neo4j e OrientDB.

[3] http://orientdb.com/why-orientdb/

4.1.2 PHP

Ogni operazione eseguita sul Web coinvolge un client ed un server. Un client è un dispositivo, esempio un browser che effettua una richiesta ad un server remoto. Il server remoto attraverso un linguaggio di scripting (come per esempio PHP[4]) interpreta la richiesta del client ed invia una risposta al client.
PHP è un linguaggio per lo scripting server-side, ovvero un linguaggio che risiede in un server in remoto e che in fase di esecuzione interpreta le informazioni ricevute da un client grazie al Web server, le elabora e restituisce un risultato al client che ha formulato la richiesta[5].

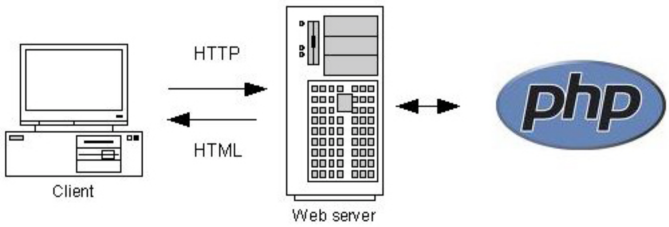

Figura 4.2: Processo di funzionamento php

Nella nostra applicazione usiamo PHP principalmente per interfacciarci con il database OrientDB per inserire o ottenere dati riguardanti le transazioni Bitcoin.

[4]Acronimo ricorsivo che sta per PHP: Hypertext Preprocessor
[5]www.html.it/guide/guidaphpdibase/

4.1.3 JSON

Come abbiamo già detto per scambiare dati tra il database e il client usiamo la codifica JSON [Fig 4.3]. JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati.
Un oggetto json è una serie non ordinata di nomi/valori. Un oggetto inizia con (parentesi graffa sinistra) e finisce con (parentesi graffe destra). Ogni nome è seguito dai due punti e la coppia di nome/valore è separata da una virgola.

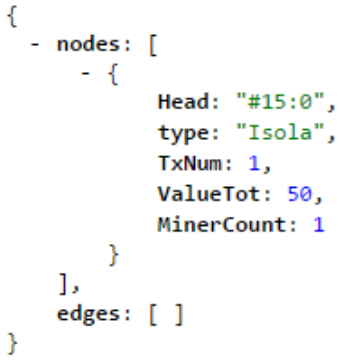

Figura 4.3: Un oggetto JSON usato nella nostra applicazione.

4.1.4 Bitcore

Caratteristica fondamentale della nostra applicazione è la possibilità di avere a disposizione l'intero storico delle transazioni Bitcoin.
Inizialmente per ottenere questi dati si era pensato di sfruttare servizi WEB che mettono a disposizione la loro copia della blockchain ad interrogazioni online, i cosidetti *blockexplorer*. Un blockexplorer è un sito web che permette di ricevere informazioni su transazioni che sono presenti nella blockchain indicandone gli input e gli output (il json

usato nel primo capitolo per spiegare la struttura di una transazione ad esempio era preso da *https://blockchain.info*).
Tuttavia questi servizi non sono adatti ad fare importazioni di massa della blockchain, infatti hanno dei limiti per le richieste che sono facilmente raggiungibili se interrogati sistematicamente via software e inoltre a discrezione del fornitore del servizio omettono alcune informazioni che invece possono essere di nostro interesse.
Abbiamo deciso dunque che la soluzione migliore fosse quella di avere una copia della blockchain nel server per poter caricare facilmente i dati della blockchain nel database.

Bitcore è un full client Bitcoin [6] che nasce con l'obiettivo di rendere più facile integrare la proprio applicazione con la tecnologia Bitcoin.
Usare direttamente il network peer to peer per la propria applicazione permette di avere prestazioni di ordini di grandezza migliori rispetto all'uso di API centralizzate. [7]
Si basa su *bitcoind* versione a linea di comando del client ufficiale bitcoin a cui poi è collegato Insight API, web service REST[8] che permette di ricevere informazioni riguardanti la blockchain in formato JSON.

[6] Come abbiamo visto nel primo capitolo un full client è un client che per funzionare ha bisogno di scaricare l'intera blockchain

[7] https://bitcore.io/

[8] Standard di progettazione di un sistema, molto popolare per la creazione di API, http://www.html.it/pag/19596/i-principi-dellarchitettura-restful/

4.2 Node.js

Per alcune query ci siamo dovuti servire del driver di OrientDB per Node.js poichè migliore dal punto di vista prestazionale rispetto a quello per php. Abbiamo dovuto quindi creare un servizio web basato su Node.js.

Node.js è un framework opensource multipiattaforma per sviluppare applicazioni web a lato server usando javascript. Sta diventando oramai l'alternativa più usata al php per la maggiore semplicità e velocità. Node.js ha un architettura basata sugli eventi e sulla gestione degli input e degli output in modo asincrono. Questa scelta di design punta a migliorare le prestazioni e la scalabilità dell'applicazione in presenza di molte connessioni simultanee.[9]

Uno dei punti di forza di questo ambiente di sviluppo sono i moduli: lo sviluppatore può facilmente espandere le funzionalità della propria applicazione usando delle librerie che sono raccolte e aggiornate usando un gestore di questi moduli chiamato npm[10]. Al momento ci sono oltre 250mila moduli che si possono includere nella propria applicazione risparmiando molto tempo non dovendo implementare funzioni standard.[11]

Ad esempio nel nostro server usiamo *hapi.js*[12] per avere già disponibile un insieme di funzioni per creare un servizio web che risponde a determinate richieste.

[9] https://en.wikipedia.org/wiki/Node.js

[10] https://www.npmjs.com/about

[11] http://www.modulecounts.com/

[12] http://hapijs.com/

4.3 Parte Front End

4.3.1 HTML

L'HTML è un linguaggio di markup che viene interpretato dai browser e che permette di determinare la formattazione dei contenuti della pagina web.
La sua sintassi viene stabilita dal World Wide Web Consortium (W3C) ed ha come componente principale l'elemento, inteso come struttura di base a cui è delegata la funzione di formattare i dati o indicare al browser delle informazioni.
Ogni elemento è racchiuso all'interno di marcature dette tag, costituite da una sequenza di caratteri racchiusa tra due parentesi angolari (<element> : per l'apertura del tag e </element> : per la chiusura).
Il documento HTML presenta una struttura ad albero annidato, composta da sezioni delimitate da tag opportuni che al loro interno contengono a loro volta sottosezioni più piccole, sempre delimitate da tag.
La struttura generale di una qualsiasi pagina HTML consiste nella definizione di un DOCTYPE, che specifica il tipo del documento, seguito da un tag <html> che a sua volta contiene il tag <head> e il tag <body>.
Nel tag <head> vengono specificate alcune informazioni di servizio per il browser ed il riferimento ad eventuali file esterni quali fogli di stile(CSS) o script (Javascipt). Il tag <body> contiene la pagina vera e propria.

4.3.2 CSS

Un foglio di stile CSS è un file a parte che, collegato all'HTML, permette al browser di identificare l'aspetto grafico che dovrà avere il sito.

Il CSS (Cascading Style Sheets) è un linguaggio usato per definire lo stile di documenti HTML, XHTML e XML. Le regole per comporre il CSS sono contenute in un insieme di direttive (Recommendations) emanate a partire dal W3C. Il codice CSS da inserire è strutturato sotto forma di una o più regole in forma di istruzioni del tipo *proprietà : valore* che vengono applicati dal browser in fase di rendering agli elementi HTML interessati,opportunamente scelti tramite un selettore. [Figura 4.4]

```
selettore{
  proprietà1:valore1;
  proprietà2:valore2,valore3;
}
```

Figura 4.4: Struttura css.

4.3.3 Javascript

JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso (mouse, tastiera, ecc...). Usato insieme ad Ajax permette al client di interfacciarsi facilmente al database.

4.3.4 Ajax

AJAX, acronimo di Asynchronous JavaScript e XML, è una tecnica per sviluppare applicazioni altamente dinamiche fondendo insieme le

caratteristiche di HTML e JavaScript utilizzando l'oggetto XMLHttpRequest per la manipolazione dei dati.
La novità di AJAX rispetto ad altre tecniche è che consente alla pagina di interagire con risorse esterne, ad esempio database o file XML, incorporando il risultato dell'elaborazione direttamente nella pagina stessa senza che sia necessario ricaricarla. Questa caratteristica migliora l'interazione dell'utente con il sito ed incrementa la velocità di caricamento, consentendo di realizzare applicazioni web molto simili a strumenti software per desktop. [13]

[13] http://www.rss-world.info/glossario/a/ajax.php

Capitolo 5

Web Application

5.1 Descrizione del progetto

Il progetto realizzato ha come obiettivo finale la visualizzazione di un grande quantitativo di dati riguardanti l'insieme delle transazioni contenute nella blockchain.

Prima di tutto occorre pensare a come strutturare il database partendo dalle conoscenze apprese sul funzionamento dei bitcoin, poi creare un software per importare i dati dei vari blocchi all'interno del database. Infine creare delle API che, dati dei parametri, interrogano il database e forniscono la risposta sotto forma di JSON.

5.2 Struttura Dati Scelta

OrientDB permette di organizzare i nodi e gli archi per classi dando a ciascuna classe una lista di attributi e vincoli.
I vincoli di "chiave" si possono specificare creando degli indici che velocizzano le query e specificando se il campo selezionato è univo-

co(*UNIQUE*) o può comparire in più oggetti dello stesso tipo.
Come abbiamo visto nel primo capitolo la blockchain non è altro che un insieme di blocchi che contengono a loro volta delle transazioni. Per analizzare i flussi di denaro non tutti i blocchi sono necessari ma solamente quelli della catena principale(la catena più lunga) , i blocchi orfani infatti contengono solamente transazioni non convalidate o doppioni delle transazioni che ci interessano. [Figura 5.1]

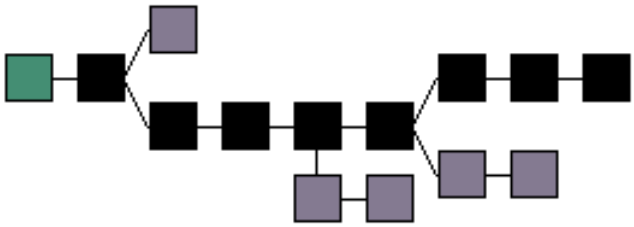

Figura 5.1: Schema della blockchain, in verde il blocco genesi, in nero la catena principale, in viola i blocchi orfani.

Appare evidente dunque che due entità che ci interessano sono i blocchi e le transazioni.
La classe Block tuttavia durante il progetto è stata rimossa per semplificare lo schema ed è diventata solamente un attributo all'interno della classe Transaction.

5.2.1 Transaction

La classe Transaction è una sottoclasse della classe V.(Vertex, classe nativa di OrientDB per indicare i nodi) Solitamente le transazioni sono univocamente distinguibili da un hash sha256 calcolato a partire dalla

transazione scritta in formato binario (esempio a pagina 13). Questo **hash** sarà quindi la nostra chiave primaria.

Altri attributi che fanno parte della classe sono:

Height: Indica l'altezza del blocco in cui si trova la transazione

Miner: Indica se la transazione è una transazione che ha generato monete (detta *coinbase*)

Time: Indica la data e l'ora in cui la transazione è stata aggiunta nel blocco

Tx_fee: Indica la commissione pagata dagli input per convalidare la transazione

valueTot: Indica il valore totale della transazione sommando tutti gli input.

È importante evidenziare che è importante che gli attributi in cui vengono salvati i valori riferiti a Bitcoin (Tx_Fee e valueTot) debbano mantenere un certo livello di precisione perché l'unità più piccola di Bitcoin esprimibile è 1 satoshi ovvero 10^{-8} ₿.[1]

5.2.2 Address

I protagonisti di una transazione sono gli indirizzi che si scambiano la proprietà dei Bitcoin. Anche gli indirizzi saranno dei nodi all'interno del nostro schema. In una transazione ci possono essere un qualsiasi numero di input e di output e come abbiamo visto lo stesso indirizzo può essere sia un input che un output per il meccanismo del *resto*.

[1] Ovvero 0.00000001 ₿, https://en.bitcoin.it/wiki/Units

Quindi l'unico attributo che può veramente avere un indirizzo è l'**hash** univoco che come abbiamo visto non è altro che la chiave pubblica scritta in forma compressa.

Se una persona rende pubblico il suo legame ad un indirizzo(ad esempio firmando volontariamente un documento usando la stessa chiave) è possibile associare a quell'indirizzo un nome, quindi aggiungeremo un campo facoltativo "name" per poter tenere traccia di eventuali indirizzi resi pubblici.

Se si collegano più indirizzi alla stessa persona è possibile definire un wallet che contiene più indirizzi dello stesso proprietario.[2]

5.2.3 ValueTx

ValueTx invece è una classe riferita ad archi e quindi estende la classe E (Edges, classe nativa di OrientDB per indicare gli archi). Mediante questi archi collegheremo i nodi Address ai nodi Transaction. Useremo archi *entranti* al nodo Transaction per indicare gli input e archi *uscenti* per indicare gli output. Gli attributi di questa classe sono:

Value: Indica il valore trasferito o ricevuto dall'indirizzo mediante la transazione.

TxIndex e Vout: Nel caso di un input questi due valori indicano la transazione di provenienza dei bitcoin (TxIndex) e il numero di output di quella transazione (Vout). Nel caso di un output indica i valori della transazione da cui esce l'arco.

[2] https://www.walletexplorer.com/

5.2.4 InvalidOutputScript

Non sempre un output corrisponde a un indirizzo, come abbiamo visto mediante l'istruzione **OP_RETURN** è possibile lasciare un messaggio collegato a una transazione. La classe InvalidOutputScript è un altro arco, entrante ed uscente alla transazione, che si limita a salvare lo script lasciato per avere la possibilità in seguito di decriptare il messaggio.

5.2.5 Riepilogo

Nella figura 5.2 troviamo riassunto lo schema che useremo per immagazzinare i dati della blockchain. Le entità in blu corrispondono ai nodi mentre quelle in rosso agli archi, gli attributi indicati in grassetto indicano chiavi univoche nei nodi.

5.3 Inserimento Dati

Dopo aver fatto uno schema non resta che inserire i dati della blockchain. Di questa operazione si occupa uno programma PHP, getBlock.php.

5.3.1 Script GetBlock

Questo programma prende come input un range di altezze di blocchi e come risultato aggiunge nel database tutte le transazioni contenute in quei blocchi. Possiamo riassumere l'esecuzione del programma in queste fasi.

1. Scarico i dati di un blocco:

   Richiamando delle API collegate alla blockchain il programma sca-

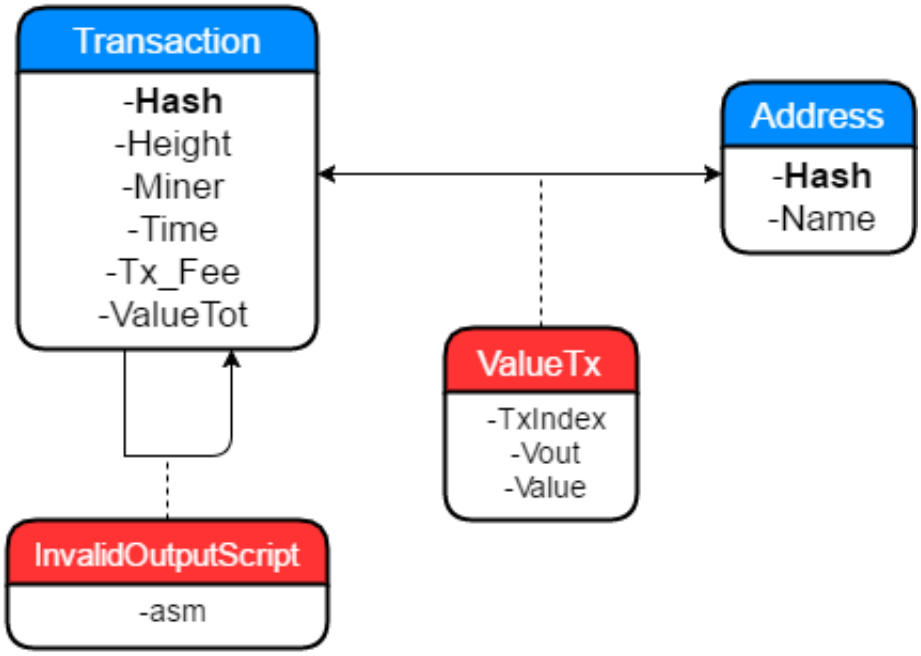

Figura 5.2: Riepilogo struttura usata nel database.

rica un array JSON contenente tutte le transazioni del blocco e le inserisce in un oggetto php per leggerle in seguito.

2. Inserisco la transazione Coinbase.
   La prima transazione di ogni blocco è la transazione che genera valuta, inserisco l'indirizzo che ha validato il blocco, se questo esiste già ne prendo solamente il riferimento e lo collego come arco uscente di tipo ValueTx (con i corrispondenti valori) al nodo transazione appena creato (con l'attributo miner settato a *true*)

3. Scorro tutte le altre transazioni e le inserisco.
   Analogamente inserisco tutte le altre transazioni, sempre controllando se l'indirizzo inserito esiste già.

4. Stampo un eventuale messaggio di successo o errore per il blocco.

Queste fasi ovviamente si ripeteranno per tutto il range di blocchi indicato. La prima fase di questo processo è il principale collo di bottiglia che abbiamo incontrato: le api pubbliche non sempre sono affidabili, alcune non avevano tutti i dati che ci interessavamo mentre altre vietavano l'accesso al servizio superato un certo limite di richieste.
Dopo aver installato il nostro nodo Bitcoin (Bitcore) nel server si è trattato semplicemente di modificare la prima parte del sistema per poter inserire i dati riguardanti le transazioni. Ovviamente in questo modo è necessario che Bitcore sia attivo quando si vuole inserire un nuovo blocco.

5.4 Parte Back end

Per facilitare l'uso dell'applicativo abbiamo creato un pannello di gestione protetto da password per evitare l'uso da parte di utenti non autorizzati.

5.4.1 Riepilogo Database

La prima sezione del sito è dedicata a riepilogare lo stato delle varie componenti della nostra applicazione poiché c'è la necessita che tutti siano attivi per un uso ottimale dell'applicazione. Come vediamo nella figura 5.3 visualizziamo lo status dei tre servizi web della nostra applicazione OrientDB, Bitcore ed il server Node.js usato per le query con la possibilità di fermarli o avviarli.
Nella tabella sottostante invece viene confrontato il livello di aggiornamento del nostro database nei rispetto al nostro nodo Bitcoin e rispetto alla rete Bitcoin globale.

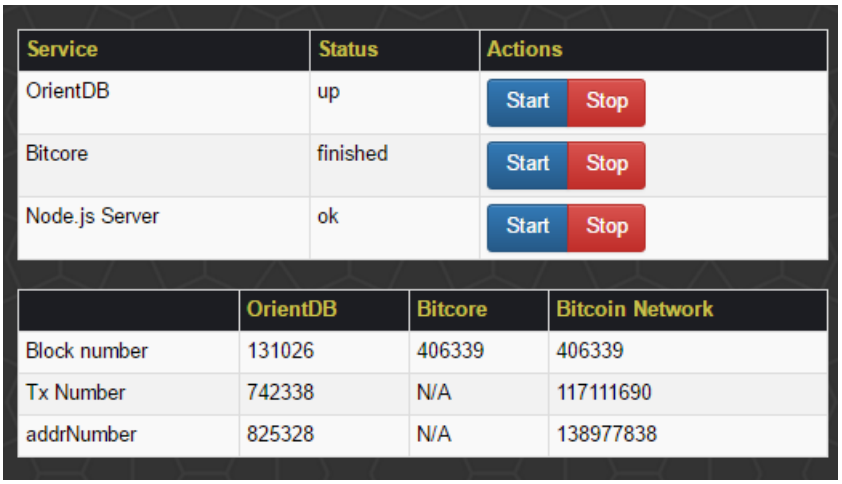

Figura 5.3: Informazioni sullo stato dell'applicazione.

5.4.2 Scarica Blocchi

Questa sezione è l'interfaccia del programma php getBlock visto precedentemente [Figura 5.4. Quando viene aperta la pagina viene fatta una richiesta AJAX per scaricare l'intervallo di blocchi presenti nel database e vengono modificati gli input in modo tale che non si possano indicare intervalli di blocchi non validi.

Una volta premuto il tasto Download per dare un feedback all'utente sullo stato di avanzamento dell'inserimento delle transazioni nel database, all'interno dello programma php alla fine di ogni blocco inserito aggiorniamo un file json. In questo file vengono scritti i blocchi scaricati sul totale, il tempo impiegato e una stima del tempo necessario rimanente.

Questo file viene continuamente consultato in modo sincrono circa ogni secondo e la barra che indica il progresso e le informazioni sul processo vengono aggiornate. Questa tecnica di aggiornamento che presuppone

la consultazione continua di una risorsa esterna è chiamata *polling*[3].

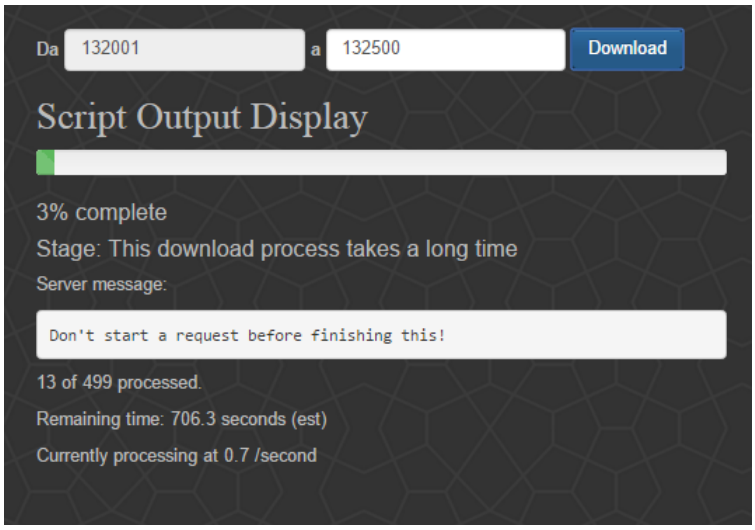

Figura 5.4: Interfaccia per il download dei blocchi.

5.4.3 Aggiungi nomi ad indirizzi

Come abbiamo visto nella descrizione dello schema, nella classe *Address* abbiamo previsto il campo 'Name' per poter, se possibile, identificare il proprietario dell'indirizzo bitcoin.

Abbiamo quindi creato una pagina nel pannello di gestione dove è possibile aggiornare il database inserendo questi nomi. Questa pagina prende come input un file di tipo CSV[4] contenente per ogni riga una coppia indirizzo-nome separata da una virgola.

[3]https://en.wikipedia.org/wiki/Polling_(computer_science)

[4]Comma-separated values è un tipo di file di testo usato per l'esportazione e importazione di tabelle di dati https://it.wikipedia.org/wiki/Comma-separated_values

L'utente per aggiungere i nomi non deve fare altro che trascinare i file CSV all'interno del rettangolo [Figura 5.5] o selezionarli cliccando su Browse. I file verranno caricati sul server e uno script php provvederà ad aggiungere i nomi ai nodi indirizzo corrispondenti nel database.

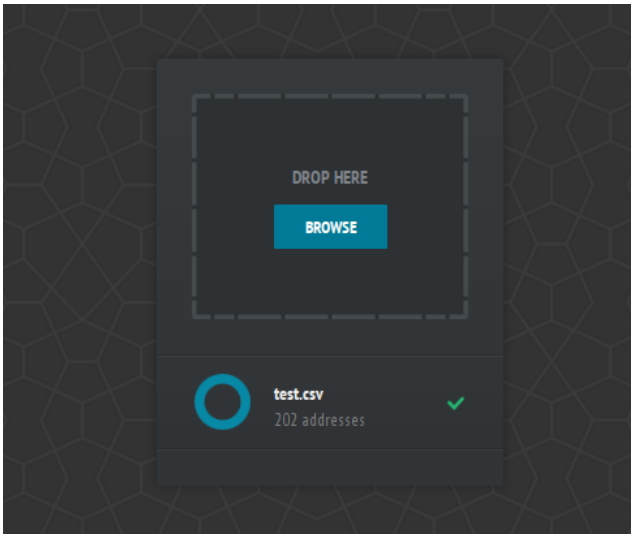

Figura 5.5: Interfaccia per l'aggiunta dei nomi agli indirizzi del database.

5.4.4 Svuota database

È possibile infine svuotare l'intero database, ovviamente questa operazione è molto delicata ed principalmente utile in fase di sviluppo dell'applicazione, e va usata con attenzione perciò viene aperta una finestra di conferma [Figura 5.6] per essere sicuri dell'intenzione dell'utente.

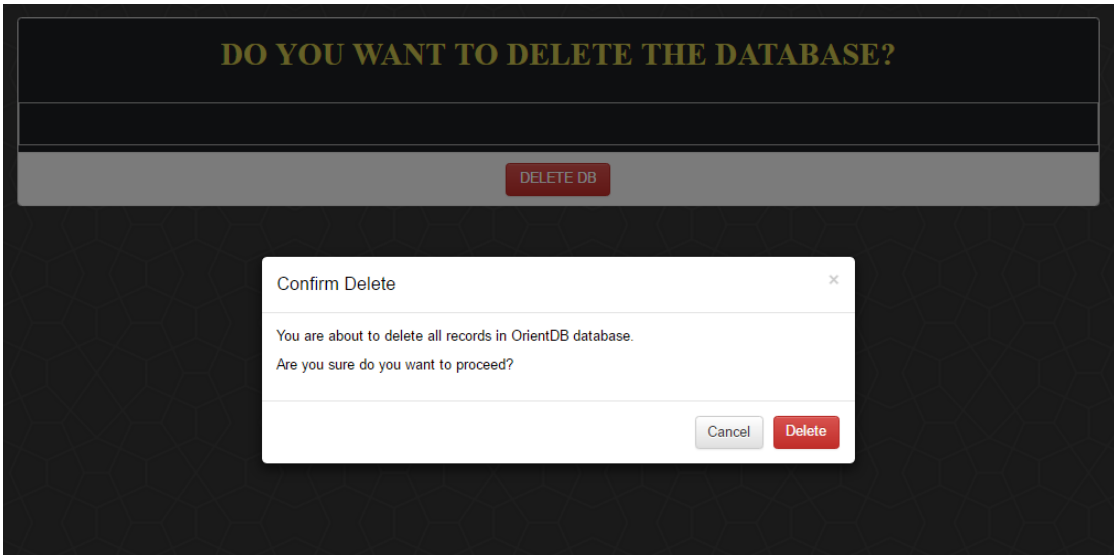

Figura 5.6: Interfaccia per l'eliminazione del database.

5.5 Query interessanti

Una volta inseriti i dati questi sono inutili se non c'è la possibilità di interrogare il database per eventuali risultati.
Data la grande mole di dati è inutile ritornare tutte le transazioni ma bisogna trovare dei criteri per trovare dei sottoinsiemi interessanti di dati.

5.5.1 Cosa sono le isole

Abbiamo individuato uno di questi sottoinsiemi nelle *Isole*. Partendo dal grafo di tutte le transazioni, un'Isola è una componente connessa del grafo in un determinato intervallo di tempo.
Una componente connessa in un grafo è un sottografo dove:

- Qualsiasi coppia di vertici è connessa da cammini

- Il sottografo non è connesso a nessun vertice addizionale del supergrafo.

Detto in altre parole dato un intervallo di tempo un'Isola è un sottografo dove le transazioni sono tutte collegate tra loro da indirizzi in comune. [Figura 5.7]

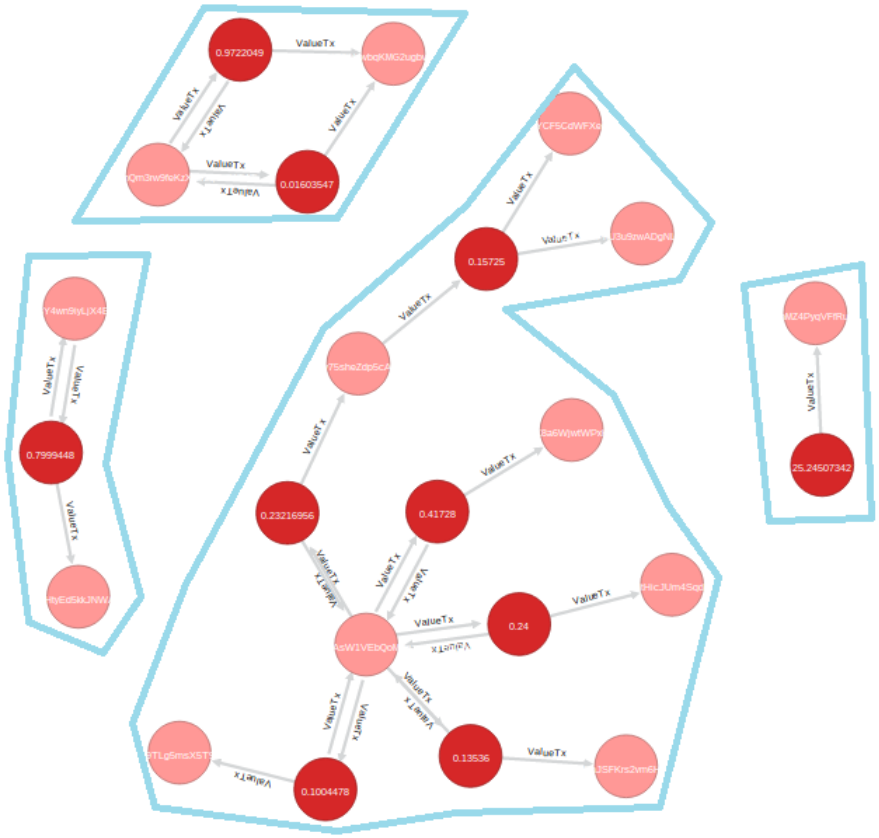

Figura 5.7: Un insieme di tre isole (cerchiate in azzurro). I nodi rossi rappresentano le transazioni, quelli rosa gli indirizzi.

5.5.2 Elenco Isole

La query per trovare queste isole non è banale, infatti trovare un semplice elenco di queste isole sarebbe piuttosto inutile, invece se a questo elenco è accompagnato anche il numero di transazioni, il numero di miner e il valore totale di Bitcoin scambiati è più semplice successivamente selezionare l'isola di nostro interesse.

```
1 select head, sum(valueTot) as valueTot, count(*) as txCount,
    sum(miner) as minerCount from
2       (select *,traversedElement(0) as head from
3       (TRAVERSE both('ValueTx') from
4       (select * from Transaction where height >= $begin and
           height <= $end )
5       while ( @class = 'Address' or (@class = 'Transaction' and
           height >= '$begin' and height <= '$end' )) )
6        where @class = 'Transaction') group by head
```

Questa query SQL è composta da più query una dentro l'altra e quindi per poterla spiegare è necessario partire dalla query più interna:
Prima di tutto (riga 4) seleziono tutte le transazioni comprese nell'intervallo di altezze dato dalle variabii $begin ed $end.
Nella query successiva (riga 3) dalle transazioni ritornate faccio una visita (*TRAVERSE*[5]) attraversando gli archi sia entranti che uscenti (*both*) di tipo "ValueTx".
Di default la visita fatta attraverso TRAVERSE è una DFS[6] quindi pos-

[5]Una delle keyword specifiche del dialetto SQL di orientDB http://orientdb.com/docs/last/SQL-Traverse.html

[6]Depth-first search, Ricerca in profondità è algoritmo di ricerca su alberi e grafi. https://en.wikipedia.org/wiki/Depth-first_search

siamo prendere il primo nodo(nella riga 2, traversedElement(0)) come rappresentante di una singola isola chiamandolo *head* (testa[7]). La condizione *while* (riga 5) riferita all'operazione di DFS indica le condizioni per la quali una visita deve continuare. In questo caso visto che ci interessano solamente le transazioni in un intervallo di tempo indicheremo di continuare la visita quando incontriamo un indirizzo (attraverso il campo *@class* è possibile individuare la classe di un nodo durante la visita) o quando la transazione incontrata è compresa nell'intervallo richiesto.

Insieme alla testa di ciascuna isola selezioniamo comunque tutte le transazioni (la condizione "*where @class='Transaction'*" serve a filtrare gli indirizzi visitati) per poter usarne i valori nella query più esterna. Infatti nella selezione finale (riga 1) oltre alla testa sommiamo anche i valori delle transazioni (*valueTot*), contiamo il numero delle transazioni (*txCount*) e il numero di miner presenti nell'isola (*minerCount*) raggruppando le somme per la testa (*group by*).

Il risultato della query viene, tramite PHP, convertito in formato JSON per essere visualizzato dal client in seguito a una richiesta. Ecco un esempio di risultato di questa query:

[7] Abbiamo scelto il termine "testa" per indicare un elemento da cui partire per una successiva visita, come fosse una lista circolare

head	valueTot	txCount	minerCount
#15:871	30849.579652270004	631	0
#15:1895	779.7178867599999	71	0
#15:876	9.123431909999997	67	0
#15:945	0.10252834000000002	44	0
#15:977	303.65301483	28	0
#15:860	235.56737467000002	23	0
#15:879	3.12215591	20	0

Figura 5.8: Elenco (incompleto) di isole comprese tra il blocco 404925 e il blocco 404927 ordinate per numero di transazioni.

5.5.3 Visita di una singola Isola

Come conseguenza della query precedente, serve un'interrogazione che restituisca tutti i nodi e archi appartenenti ad un isola.
Tramite PHP successivamente elaboreremo questi dati ritornati per ottenere un JSON adatto.

```
1 select @rid AS id_tx , hash ,miner,time, valueTot AS
    TransactionValue ,tx_fee AS TXFee, in('ValueTx').@rid AS
    rid_input,in('ValueTx').hash AS InputAddress
    ,inE('ValueTx').Value AS InputValue,inE('ValueTx').TxIndex AS
    InputTx, inE('ValueTx').Vout AS InputTxVout,
    out('ValueTx').@rid AS rid_output, out('ValueTx').hash AS
    OutputAddress, outE('ValueTx').Value AS
    OutputValue,outE('ValueTx').TxIndex AS OutputTx,
    outE('ValueTx').Vout AS OutputTxVout from
```

```
2 (traverse both('ValueTx') from '$head' while ( @class =
    'Address' or (@class = 'Transaction' and height >= '$begin'
    and height <= '$end' ) ) where @class = 'Transaction')
```

Pur essendo lunga questa query in realtà è abbastanza semplice.
Nella query più interna (riga 2) faccio una visita specificando questa volta un punto di partenza (la testa dell'isola che abbiamo scelto di visitare) e la stessa condizione *while* vista nella query precedente.
Nella query all'esterno seleziono dalle transazioni ritornate tutte le informazioni che mi interessano riguardante ogni singola transazione: l'hash, se è un miner, il tempo della transazione, il valore totale e tutte le informazioni riguardanti gli indirizzi di input e output.
In questo caso l'elaborazione del JSON è più complessa poiché sono coinvolti nodi e archi doppioni. Infatti se un indirizzo partecipa a due transazioni nel risultato che ho ottenuto apparirà due volte ma a noi interessa visualizzarlo una volta sola.
Allo stesso modo come abbiamo visto nel primo capitolo un indirizzo può usare più input per raggiungere la cifra di Bitcoin che vuole spendere, e quindi nel database ci potrebbero più archi che collegano lo stesso nodo mentre a noi interessa visualizzarne uno solo.
Nella funzione PHP impiego degli array chiave-valore per gli archi e i nodi in modo tale da non avere doppioni. Durante questa operazione calcolo anche il bilancio dei nodi Address sommando a una variabile "balance" quando ricevono soldi e sottraendo quando invece sono input.
Quando invece incontro due archi che hanno la stessa origine e destinazione concateno le informazioni dei doppioni in un unico arco per non perdere informazioni. Infine creo un oggetto JSON con all'interno un

array di nodi e uno di archi pronto ad essere visualizzato.[Figura 5.9]


```
{
    tx_number: 1,
    valueIsola: "25.03658182",
  - nodes: [
      - {
            id: "#15:859",
            type: "Transaction",
            hash: "d2fcce21951e98f70168c1365f58fa011f1af54e815f3373011cddeb3d1dd7ed",
            miner: 1,
          - time: {
                date: "2016-03-30 03:10:30.000000",
                timezone_type: 1,
                timezone: "+00:00"
            },
            Value: 25.03658182,
            Fee: 0,
            NumInput: 0,
            NumOutput: 1
        },
      - {
            id: "#16:6052",
            hash: "1CK6KHY6MHgYvmRQ4PAafKYDrg1ejbH1cE",
            type: "Address",
            Balance: "25.03658182",
            NumIn: 1,
            NumOut: 0
        }
    ],
  - edges: [
      - {
            source: "#15:859",
            target: "#16:6052",
            type: "Output",
            ValueTx: 25.03658182,
            Tx: "#15:859",
            OriginTx: "d2fcce21951e98f70168c1365f58fa011f1af54e815f3373011cddeb3d1dd7ed",
            InputTxVout: 0
        }
    ]
}
```


Figura 5.9: Oggetto JSON che rappresenta un’isola composta solamente da un miner.

5.5.4 Transazioni di un indirizzo

```
1 select @rid AS id_tx , hash ,miner,time, valueTot AS
    TransactionValue ,tx_fee AS TXFee, in('ValueTx').@rid AS
    rid_input,in('ValueTx').hash AS InputAddress
    ,inE('ValueTx').Value AS InputValue,inE('ValueTx').TxIndex AS
    InputTx, inE('ValueTx').Vout AS InputTxVout,
    out('ValueTx').@rid AS rid_output, out('ValueTx').hash AS
    OutputAddress, outE('ValueTx').Value AS
    OutputValue,outE('ValueTx').TxIndex AS OutputTx,
    outE('ValueTx').Vout AS OutputTxVout from
2 (TRAVERSE both('ValueTx') FROM (select * from address where hash
    = '$hash') WHILE $depth <= 2 STRATEGY BREADTH_FIRST) where
    @class = 'Transaction'
```

La prima parte (riga 1) di questa query è identica alla precedente, nella seconda parte (riga 2) faccio una visita a partire da un certo indirizzo. Questa visita a differenza delle altre è per ampiezza (BFS [8]) e oltre al nodo di partenza specifico anche la profondità (*depth*) che voglio raggiungere.

Nel nostro caso specificando 2, visitiamo per ciascun arco collegato al nostro indirizzo la transazione corrispondente (profondità 1) e tutti gli indirizzi collegati a quella transazione (profondità 2).

Ecco un esempio di un grafo ricavato da questa query disegnato attraverso l'interfaccia web di OrientDB. [Figura 5.10]

[8] Breadth-first search, https://en.wikipedia.org/wiki/Breadth-first_search

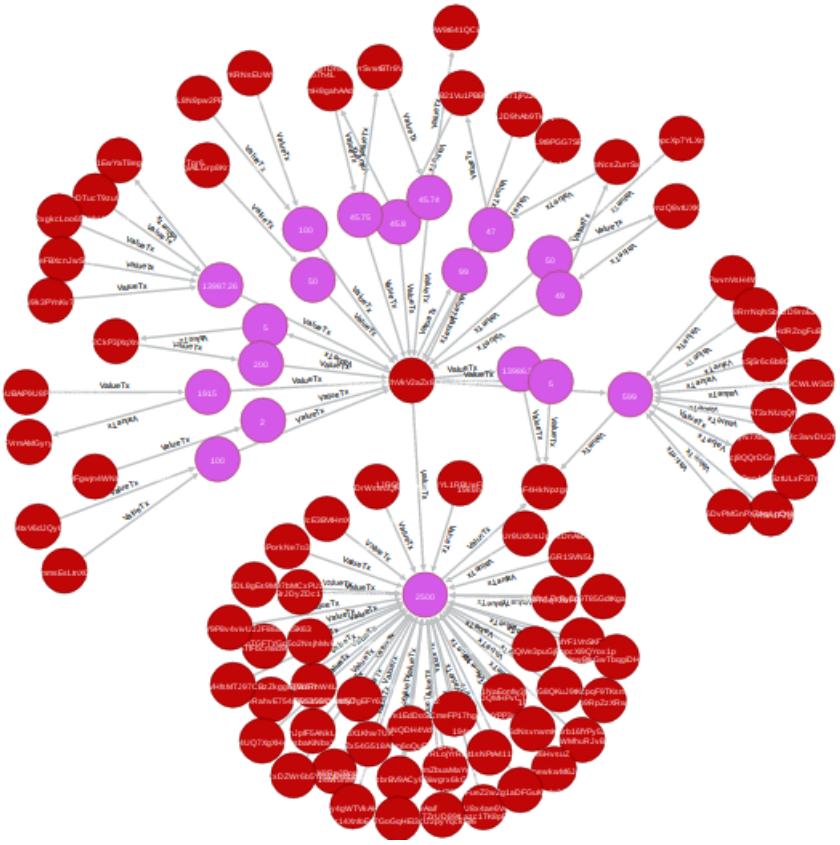

Figura 5.10: Esempio di risultato di una query per indirizzo. I nodi rossi sono gli indirizzi, in particolare quello al centro è l'indirizzo per cui è stata fatta la query, i nodi viola sono invece le transazioni

5.6 Riepilogo

Nella figura 5.11 viene riassunto il processo che ci ha permesso di caricare i dati della blockchain all'interno di un database a grafo.

1. Download dell'intera blockchain mediante il full client Bitcore.
2. I dati relativi alla transazioni vengono letti dallo programma PHP GetBlock e inseriti in OrientDB.
3. Mediante il server node.js e PHP è possibile interrogare OrientDB.
4. Il client interfacciandosi mediante Ajax alla pagina PHP riceve le query richieste in formato JSON

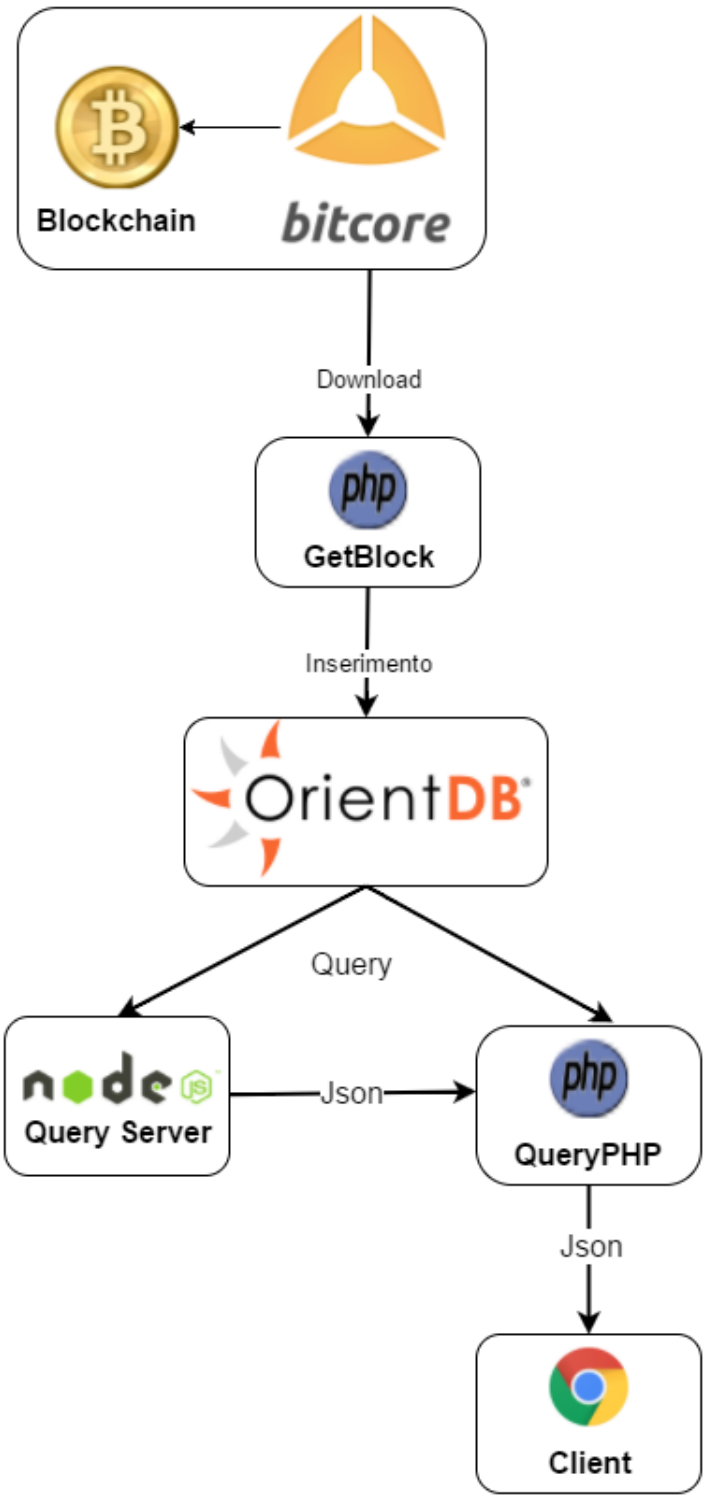

Figura 5.11: Struttura della web application

Conclusioni

In questa tesi abbiamo visto come sia possibile immagazzinare le informazioni contenute nella blockchain in un database a grafo per poi cercare di studiare i flussi di valuta tra i vari indirizzi. Infatti l'insieme di tutte le transazioni bitcoin essendo talmente grande da essere considerabile un esempio di Big Data non è adatto ad essere rappresentato mediante un database relazionale.

Per decidere la struttura in cui immagazzinare i dati abbiamo dovuto comprendere appieno il funzionamento del protocollo Bitcoin, scoprendo alcuni dettagli che non sono conosciuti agli utenti comuni come ad esempio gli script degli output non validi.

Dopo aver fissato le classi di nodi e di archi che avrebbero formato lo schema del nostro database abbiamo dovuto trovare il metodo migliore per scaricare questa grande quantità di dati. Dopo aver appurato di non potersi affidare a servizi di terze parti abbiamo installato nel nostro server un nodo bitcoin per poter interrogare la blockchain in ogni momento ed avere a disposizione tutti i dati di nostro interesse.

Tuttavia il nostro lavoro non è finito qui. Quando si ha a che fare con tanti dati è necessario trovare un modo per dividere questo insieme in sottoinsiemi significativi.

Per fare ciò abbiamo introdotto il concetto di isola, un sottografo dove

le transazioni sono tutte collegate tra loro da indirizzi in comune in un determinato intervallo di tempo. Quindi le query più significative che abbiamo individuato sono:

- Trovare l'elenco delle isole in un determinato intervallo di tempo.
- Trovare tutte le transazioni di una determinata isola.
- Trovare tutte le transazioni in cui è coinvolto un indirizzo.

Per rendere più facilmente usabile la nostra applicazione abbiamo implementato nella nostra applicazione web, una sezione dedicata all'inserimento delle transazioni per poter mantenere aggiornato nel tempo il nostro database e abbiamo aggiunto la possibilità di associare un nome a ciascun indirizzo per poter raggruppare indirizzi che secondo le nostre analisi potrebbero appartenere allo stesso wallet.

Bibliografia

[1] Giuseppe Di Battista, Valentino Di Donato Maurizio Patrignani, Maurizio Pizzonia, Vincenzo Roselli, Roberto Tamassia, "BitConeView: Visualization of Flows in the Bitcoin Transaction Graph",IEEE Symposium on Visualization for Cyber Security, ottobre 2015

[2] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System". www.bitcoin.org/bitcon.pdf, novembre 2008.

[3] Marco Mantilacci,"Un sistema di voto on-line decentralizzato basato sulla tecnologia del Bitcoin".Tesi Università degli studi di Perugia, dipartimento di matematica e informatica, cap. 3, maggio 2014

[4] Andreas M. Antonopoulos,"Mastering Bitcoin,Unlocking Digital Cryptocurrencies", O'Reilly, maggio 2015

[5] Leslie Lamport, Robert Shostak e Marshall Pease,"The Byzantine Generals Problem in ACM" , Transactions on Programming Languages and Systems, luglio 1982.

[6] M. Pease, R. Shostak, L. Lamport, "Reaching agreement in the presence of faults", Journal of the ACM, Aprile 1980

[7] A. Back, "Hashcash – A Denial of Service Counter-Measure", Agosto 2002

[8] Hilbert, Martin, López, Priscila, "The World's Technological Capacity to Store, Communicate, and Compute Information". Science 332, 2011

[9] Marcos D. Assunção, Rodrigo N. Calheiros, Silvia Bianchi, Marco A. S. Netto, Rajkumar Buyya, "Big Data Computing and Clouds: Trends and Future Directions", Journal of Parallel and Distributed Computing, maggio 2015

[10] O'Neil, Cathy and, Schutt, Rachel, "Doing Data Science", O'Reilly, 2014

[11] Rasiel, Ethan, "The McKinsey Way", McGraw-Hill, febbraio 1999.

[12] Robert Amar, James Eagan, and John Stasko, "Low-Level Components of Analytic Activity in Information Visualization", 2005

[13] Leavitt, Neal "Will NoSQL Databases Live Up to Their Promise?", IEEE Computer,2010

[14] Venu Anuganti,"SQL/NoSQL How to choose?", http://venublog.com/, 2011

[15] Katsov, Ilya, "NoSQL Data Modeling Techniques" . marzo 2012

Ringraziamenti

Ringrazio tutte le persone che mi hanno sostenuto durante la stesura di questa tesi. Innanzitutto la mia famiglia per avermi supportato durante questi anni di studi e tutti i miei amici per essermi stati sempre accanto.

Un ringraziamento particolare al prof. Stefano Bistarelli per la disponibilità e la pazienza dimostrata durante la realizzazione di questa tesi.