Gestione e Analisi dei Big Data

Ivan Diliso

Indice

Capitolo 1

Business Intelligence

1.1 Introduzione ai big data

Le principali caratteristiche dei big data sono:

- Volume: Dimensioni che richiedono la distribuzione dei dati
- Velocità: Rapidità dei arrivo e tempo necessario ad elaborarli, possono essere in stream o real time
- Varietà: Dati strutturati e semi strutturati, difficile adattarli ai DBMS moderni (questi presentano forti restrizioni essendo costruiti per assicurare la consistenza). Forte eterogeneità.
- Veridicità: Sorgenti non controllabili o non controllate, incertezza sulle informazioni. Questo problema può non essere considerato creando sistemi robusti ai dati non veritieri).
- Variabilità: Variazioni sia nella struttura che nella semantica dei dati
- Valore: potenzialità dei dati in termini di vantaggi competitivi raggiungibili con la loro analisi

A causa del problema della **scalabilità** volume e velocità sono le caratteristiche considerate più importanti nella gestione deibig data. Sono state proposte soluzioni alla sfida della scalabilità che sfruttano il massive parallel processing (BigTable, DynamoDB, HBase, Cassandra, Hadoop). L'analisi dei big data molte altre sfide oltre alla scalabilità, un data scientist si occupa infatti di:

1. Comprendere i dati disponibili
2. Analisi esplorativa
   (a) Problemi di qualità
   (b) Data cleaning
   (c) Sanity checking
3. Analisi sintattica dei log per ottenere rappresentazioni strutturate
4. Mantenere un catalogo delle tabelle e dei loro schemi
5. Ricostruire attività utente da tabelle di dati distribuite
6. Trattare impedance mismatch
7. Adattare algoritmi sequenziali di machine learning e data mining
8. Integrare strumenti di Big Data analytics in sistemi operativi

1.2 Business intelligence

La business intelligence è l'insieme dei processi, delle tecniche e degli strumenti basati sulla tecnologia dell'informazione e che supportano i processi decisionali di carattere economico. L'obiettivo è avere sufficienti informazioni e conoscenze in modo tempestivo e fruibile cosicché da poter avere un impatto positivo sulle strategie, le tattiche e le operazioni aziendali. Le informazioni/conoscenze riguardano la specifica impresa oppure situazioni più generali di mercato. Quando le informazioni riguardano esclusivamente la concorrenza si parla anche di Competitive Intelligence. Si definiscono diversi tipi di strumenti informatici a supporto delle decisioni aziendali in basi alle differenti attività aziendali (piramide di Anthony).

1.2.1 DSS e EIS

Gli strumenti di supporto al personale direttivo sono i Sistemi di Supporto alle Decisioni (DSS) e Sistemi Informatici Direzionali (EIS). I DSS sono sistemi interattivi che aiutano i decision maker a utilizzare dati e modelli matematici per risolvere problemi decisionali semi-strutturati e non strutturati. Sono formati da:

- Base di dati
- Insieme modelli matematici
- Modulo per la gestione del dialogo tra sistema e utilizzatori

Sono focalizzati su specifici problemi, utilizzati dalla direzione in modo autonomo ma questa non aveva le skill e il tempo per utilizzarli. Gli EIS invece forniscono un ausilio di tipo passivo (rendono più accessibili i DSS), con accesso tempestivo e versatile alle informazioni chiave dell'organizzazione. La principale differenza con i DSS è nella differenza dell'utente utilizzatore, gli EIS vengono utilizzati dai manager per trovare i problemi, i DSS vengono utilizzati dallo staff per studiare i problemi e offrire alternative e soluzioni.

1.2.2 OLTP e OLAP

La teconolgia delle basi di dati era finalizzata alla gestione efficente di dati in linea (On Line Transaction Processing OLTP) ma la natura dei dati decisionali (aggregati) richiede tecnologie per il trattamento di dati analiciti (On Line Analytical Processing OLAP). Nelle basi di dati normali ci sono aggiornamenti modifiche e cancellazioni ma non viene analizzato l'andamento dei dati nel tempo.

OLTP Realizzano i processi operativi dell'organizzazione. Operazioni predefinite, brevi e semplici, vengono rispettate le proprietà ACID e vengono coinvolti pochi dati che devono essere sempre aggiornati. Operazioni quotidiane.

OLAP Sistemi dedicati alla business intelligence nati con lo scopo di supportare i processi decisionali per i soggetti al vertice della piramide di Anthony fornendo informazioni in modo tempestivo e fruibile. Le operazioni olap permettono all'utente di manipolare i modelli dei dati attraverso molte dimensioni di analisi in modo da comprendere i cambiamenti in atto. OLAP è considerata una estensione dei DSS, dove queti permettevano solo di accedere ai dati gli OLAP accedono e analizzano i dati multidimensionali con operazioni quali roll up, pivoting e slicing etc etc. Query complesse e casuali, nessun interesse nel rispettare le proprietà ACID.

	OLTP	**OLAP**
Utente	impiegato	dirigente
Funzione	operazioni giornaliere	supporto alle decisioni
Progettazione	orientata all'applicazione	orientata ai dati
Dati	correnti, aggiornati, dettagliati, relazionali, omogenei	storici, aggregati, multidimensionali, eterogenei
Uso	ripetitivo	casuale
Accesso	read-write, indicizzato	read, sequenziale
Unità di lavoro	transazione breve	interrogazione complessa
Record acc.	decine	milioni
N. utenti	migliaia	centinaia
Dimensione	100GB - 1TB	100TB - 1PB
Metrica	throughput	tempo di risposta

Avendo obiettivi contrastanti OLTP e OLAP possono danneggiarsi a vicenda:

1. OLTP su OLAP non presenta consistenza non avendo proprietà acide
2. OLAP su OLTP può richiedere molto tempo per la computazione

Si sviluppa quindi un nuovo database chiamato data warehouse per le operazioni di analisi OLAP.

Capitolo 2

Data Warehouse

2.1 Data warehouse

Un data warehouse è una base di dati dedicata al supporto decisionale. Rispetta le seguenti caratteristiche:

- Integrato: I dati provengono da diverse sorgenti informative che richiedono una fase di riconciliazione delle eterogeneità (pulizia, validazione e integrazione). Consistente rispetto ad uno schema concettuale globale.
- Orientato al soggetto: Raggruppati per aree di interesse e non per processi operativi. Non è necessaria la normalizzazione.
- Time variant: è di interesse l'evoluzione storica dei dati. Vengono applicati timestamp ai dati operazionali.
- Non volatile: Non può essere modificato dall'utente per mantenere la consistenza e il confronto dell'analisi nel tempo.

Ulteriori concetti chiave sono:

- Aggregato: Contiene informazioni aggregate su specifiche coordinate. Questo può portare a forte ridondanza. I dati di un data warehouse sono:
  - Dati dettagliati attuali e del passato
  - Dati leggermente e fortemente riepilogati
  - Metadati

  Necessaria la scelta del giusto livello di granularità dei dati.
- Autonomo: Ha una esistenza autonoma, separata dalle sorgenti informative.
- Fuori linea: è una base di dati fuori linea, le importazioni sono asincrone e periodiche, un DW infatti può contenere dati non perfettamente aggiornati rispetto alle transazioni dei sistemi OLTP. Il disallineamento deve essere però controllato.

Le differenze con i sistemi OLTP sono simili alle differenze con i sistemi OLAP. Se un OLTP supporta transazioni giornaliere breve e ripetirive un DW deve supportare interrogazioni complesse su enormi quantità di dati.

2.1.1 Data mart

Versione in scala di un DW sviluppato per uno specifico settore, i vantaggi sono i costi e tempi di implementazione ridotti, l'essere controllati localmente dai diversi settori e forniscono tempi di risposta più rapidi essendo più piccoli. I vari data mart possono differire molto tra i vari settori, nella creazione di un data warehouse generale è necessaria una fase di integrazione.

2.1.2 Qualità dei dati

I fattori che pregiudicano la qualità dei dati sono le basi di dati prive di vincoli di integrità e il problema del disallineamento dei dati provenienti da diverse fonti. Fondamentali l'impiego di filtri e osservazione del processo di produzione dei dati.

- Accuratezza: la conformità fra il valore memorizzato e quello reale.
- Attualità: il dato memorizzato non è obsoleto.
- Completezza: non mancano informazioni.
- Consistenza: la rappresentazione dei dati è uniforme.
- Disponibilità: i dati sono facilmente disponibili all'utente.
- Tracciabilità: è possibile risalire alla fonte di ciascun dato.
- Chiarezza: I dati sono facilmente interpretabili.

2.2 Architettura

Esterni (Extraction Transformation e Loading ETL):

1. Data source: sorgenti diversi provenienti da DBMS (anche legacy), dati non gestiti da un DBMS.
2. Filtraggio: controllano la correttezza dei dati, eliminazinoe dei dati scorretti sulla base di vincoli e controlli, rilevare e correggere inconsistenze nelle diverse data source. (Data cleaning)
3. Export: Estrazione dei dati dalla data source, l'esportazione può essre incrementale (vengono collezionate le sole modifiche che vengono poi importate dal DW)

Interni:

1. Acquisizione: Caricamento inizile dei dati. Predispone i dati all'uso operativo, svolgengo operazioni di riconciliazione delle eterogeneità, di aggregazione, denormalizzazione, ordinamento e costruzione delle strutture dati del DW. Nel caso di archietetture distribuite si occupa della frammentazinoe iniziale dei dati. Opera in modo batch quando il DW non è utilizzato. Invocato periodicamente se il volume di dati è piccolo.
2. Allineamento: propaga incrementalmente le modifiche della data source in modo da aggiornare il contenuto del DW.
    - Invio dei dati: Utilizza i trigger della data source che in modo trasparente registrano inserimenti cancellazioni e modifiche negli archivi variazionali, le modifiche vengono quindi trattate come coppie di inserimenti e cancellazioni.

- Invio delle transazioni: Vengono utilizzati i log di transazione per costruire archivi variazionali.

In entrambi i casi questi vengono utilizzati per rinfrescare il DW, aggiungendo i dati. Per le cancellazioni i dati vegono mearcati come storici ma non cancellati.

3. Accesso: Realizza le operazioni necessarie all'analisi dei dati. Questo modulo realizza interrogazioni complesse e consente operazioni come roll-up e drill down per supportare le funzionalità OLAP.
4. Data mining: Ricerche complesse per scovare informazioni nascoste nei dati
5. Export: Esportare i dati da un DW in un altro DW costruendo cosi una archietettura gerarchica.

Moduli di ausilio:

1. Assistenza allo sviluppo: Permette di definire lo schema dei dati e il meccanismo per l'importazione dei dati.
2. Dizionario dei dati: Descive il contenuto del DW

2.3 Schema concettuale

Rappresentazione di un data warehouse tramite un modello multidimensinale che permette di rappresentare i dati organizzti per aree di interesse, fornendo una rappresentaizone di alto livello.

- Fatto: Concetto sul quale ha senso svolgere un processo di analisi
- Misura: Proprietà atomica di un fatto che intendiamo analizzare
- Dimensione: Particolare prospettiva lungo la quale l'analisi di un fatto può essere condotta. I valori possibili sono detti membri. Le dimensioni di analisi possono essere organizzate in gerarchie.

2.3.1 Cubo multidimensionale

Rappresentazione grafica tramite un cubo (data cube) costituito da elementi atomici detti celle. Un cubo è incentrato su un fatto, ogni asse del cubo è una dimensione di analisi. Per accedere ai dati necessario specificare le coordinate.

- Se per ogni dimensinoe viene specificato un valore ben prciso allora nell'ipercubo verrà individuata una cella o un singolo fatto.
- Se per una dimensine si specifica un valore ben preciso si determina una fetta (slice) dell'ipercubo

Le interrogazioni sono selezioni di porzioni di ipercubo, sono quindi più semplici delle interrogazioni delle basi di dati relazionali (non sono necessarie complesse operazioni di join). Questo modello può sembrare simile ad un array multidimensionale ma a differenza di questo sugli indici potrebbe non essere deginito un ordino o si potrebbe avere un ordinamento parziale.

2.3.2 Operazioni sul cubo

Le operazioni che si aplicano ai cubi dimensionali restituiscono nuovi cubi non necessariemante con lo stesso numero di dimensioni:

- Slice: Fisso un valore per una delle dimensioni e analizzo i dati delle altre (ipercubo n-1, contrazione dimensionale)
- Dice: Fisso un intervallo su ciascuna dimensione, si analizza una riduzione volumetrica.
- Roll up: Consente di risalire lungo una o più dimensioni gerarchiche
- Drill down: Scende in dettagli lungo una dimensione gerarchica, utile per analizzare una causa o un effetto perqualche fenomeno osservato nei dati aggregati
- Rolling o pivoting: Cambiameto della modalità di presentazione, analizzo le stessa informazioni sotto divesi punti di vista. Rotazione del cubo portando in primo piano una differente combinazione di dimensioni (se m dimensionale m! diverse propsettive)

2.4 Schema logico

Traduzione di un modello concettuale multidimensionale in un modello logico con proprietà analoghe. Questo presuppone l'uso di un Multi Dimensionale DBMS (MDDBMS) che presentano:

- Vantaggi: Ottimizzati per valocizzare e semplificare le interrogazioni grazie a operatori di pre elaborazione
- Svantaggi: Problemi di scalabilità, disponibilità verso le aziende solo di basi di dati relazionali.

Prendo quindi in considerazione la possibilità di trasformare un modello concettuale multidimensionale in un modello logico relazionale. Si presentano quindi problemi di simulazione di un approccio multidimensinale.

2.4.1 Stella

Presenta una tabella centrale (tabella dei fatti) che contiene i dati di una cella dell'ipercubo del modello multidimensinale e le chiavi che colelgano i dati relative alle dimensioni. Le tabelle dimensionali contengono gli attributi che descrivono le componenti dei dati. I fatti sono in forma normale di Boyce-Codd in quanto ogni attributo non chiave dipende funzionalmente dalla sua unica chiave. Le tabelle di dimensione sono denormalizzate e ridondanti in moto da ridurre il numero di join necessario per risolvere una query e aumentare l'efficenta, portando però ad uno spreco di memoria dovuto alla ridondanza. Le misure devono necessariamente essere numerici.

2.4.2 Costellazione

Utile per rappresentare più ipercubi, situazioni in cui abbiamo vari gruppi di misure. Si presentano più tabelle dei fatti, queste possono condividere le dimension table, in questo caso si può procedere in due modi:

- Separazione dei fatti
- Schema a costallazione: preferibile in quanto consente di ridurre notevolmente lo spazio di memoria, ma la condivisione richiede la conformità o consistenza dei valori degli attributi delle dimensioni dei due schemi

2.4.3 Fiocco di neve

Le dimensioni vengono normalizzate, consentedo di dividere i dati in funzione delle gerarchie individuate. La tabella dimensionale relativa al più baso livello gerarchico è collagata alla tabella dei fatti. L'utilizzo di questo schema è sconsigliato in quanto il guadagno in termini di occupazione di memoria non è sufficente a compensare la perdita di efficentza dovuta al maggior numero di join necessraio per risolvere la gerarchia, il guadagno di memoria inoltre è poco significativo. Questo schema permette di rappresentare relazioni molti a molti ed è inoltre utilizzato per rappresentare dati non analitici o di dettaglio (che spesso non sono considerati nella progettazione del DW) tramite l'utilizzo di shadow table collegate alla tabella dimensionale. Interrogazioni che non richiedono dati analitici accederanno direttamente alla tabella dimensionale senza attraversare la shadow table Uno schema a fiocco di neve può anche essere applicato parzialemente, solo sui rami che lo necessitano.

2.4.4 Dimensione del tempo

Utile modellare il tempo come una gerarchia per far fronte ad interrogazioni che richiedono periodi particolari di tempo (festività, stagioni, etc). Necessario valutare la granularità del tempo per evitare grandi sprechi di spazio.

2.5 Operatori in SQL

- cube: effettua tutte le possibili aggregazioni su una tabella in base agli attributi di raggruppamento specificati. Viene utilizzato un nuovo valore polimorfo ALL che corrisponde all'insieme di tutti i possibili valore presenti nel dominio. La complessità di valutazione cresce in modo combinatorio col crescere del numero di attributi di raggruppamento
- rollup: Simile a cube ma le aggregazini sono progressive rispetto all'ordine degli attributi di raggruppamento

Un altra funzione utile è la materalizzazione delle viste, che permette di valutare viste che esprimano i dati aggregati una volta per tutte e memorizzarle. Questo aggrava però le esigenze di memoria, questo può essere ulteriormente aggravato dalla presenza di gerarchie. Se L_i indica il numero di livelli gerarchici associati alle i-esima dimesnione di un cubo n-dimensionale il numero totale di cuboidi è:

$$T = \prod_{i=1}^{n} (L_i + 1)$$

Si procede quindi alla materializzazione parziale che coinvolge solo i cuboidi con maggiore frequenza di accesso. Tabelle non materializzate permettono di avere dati sempre aggiornati in quanto la query è fatta al momento ma in caso di query ripetute questo potrebbe significare ripetere calcoli complessi.

2.6 Utilizzo

2.6.1 Reportistica

Approccio di query e reporting orientato agli utenti che hanno necessità di accedere, a intervalli di tempo predefiniti, a informazioni strutturate in modo pressoché invariabile. Un report è formato da:

- Interrogazione: Selezione e aggregazione dei dati multidimensionali
- Presentazione: In forma tabellare o grafica

La reportistica si valuta in base alla ricchezza della presentazione e alla flessibilità dei meccanismi per la distribuzione, il rapporto può infatti essere generato su richiesta o essere distribuito automaticamente e periodicamente. La reportistica non nasce con il data warehousing ma questo apporta banefici quali:

- Affidabilità e correttezza dei risultati
- Tempestività

2.6.2 Altri utilizzi

Utilizzati per OLAP, DSS/EIS e data mining (per dettagli visionare i paragrafi corrispondenti)

2.7 Progettazione

2.7.1 Dati in ingresso

- Requisiti: descrizione delle esigenze aziendali di analisi
- Schemi delle sorgenti informative aziendali: descrizione formale della struttura delle basi di dati operative disponibili. delle sorgenti informative (sistemi legacy) e ralativa documentazione si supporto
- Schemi di altre sorgenti informative: schemi di dati non di proprietà dell'azienda ma accessibili da essa

2.7.2 Analisi dei dati in ingresso

1. Selezione delle sorgenti informative: banale quando la sorgente è solo una, più complessa quando è presente un db centrale da integrare con altre informazioni.
2. Stabilire delle preferenze: a parità di informazione se preferire un sorgente su un altra
3. Rappresentare le sorgenti selezionate: rappresentazione con un unico formalismo. In caso di assenza di documentazione creare un unico schema a partire dalla tabella tramite reverse engineering
4. Analisi schemi delle sorgenti: Mira ad invidurare nei vari schemi concetti irrilevanti e ridondanze.

2.7.3 Integrazione

Fusione dei dati presenti nelle varie sorgenti informative in un unica base di dati globale. Vengono identificate le porzioni delle sorgenti informative che si riferiscono a uno stesso aspetto della realtà e alla loro unificazione di rappresentazione. Si risolvono conflitti di tipo:

- Terminologici: diversi termini per rappresentare lo stesso concetto
- Strutturali: rappresentazione dello stesso schema in formati diversi. DUe schemi relativi allo stesso concetto potrebbero contenere dati differenti
- Di codifica: uso di diversi criteri per rappresentare la stessa informazione

Procedure per la riconciliazione dei dati quando vengono caricati nel DW (ETL). Produco una vista integrata del patrimonio informativo dell'azienda.

2.7.4 Progettazione del data warehouse

- Top down: sviluppo il DW aziendale e poi si costruiscono i data mart settoriali
- Bottom up: si parte dai data mart e il data warehouse si ottiene in maniera incrementale

In scenari complessi è preferibili il secondo approccio in quanto lo sforzo realizzativo è fortemente ridotto. La progettazione richiede di guardere oltre ai requisiti anche allo schema integrato creato nella fase precedente.

Capitolo 3

NoSQL

3.1 NoSQL Data Model

Sistemi puramente basati sulla scalabilità (scale) che sulle analytics. Si allontana dal modello relazionale:

- Key Value, Document, Column Family (Aggregate orientation)
- Graph
- Sparse

3.1.1 Aggregate orientation

I database basati su aggregati sono progettati per gestire i dati con unità logiche chiamate aggregati, che rappresentano la parte più granulare dei dati gestiti dal sistema. Un aggregato è una collezione di oggetti legati tra loro che vogliamo trattare come unità per la manipolazione, gestione e consistenza. Spesso pensati come unità astratte che rappresentano oggetti del mondo reale. Nei database aggregate orientede si ha una visione chiara della semantica permettendoci di dare attenzione all'unità di interazione con lo storage. Nell'approccio aggregate l'unita dei dati ha un struttura più complessa di un set di tuple. I record possono contenere liste, mappe o altre strutture dati.

- I modelli aggregati facilitano la divisione di grandi quantità di dati in cluster in quanto sono unità per la replicazione (multiple copie distributie) e lo sharding (partizionamento orrizontale dei dati). è richiesto pero di minimizzare il numero di nodi da interpellare quando vengono richiesti i dati, la nozione di aggregato da al databse una visione di quali informazioni dovrebbero essere conservate insieme.
- Permettono inoltre di risolvere il problema dell'impedance mismathc dei database relazionali.
- Rimane comunque il problema della query su dati storici.

Si pone quindi l'obiettivo di progettare aggregati in modo da minimizzare il numero di aggregati a cui accediamo durante una data interaction. Nel design bisogna pensare a come accedere ai dati. La progettazzione dell'aggregato dipende dal modello di accesso ai dati.

3.1.2 Aggregate ignorant

Database che non hanno il concetto di aggregato, come database relazionali e a grafo. Sono utili in domini in cui è difficile definire i confini dell'aggregato o quando è necessaria una analisi dei dati nel tempo (si dovrebbero analizzare tutti gli aggregati per estratte l'history dei cambiamenti). Guardando le specifiche del modello relazionale:

- Le informazioni hanno una struttura dati limitata, divisa in tuple, che cattura solo un insieme di valori, non è possibile innestare una tupla in un altra o utilizzare array
- La manipolazione dei dati è vista come operazioni che prendono in input le tuple e restituiscono tuple.
- Le relazioni sono rappresentate dalla relazione di chiave esterna, non è possibile distinguere una relazione che rappresenta una aggregato da una che non lo è, non è quindi possibile sfruttare questa informazione per distribuire e conservare i dati. Nonostante esistano delle tecniche di modellazione che permettono di markare strutture aggregate nei modelli relazionali queste non permettono di definirne la semantica che le differenzia dalle altre relazioni.

3.1.3 Teorema Brewer's CAP

Un sistema distribuito può supportare solo due di queste caratteristiche:

- Consistency: Tutti i nodi han no lo stesso valore
- Availability: Ogni richiesta ha una risosta (non ci sono errori)
- Partition tolerance: Il sistema continua ad operare anche se un numero arbitrario di messaggi si perde o è ritardato nella rete dei nodi

In generale, i modelli key-value e document-based garantiscono availability e partition tolerance, ma non la consistency in tempo reale. Ciò significa che i dati possono essere recuperati in qualsiasi momento anche se uno o più nodi vanno offline, ma potrebbe essere necessario del tempo prima che questo sia visibile in tutti gli altri nodi del sistema. Nel caso dei modelli document-based e key-value, il partizionamento avviene utilizzando algoritmi di hash (come il Consistent Hashing). I modelli column-family, come big table, individuano come unità di distribuzione un range di identificatori di chiavi, chiamato tablet. Quindi ad ogni nodo verrà associato un tablet. Nel caso di modelli a grafo, la partition tollerance è poco applicabile visto che distribuire nodi del grafo significa coinvolgere un elevato numero di macchine ad ogni query. Piuttosto ci si concentra su consistenza e affidabilità. Ciò viene gestita in modo diverso a seconda del DBMS. In genere abbiamo un master sul quale avvengono le operazioni di scrittura, e dei slave che possono partecipare alle query per ridurre i tempi di risposta.

3.2 Modelli aggregati

	Key Value	Document
	Insieme di aggregati con una chiave per accedere ai dati	
Tipo di aggregato	Opaco (BLOB) (Dato definito dall'utente fully encapsulated con struttura interna ignota al database)	Vediamo una struttura nell'aggregato
Dati consentiti	Qualsiasi tipo (Un database può imporre un limite)	Imposto un limite su cosa possiamo inserire definendo una struttura dei dati (in questo modo abbiamo un lin guaggio di query dei documenti)
Accesso	Chiave	Query sui campi Richiedere parti dell'aggregato Il database può creare indici basati sui campi dell'aggregato

Nello specifico il modello key value è una tabella di hash dove ogni accesso al database avviene tramite chiave primaria. Un utente può leggere il valore per una chiave, inserire un valore per una chiave o eliminare una chiave. Le caratteristiche chiave sono:

- Alte performance e avalilability
- Le chiavi e valori possono essere oggetti compound complessi (anche liste mappe o altre strutture dati)
- Consistenza è applicabile solo per le operazioni su singola chiave (eventual consistency)

E presenta i seguenti vantaggi e svantaggi:

- Vantaggi: Query efficenti, facilità di distribuzione su cluster, non è presente object-relational mismatch
- Svantaggi: Non sono presenti filtri complessi per le query, i join devono essere effettuati nel codice, non ci sono vingoli di chiave esterna e non ci sono trigger

In sintesi, i database orientati ai documenti sono più adatti per i casi in cui si ha la necessità di eseguire query sui campi dei documenti, i database key-value sono più adatti per i casi in cui l'accesso agli aggregati avviene solo sulla base della chiave.

3.3 DynamoDB

Database peer to peer key value con SLA al 99.99% percentile che può supportare aggiunta di nodi e fallimento online, supporta inoltre object versioning e risoluzione dei conflitti assistita. Nasce dall'esigenza di Amazon di avere accessi ai dati solo tramite la chiave primaria che porta forte inefficenza con soluzioni di tipo relazionale e limitano la scalabilità.

3.3.1 Scalabilità e affidabilità

La affidabilità e scalabilità di un sistema dipende da come è gestito lo stato dell'applicazione. Amazon utilizza una architettura orientata al servizio fortemente decentralizzata e accoppiata in modo flessibile composta da centinaia di servizi. é necessario che lo storage sia sempre accessibile (anche in caso di crash, evento normale in sistemi cosi grandi)

- SLA (Service Level Agreement): Garanzia che l'applicazione può fornire i suoi servizi in un tempo limitato. Descritta in termini di media, mediana e varianza attesa.

3.3.2 Interfaccia utente

Sono presenti due operazioni:

- get(key): Localizza le repliche dell'oeggetto associato alla chiave e restituisce un insieme di oeggetti con versioni conflittuali insieme ad un contesto
- put(key, context, object): Determina dove le repliche dell'oggetto vanno inserite e le scrive su disco
- context: Codifica i metadati del sistema sull'oggetto

Chiave e oeggetto sono entrambi array di byte opachi su cui viene applicato un hashing MD5 per generare un identificatore a 128 bit per determinare i nodi che sono respondabili di fornire la chiave.

3.3.3 Partitioning e consistent hashing

Dynamo deve avere uno scaling incrementale per questo necessita di un meccanismo per partizionare dinamicamente i dati su un set di nodi. Lo schema di partioning si basa su consistent hashing, un metodo in cui l'output range di una funzione di hash è trattato come uno spazio circolare per l'assegnazione dei nodi ai dati.

1. Ogni nodo nel sistema è associato ad una valore randomico all'interno dell anello (rappresenta la sua posizione sull'anello)
2. Si applica l'hashing sulla chiave dell'oggetto per trovarne la posizione sull'anello
3. Si percorre l'anello in senso orario dalla posizione trovata per trovare il primo nodo con una posizione più grande della posizione dell'item

Ogni nodo è quiondi responsabile della regione dell'anello delimitata da se stesso e il nodo precedente. In caso di arrivo di un nuovo nodo o fallinmento di un nodo vengono modificati solo i nodi vicini (neighbor). Il consinsten hashing permette di risolvere i seguenti problemi:

- Data una chiave e una lista si server come trovo i server primari, secondari e terziari per la risorsa? A partire dalla posizione della risorsa percorro l'anello fino a trovare tre server
- Se ho server con risorse hardware diverse come posso assegnare una quantità di lavoro adeguata? Server con capacità maggiore avrenno una porzione di anello più grande
- Come posso aggiungere capacità al sistema senza downtime?
  - Come posso evitare di scaricare 1/N del carico totale ad un server appena questo si accende?
  - Come posso evitare di rehash più chiavi del necessario?

  Risolti tramite l'utilizzo di nodi virtuali

3.3.4 Nodi virtuali

Dato che la posizione degli oggetti sull'anello è randomica è possibile che si crei un distribuzione non uniforme degli oggetti. Per risolvere questo problema è stata introdotta l'idea dei nodi virtuali.

- Ad ogni server è assegnato più di un nodo nell'anello
- Ogni nodo è responsabile di più di un nodo virtuale
- Quando un nodo viene aggiunto all'anello gli vengono assegnate posizioni multiple

Questo risolve i problemi citati in precedenza:

- Se un nodo diventa inacessibile il carico gestito da questo nodo viene disperso in maniera equa nei nodi della rete
- Se un nodo torna accessibile questo accetta un numero equivalente di carico dagli altri nodi della rete
- Il numero di nodi virtuali è scelto in base alla capacità del server

3.3.5 Data replication

Per ottenere alta availability e durabilità Dynamo replica i suoi dati su host multipli.

1. Ogni dato è replicato su N host (con N parametro configurato per istanza)
2. Ogni chiave k è assegnata ad un nodo coordinatore
3. Il coordinatore applica la replicazione dei dati che ricadono nella sua porzione di anello
   (a) Salva ogni chiave nel suo range
   (b) Replica queste chiavi ai N-1 nodi successori nell'anello

3.3.6 Eventual consistency

Le scritture sono propagate in modo asincrono, la get() può restituire un oggetto che non ha gli ultimi update. Ad esempio, in caso di aggiunta di un item ad un dato, se l'ultime versione non è disponibile questo sarà aggiunto alla versione più recente disponibile e le due versioni divergenti saranno riconciliate successivamente. Ogni modifica viene trattata come una versione nuova e immutabile del dato, permettendo a più versioni dello stesso oggetto di essere presenti nel sistema allo stesso tempo. Le nuove versioni possono sussumere quelle vecchie e il sistema può determinare la versione authoritative (syntactic reconciliation)

3.4 Column Family

La struttura dei dati non è una tabella ma una mappa a due livelli. Questi modelli sono chiamati "column stores" o "column family". La differenza del column storage NoSQL è nel modo di conservare fisicamente i dati. Vengono infatti conservati gruppi di colonne per tutte le righe come unita base di storage. Particolarmente utile quando:

- Le operazioni di scrittura sono rare
- Devo accedere a poche colonne di tante righe

La struttura alla base è aggregato a due livelli:

1. Identificatore di riga: Recuperare una chiave ritorna una mappa di dati dettagliati
2. Colonne

Fissando una riga possiamo accedre a tutte le column family o ad un particolare elemento. Ogni colonna è parte di una famiglia è la column family è una unità di accesso. Se nei database orientati alle righe ogni riga è vista come un aggregato, nei databse orientati alle colonne ogni column family definisce un tipo di record con righe per ognuno dei record, l'informazione dei raggruppamenti delle colonne viene utilizzata per lo storage e l'accesso ai dati. Gli elementi di una column family sono ordinati sulla chiave, questo permette di rinominare le colonne ad accesso frequente in modo che siano le prime ad essere ritrovate. Nella scelta della modellazione dei dati tramite column family bisogna ricordare che le scelte vanno fatte per i requisiti delle query e non per scopi di scrittura. La regola generale è di rendere facile le query e denormalizzare i dati durante la scrittura.

3.4.1 BigTable

Struttura formata da colonne sparse e nessuno schema implementato come una multi dimensional sorted map sparsa, distribuita e persistente. Utilizzato per batch oriented processing e low latency serving dei dati agli utenti. Non è supportato un modello full relational ma con un modello in grado di:

- Controllo dinamico del layout e formato dei dati
- Permette ai clienti di ragionare sulla localizzazione dei dati nello storage sottostante.

Le sue principali caratteristiche sono:

- Indexare i dati usando nomi di riga e colonna (stringhe arbitrarie)
  - Livello 1: Row key (indica ordine lessicografico)
  - Livello 2: Column key
  - Livello 3: Timestamp
  - Value: Array di bytes
- Salvare i dati come stringhe non interpretate
- Operazioni di lettura e scrittura dei dati di un singola riga è atomica

3.4.1.1 Tablet e column family

Insiemi ordinati di row key sono raggruppati (tablet) in range e partizionati dinamicamente

- Tablet unità di distrubuzione e bilanciamento del carico
- Letture di piccoli tablet sono efficenti e richiedono la comunicazione con un piccolo numero di macchine

Column keys sono raggruppati in column families contenenti dati dello stesso tipo

- Si cerca di mantenere un basso numero di column family (minore di 101)
- Numero di colonne non limitato

3.4.1.2 Timestamps

Ogni cella contiene versioni multiple dello stesso dato indexate e ordinate dal timestap in microsecondi (integer a 64 bit). L'ordinamento permette di leggere come prima la versione più recente. Vengono conservate 3 versioni dello stesso dato, i dati immutati nel tempo vengo garbage collected automaticamente.

3.5 Database orientati a grafo

Gli oggetti sono modellati come nodi e le relazioni tra essi come archi tra i nodi. Viene messo in primo priano il concetto di relazione. Ogni relazione possiede un nome, una direzione e un tipo. Utilizzando i database relazionali e NoSQL è difficile gestire dati implicitamente connessi:

Relazionali I database relazionali hanno difficoltà a modellare dati di tipo connesso e semi strutturato in quanto sono progettati per modellare strutture tabulari e hanno difficoltà a modellare relazioni had hoc situazionali (outlier) del mondo reale (si creerebbero tabelle molto sparse, non tutte le righe hanno tutte le relazioni, inoltre sarebbero necessari molti join per eseguire le query).

NoSQL I dati sono salvati come insiemi di documenti, valori o colonne disconnessi tra loro, anche se è possibile inserire un identificatore dell'aggrefato all'interno di un altro aggregato questo non risulta efficente, in quanto necessita di join complessi a livello applicativo. Le relazioni non sono cittadini di prima classe.

3.5.1 Caratteristiche

L'utilizzo di un database orientato ai grafi porta le seguenti caratteristiche:

- Consistenza: Operando su nodi connessi non permette una efficente distribuzione dei nodi su più server
- Transazioni: ACID compolaing, prima di eseguire una operazione bisogna inizializzare una transazione
- Availability: Dipendente dall'applicazione, Neo4j la ottiene utilizzando slave replicati con uno zookeeper che tiene traccia dell'ultima transazione avvenuta su ogni nodo slave e sul nodo master

3.5.2 Relazioni

Le relazioni di un grafo formano naturalmente delle paths, l'attraversmento o l'esecuzione di una query coinvolger il seguire una path (traversing), è possibile cambiare i requirement di traversing senza cambiare nodi e archi. Le operazioni di attraversamento di un grafo sono estremamente più efficenti su un database orientato a grafo rispetto a un database relazionale.

3.5.3 Pattern

Più grafi proveniente da diversi domini possono essere uniti, questo permette di sfruttare le relazioni tra i nodi e tra i grafi per ritrovare pattern nel grafo (operazione di difficile scrittura in SQL e computazionalmente complessa sui database aggregati)

3.5.4 Query language

1. Indexing: Necessari per trovare il nodo iniziale del percorso. è possibile indexare proprietà di nodi e archi
2. Relazioni in e out: Utilizzo il nodo iniziale per trovare relazioni in entrata e in uscita. è possibile applicare filtri direzionali

3. Strategia di ricerca: La ricerca può avvenire top down (depth first) o sideways (breadth first)

4. Querying Paths: I database a grafo permettono di trovare il percorso tra due nodi, trovare percorsi multipli, e trovare il percorso più breve (Dijkstra's)

5. Operatori:

   - Match: Matching di pattern nelle relazioni
   - Where: Filtro delle proprietà di un nodo o di una relazione
   - Return: Coso inserire nel result set

3.5.5 Data modeling

In questo tipo di modellazione, a differenza delle altre metodologie, c'è una forte vicinanza tra il modello logico e il modello fisico. UN grafo delle proprietà è formato da:

- Nodi: Documenti che contengono le proprietà in forma di coppie chiave valore, con chiave stringhe e valore di tipo arbitrario
- Relazioni: Formate da direzione, nome, nodo di partenza e nodo di arrivo. QDIrezione e nome danno chiarezza semantica nello strutturare i nodi. Una relazione può anche contenere proprietà, queste permettono di:
  - Aggiungere metadati
  - Aggiungere semantica
  - Filtrare le relazioni tramite le query

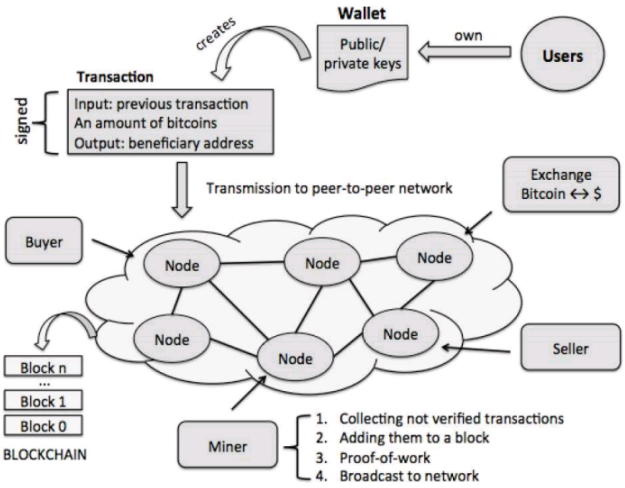
Capitolo 4

Bitcoin

4.1 Bitcoin

Bitcon è un peer to peer electronic cashsystem implementato per la prima volta nel 2009 con il primo nodo della rete chiamato Bitconi-Qt. Chiamata anche una criptovaluta a causa dell'uso della crittografia asimmetrica per l'invio e la riscossione della moneta.Può essere visto come un database distribuito che mantiene lo storico di tutte le transazioni.

Crittografia assimmetrica Se A deve mandare un messagio a B lo cifra utilizzando la PUBKEY (chiave pubblica) di B. Successivamente per decifrare il messaggio B utilizzerà la usa SIGNATURE (chiave privata).



Concetti chiave sono:

- Portafoglio: Chiave pubblica e privata, ogni utente può avere un numero arbitrario di portafogli (anonimicità). I portafogli sono generati manualemente dagli utenti
- Transazioni: Broadcasting delle transazioni a tutti i nodi della rete, decentralizzazione. Ogni utente può visualizzare liberamente tutte le transazioni.
- Controllo: Eseguito dai miner tramite protocolli di consenso che disincentivano i comportamenti scorretti (proof of work).

4.2 Struttura della blockchain

Una blockchain è una linked list in cui ogni blocco ha un riferimento al blocco immediatamente precedente. Ogni blocco contiene un certo numero di transazini che sono state validate come corrette.

- Codice identificativo unico: Hashing del contenuto del blocco e un numero chiamato nonce.
- Transazioni: Insieme delle transazioni verificate
- Codice identificativo del blocco precedente

4.3 Portafoglio

Un portafoglio bitcoin è formato da una chiave pubblica e una chiave privata:

1. Genero una chiave privata randomica a 256 bit. Se viene utilizzato un seed per la generazione della chiave è possibile utilizzarlo per ritrovare la chiave se persa.
2. Ottengo la chiave pubblica multiplicando la chiave con un punto generatore G tramie ECDSA (Eliptic Curve Criptography). La chiave pubblica è quindi un punto in notazione compressa:

$$K = (x, y) \rightarrow 04xy \begin{cases} 02x & \text{se y pari} \\ 03x & \text{se y dispari} \end{cases} \tag{4.1}$$

3. Applico RACE Integrity Primitives Evaluation Message Digest (RIPDEM 160) all'hashing SHA256 della chiave pubblica per ottenere la fingerprint
4. Compongo la chiave $(0x00 + \text{fingerprint} + \text{checksum})$ e applico Base58 (il checksum è formato dai primi 4 bytes di SHA256(SHA256(fingerprint))) per ottenere il bitcoin address.

Indirizzo Bitcoin

- Primo carattere: 1 o 3
- Non contiene caratteri ambigui
- Stringa da 25 a 33 caratteri in cui i byte iniziali a 0 sono esplessi come un singolo 1.

Saldo Somma delle UXTO (Unspent Transaction Output), transazioni di bitcoin non spese (ho ricevuto del denaro che non ho usato). Possono essere visti come dei cassetti che contegono denaro, una transazione invia al mio indirizzo un cassetto con del denaro, quando eseguo una transazinone utilizzo l'intero contenuto di uno o più cassetti. Il saldo infatti non è un valore reale ma si possono utilizzare dei tool che lo calcolano sommando le UXTO. Come spiegato in più dettaglio più avanti, le UXTO sono gli output delle transazioni, le reference agli output di queste transazioni rappresentano l'ammontare di denaro a cui un utente ha acesso e sono direttamente utilizzati nelle transazioni.

4.4 Algoritmo di consenso distribuito

Bitcoin è un sistema distribuito decentralizzato in cui il controllo delle transazioni è in mano agli stessi utenti della rete. Ogni nodo quando riceve una transazione la aggiunge alla pool di tutte le transazioni non ancora aggiunte alla blockchain e da essere seleziona una sottoinsieme da aggiungere nel nuovo blocco della rete. Gli utenti stessi (miner) controllano la veridicità e correttezza delle transazioni e le eseguono, è necessario quindi un sistema che non permetta ai miner di validare transazioni fraudolente e disincentivi i tentativi di compromettere la rete.

Consensus protocolo: lavoro e comunicazione La verifica delle transazioni è una azione facile, viene quindi resa complessa tramite un hashing complesso (lavoro), il broadcast dei blocchi a tutta la rete permette a tutti i nodi di verificare le transazioni e nel caso rifiutarle (comunicazione)

4.4.1 Proof of work

Il calcolo del codice identificativo è una semplice operazione di hashing dei contenuti del blooco e di un numero chiamato NONCE. Questa operazione è resa più complessa da un requirement imposto sul risultato dell'hashing, questo infatti dovrà avere un certo numero di 0 iniziali nel risultato. Questo richiede cambiare una parte dell'input all'algoritmo di hashing finchè non si raggiunge il risultato richiesto, la parte che viene fatta variare è il NONCE (si provano tutti i numeri finchè non si raggiunge un risultato richiesto). Il requirement dell'hashing è dinamico e varia in base al numero di miner nella rete correnti e la potenza di calcolo totale dei nodi (hash rate). Questo meccanismo permette di:

- Fornire una prova di un lavoro effettuato nel convalidare una transazione. Non conviene convalidare una transazione fasulla in quanto si dovranno sprecare delle risorse e gli altri nodi della rete potrebbero rifiutarla.
- Ridurre gli accessi alla rete e i miner.
- Rallentare o velocizzare il numero di blocchi prodotti aumentando o diminuendo la difficoltà dell'hashing.
- Riduco la possibilità che due miner minino lo stesso identico blocco grazie ad un hash complesso che richiede tempo

Il mining non è un lavoro gratuito, infatti quando viene validato un blocco il protocollo aggiunge automaticamente una transazione con mittente nullo e destinatario la chiave pubblica del miner con una ricompensa in bitcoin (questi bitcoin sono creati al momento). La ricompensa minata si riduce ogni tot blocchi creati. La ricompensa del miner è quindi l'ammontare di bitcoin minato più le eventuali mance (fee) presenti nelle transazioni.

4.4.2 Fork

Un fork avviene quando due nodi della rete minano contemporaneamente un blocco e lo propagano ai nodi neighbor della rete. Durante la propagazione si raggiungerà un punto in cui una parte dei nodi della rete avrà una blockchain contentente il blocco A e l'altra parte avra il blocco minato B. La scelta della catena corretta avviene in base alla lunghezza della catena, se viene infatti aggiunto un blocco alla catena B prima che venga aggiunto alla catena A questa diventerà la catena principale e verrà propagata e accettata da tutti i nodi della rete. Le transazioni presenti nella catena rimossa vengono nuovamente aggiunte nel pool delle transazioni. Bitcoin non gestisce gestione e controllo delle frodi o blocchi del wallet. Un blocco può ritenersi immutabile dopo 6 blocchi successivi

4.4.2.1 Hard fork

Si presenta un hard fork quando un insieme consistente di miner non accetta un cambiamento nei protocolli della rete o la rimozione di uno dei fork della rete a causa di una catena più lunga. In questo caso il fork della rete diventa un nuova catena e una nuova criptovaluta. Esempi sono il bitcoin cash che nasce da un disagreement sulla dimensione del blocco di bitcoin o l'hard fork eseguito dalla ethereum fondation per ripristinare lo stato della rete dopo una grande quantità di fondi persi a causa di una attacco alla rete. Il ramo con la transazione fraudolenta è ora chiamato ethereum classic e non è mantenuto dalla ethereum foundation.

4.5 Vulnerabilità

4.5.1 Double spending

Questo porta a problemi quali il double spending, in quanto potrei provare a spendere dei bitcoin prima che questi siano davvero inseriti stabilmente nella rete (potrebbero essere rimossi successivamente a causa di un fork). Questo problema è limitato dal proof of work tramite gli hash complessi e il limite di tempo per l'aggiunta di un blocco (in bitcoin può essere aggiunto un blocco ogni 10 minuti, questo tempo è variabile)

4.5.2 Mining attack

4.5.2.1 Race attack

Miner malevolo aggiunge blocchi alla sua blockchain privata ma NON effettua un broadcast agli altri nodi della rete, una volta superata la lunghezza della chain legittima pubblico l'intera catena agli agli nodi della rete, questi seguendo il governance model accettereanno questa nuova catena come catena principale in quanto più lunga. Il limite di tempo per l'aggiunta di blocchi cerca di risolvere questo problema in quanto rende più difficile creare catene più lunghe offline.

4.5.2.2 Goldfinger attack

Occorre quando un solo miner possiede più del 50% del potere computazionale della rete. In questa situazione il miner può convalidare autonomanente transazioni anche malevole (in più del 50% dei nodi deve accettare un blocco per poter aggiungerlo alla rete).

4.5.3 Wallet attack

Problema di sicurezza di tipo endpoint. Difficoltà di storage delle chiavi per evitare furti e perdita, possibili soluzioni sono:

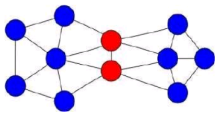
- Crittografare la chiave privata con una password: Suscettibili a dictionary attacks
- Splitting della chiave su più dispositivi: Utilizzo della multisignature feature di bitcoin
- Conservare su dispostivi esterni
- Utilizzare dispositivi smart creati ad hoc-

4.5.4 Network attack

4.5.4.1 Partitioning attack

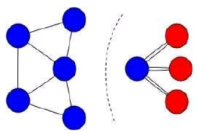
Divisi in due attacchi più piccoli:

- Partitioning attack: L'attacker prova a dividere la rete in due o più gruppi disgiunti (non possono comunicare tra loro). Fatto compromettendo i punti della rete che fungono da collegamento tra i due gruppi.
- Delay attack: L'attaccante prende le informazioni propagate da uno dei gruppi della rete e le propaga all'altro gruppo dopo averle modificate.



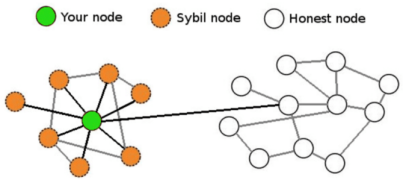
4.5.4.2 Eclipse attack

I nodi malevoli isolano uno dei nodi onesti della rete monopolizzando le sue conenssioni



4.5.4.3 Sybil attack

Uso un singolo nodo che opera una serie di identità fake (nodi fasulli, sybil nodes) simultaneamente. Questo permette di compromettere la sicurezza della rete guadagnando maggiore potere e influenza sulla rete (maggiore numero di nodi votanti). Questo attacco può bloccare alcuni utenti nella rete (out voting nodi onesti e rifiutando di fare broadcast o ricevere blocchi) e portare un attacco di tipo goldfinger. Il proof of works di bictcoin rende difficile un attacco di tipo sybil.



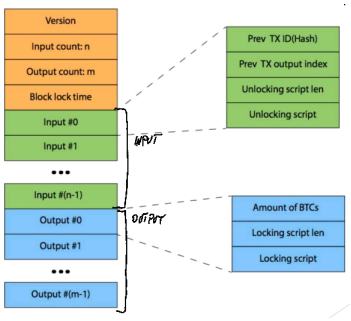
4.5.5 Sicurezza di bitcoin

I principali problemi di bitcoin risiedono nell'utilizzo della moneta per scambi nel mondo reale, sono necessari infatti protocolli di secondo livello per la sicurezza (wallet block, tool esterni). Possiamo notare come il meccanismo di consenso, proof of work, e limite temporale dei blocchi permettono di:

- Aumentare la sicurezza della rete
- Prevenire gli attacchi
- Gestire l'hashing rate

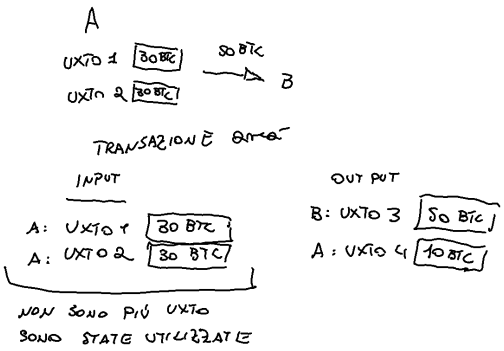
4.6 Transazioni

Le transazioni vengono gestite tramite un liguaggio a stack. Questo linguaggio è una macchina intermedia (linguaggio di tipo 2).

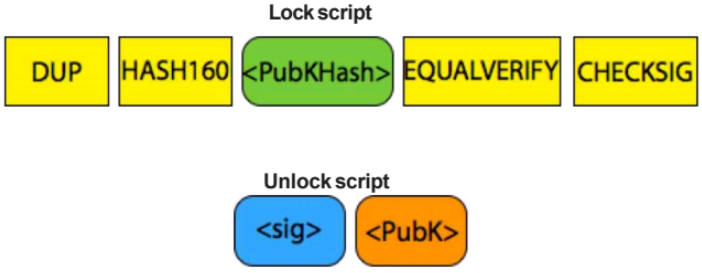


4.6.1 Pay to Public Key Hash (P2PKH)

Gli input di una transazione sonoi le reference alla precedente UXTO (spiegata nel paragrafo del portafoglio). Quando una UXTO viene impiegata come input in una nuova transazione questa viene utilizzata interamente (non posso usare ad esempio metà dei bitcoin presenti in una UXTO). Se A vuole inviare 50 BTC a B e utilizza due UXTO da 30 BTC l'input corrisponderà a questi due "cassetti" mentre l'output sarà una nuova UXTO di proprietà di B dal valore di 50 BTC e una UXTO di proprietà di A dal valore di 10 BTC (resto della transazione).



In generale una transazione utilizza la chiave pubblica del destinatario per l'invio del bitcoin e la chiave pubblica del mittente verificata tramite la chiave privata per l'invio di moneta. La verifica avviene tramite uno script di blocco e uno script di sblocco.



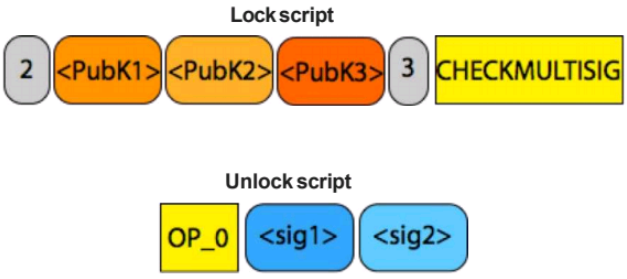
Lo script di sblocco è concatenato all'inizio dello script di blocco e eseguito dalla macchina stack, se il risultato dell'operazione è true la transazione è valida. Nello script si confronta se la chiave pubblica che vuole accedere ai fondi è uguale alla chiave pubblica che ha lockato la transzione e poi verificato l'accesso con la chiave privata.

4.6.2 Pay to Public Key (P2PK)

In questo caso lo script di locking contiene solo la chiave pubblica e l'operazione di controllo della signature e lo script di sblocco contiene la chiave privata (signature)

4.6.3 Multisig

Utile in situazioni critiche in cui è necessario un consenso minimo di k signature su n.



4.6.4 Data Output (OP RETURN)

Utilizzato per caricare dati sulla blockchainl, contiene un operazione di return e il dato da caricare. L'accesso alla risorsa è pubblico o si possono definire script con criteri di accesso. Il limite di spazio del dato è la grandezza del blocco (ad oggi 1MB).

4.6.5 Transazioni non standard

- OnlyHash: Presenta solo un lock script con l'hash di qualcosa
- CheckLockTimeVerify OP DROP (CLTV): Bitcoin non spendibili fino a che non passa del tempo o non sono stati aggiunti n blocchi alla rete. Lo script di lock contiene il dato, il controllo sul tempo e l'operazione di drop. Lo script di unlock contiene solo True.
- UnLocked (UL): Transazione senza locking, dato accessibile a chiunque.

Capitolo 5

Ethereum

5.1 Ethereum

Evoluzione della tecnologia originale Bitcoin. Il vantaggio più grande è la feature degli smart contract sulla blockchain, con un meccanismo di consenso che permette di eseguire il codice dei contratti su tutti i nodi della rete. Gli smart contract sono decentralizzati, replicabili, deterministichi, processati su ogni nodo della rete e valutati da ogni nodo (tutti i nodi devono essere d'accordo sul risultato). Ethereum è una macchina a stati distribuita, in cui lo stato globale è lo stato di tutti gli smart contract e le transazioni cambiano lo stato globale.

5.1.1 Moneta

La valuta della blockchain Ethereum si chiama Ether. È possibile frazionare Ether fino al wei (10 alla -18 Ether). A differenza di Bitcoin, la quantità di Ether generabili non ha limiti, questo è uno dei motivi per cui la piattaforma Ethereum viene criticata: c'è una continua inflazione infinita. Per mitigare questo effetto è stato via via ridotto il premio corrisposto ai miner modificando il protocollo; inoltre è stata fatta una modifica al costo del gas in modo tale che l'ecosistema Ethereum consumi giornalmente più Ether di quello che viene generato; inoltre, per ogni transazione, parte delle fee vengono distrutte.

5.1.2 Differenze con Bitcoin

Nonostante il linguaggio di scripting di Bitcoin possa essere utilizzato per scrivere smart contract ma questo linguaggio non è turing completo e non presenta variabili di stato arbitrarie.

	Ethereum	**Bitcoin**
Turing completo	Si	No
Variabili di stato	Si	No
Smart contracts	Si	No
Storage distribuito	Si	Si
Computazione distribuita	Si	No
Similarità	Basati su blockchain Blockchian pubblica e permissionless Proof of work mining Cryptovaluta nativa (BTC e ETH)	
Utenti	Basato su account, tengo traccia del saldo di un utente, le transazioni modificano esplicitamente il saldo di un account	Basato su address UXTO, saldo caricato implicitamente sulla blockchain somma delle UXTO
Ricompense ai nodi uncles	Si	No
Mining	Ethash, proof of work con difficoltà in termini di memoria	Proof of work con alta difficoltà computazionale
Altro	Utilizzo di gas per l'esecuzione del codice dei contratti	

5.1.3 Smart contract

Un protovollo di transazione computerizzato che esegue i termini di un contratto. L'obiettivo di uno smart contract è soddisfare le condizioni contrattuali comuni, minimizzare le eccezzioni e minimizzare il bisogno di intermediari fidati. Ulteriori obiettivi sono ridurre il numero di frodi e i costi di gestine e delle transazioni.

Rispetto ad un normale contratto uno smart contract traduce le clausole contrattuali in codice, risulta quindi più funzionale e riduce i costi dei contratti cartacei. Essendo del codice eseguito rende pià difficili per malintenzionali di compromettere e non seguire le clausole contrattuali. L'utilizzo di crittografia fa in modo che le relazioni stipate nel contratto non siano violate e assicura che i termini del contratto siano soddisfatti.

Può essere visto come un codice che automatizza la parte "se succede questo fai questo" dei normali contratti, fornendo un metodo deterministico che si esegue esattamente ciò che è stato scritto per fare e non presenta i problemi linguistici del linguaggio umano.

5.1.3.1 Smart contract distribuiti

- Trasparenza: Tutti i partecipanti della blockchain eseguono lo stesso codice, verificandosi l'esecuzione a vicenda. La logica degli smart contract è visibile a tutti. La privacy potrebbe essere un problema, in questi casi si potrebbero usare soluzioni di dimostrazioni a zero conoscenza (dimostro che una affermazioni è vera senza mostrare la affermazione in se).
- Flessibilità: Scrittura di contratti in un linguaggio turing completo, devo però pagare tutti i nodi della rete per eseguire il mio codice in parallelo, un programma scritto per ethereum sarà più pesante da eseguire che su un normale computer.

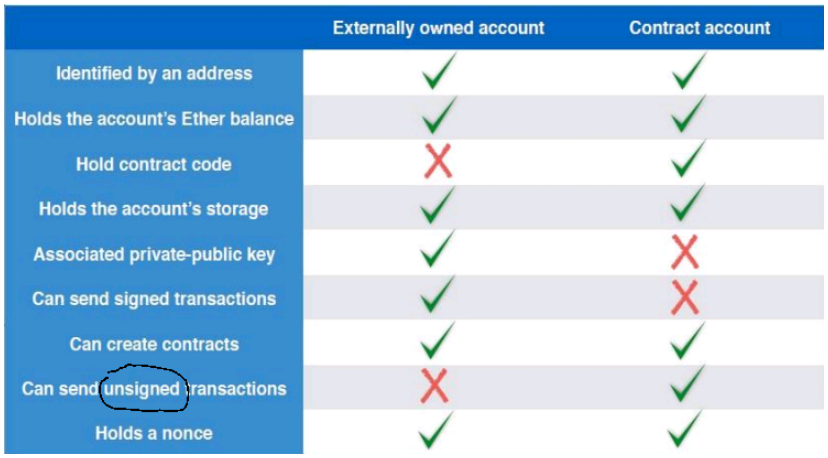
Bisogna tenere conto di possibili problemi di DOS.

5.2 Account

Lo stato globale è formato da tutti gli account e di piccoli oggetti che interagiscono tra loro tramite passaggio di messagi. Un account ha uno stato è un identificatore a 160 bit. Esistono due tipi di account:

- Externally owned: Chiavi private controllate con nessun codice associato. Sono controllati da una entità esterna, l'accesso ai fondi e la possibilità di inviare transazioni e esecuzione di contratti richiede la chiave privata dell'account. Possono inviare transazioni per inviare Ether ad altri account esterni (o contract account) e triggerare l'esecuzione di smart contract. Possiedono un address e il bilancio di ether.

- Contract accounts: Saldo con codice associato. Il saldo e l'account sono controllati dal codice. Contiene codice eseguibile, dati associati e un bilancio dell'account. Il codice è l'owner dell'account, controllato dalla logica del codice del contratto. Viene eseguito sulla Ethereum Virtual Machine (EVM). Contiene un address, il codice contrattuale, storage persistente, un balancio di ether ma non possiede una chiave privata.



Unsigned significa che non si verifica l'iniziatore. I contrattni hanno un attivatore e creatore del contratto che permettono di risalire al mandante.

5.2.1 Comportamento di un contract account

Un contratto può essere triggerato da una transazione o daun messagio. Quando una transazione è diretta ad un indirizzo di contratto questo viene eseguito nella EVM. Le transazioni possono chiamare eplicitamente funzioni dentro un contratto e contenere ether da trasferire al contratto e dati (parametri di input alle funzioni del contratto). I contratti possono inviare e ricevere ether, come gli account EOA. Questi account non hanno una chiave privata e non possono iniziare una transazione, loro reagiscono alle transazioni e ai messagni, possono chiamare altri contratti manon se stessi. Possono generare transazioni come risposte alle transazioni ricevute da EOA o altri contratti.

5.3 Transazioni

Ogni azione sulla blockchain etherum è messa in moto da una trasnazione eseguita da un EOA. Una transazione è un pacchetto di dati firmato che contiene un messaggio. Una transazione è il ponte di collegamento tra il mondo esterno e lo stato interno di ethereum.

- EOA a EOA: Scambio di denaro

- EOA a CA:
  - Scrivi su storage interno
  - Esegui calcoli
  - Manda messaggi ad altri CA tramite transazioni interne.
  - Crea nuovi contratti

5.3.1 Transazioni interne

Come delle transazioni ma prodotte da CA, possono triggerare l'esecuzione di codice associato al contratto. Le transazioni interne non hanno limite di gas, in quanto questo è definito dalla transazione che ha avviato il contratto, il gas limit infatti non riguarda solo l'esecuzione del singolo contratto ma anche delle sotto esecuzioni che il contratto potrebbe fare come i messaggi da contratto a contratto. Se nell'esecuzione dei messaggi interni un particolare contratto finisce il gas, l'esecuzione di quello specifico messaggio verrà ripristinata insieme a tutte le eventuali sottochiamate fatte dal contratto. L'esecuzione parent potrebbe non necessariamente effettuare il revert.

5.3.2 Componenti transazione

Una transazione contiene:

- nonce
- gasLimit e gasPrice: Visti più avanti
- to: Destinatario
- value: Valore da inviare
- v,r,s: Valori della signature
- data: Dati inviati

5.3.2.1 nonce (account)

Il nonce è un auto increment delle transazioni di uno speficico account. Numero delle transazioni partite (non si contano le ricevute). Nel caso di contract account è il numero dei contratti crati da quell'account. Il nonce è quindi un attributo del mittente non della transazione. Utile per registrare l'ordine delle transazioni e proteggere dalla duplicazioni delle transazioni, queste infatti vengono eseguite in ordine di nonce, se una viene bloccata a causa di starvation le successive verranno eseguite soltanto dopo. Non è possibile cambiare il nonce di una transazione senza invalidare la signature della transazione.

5.3.2.2 to

Indirizzo ethereum a 20-byte (chiave pubblica), può esere un EOA o CA. Non viene eseguita la validazione dell'indirizzo, se non è valido gli ether inviati saranno persi. Rispetto a bitcoin il valore da inviare è inserito direttamente nella transazione, non è necessario fare la reference a UXTO precedenti.

5.3.2.3 value e data

Questi valori possono essere significativi o null:

- Value significativo: Un pagamento
  - Se il destinatario è un EOA viene aggiunto value al balance dell'account destinatario
  - Se il destinatario è un CA viene eseguita la funzione definita nel data pauyload o una funzione di callback (la funzione deve essere pagabile) se non è pagabile il valore di value viene aggiunto al bilancio del contratto.
- Data significativo: Una chiamata a funzione
  - Se il destinatario è un EOA è possibile eseguire questa istruzione ma non è specificata nel protocollo ethereum
  - Se il destinatario è un CA il data payload contiene il selettore di funzione (primi 4 bytes del Keccack-256 hash del prototipo di funzione per la non ambiguità) e i parametri di funzinoe codificati seguendo le specifiche ABI. Il data payload può anche contenere il bytecode compilato che permette di creare un nuovo contratto.
- Data e value significativi: Pagamento e invocazione di funzione

5.3.3 Gas

Il gas è un concetto introdotto per risolvere l'halting problem. Senza il gas infatti un contratto potrebbe avere codice che non termina mai la sua esecuzione in quanto la EVM è una macchina a singolo tread senza scheduler, se finisce in loop, tutti i nodi della rete saranno in loop. Il gas permette di terminare l'esecuzione del codice dopo un certo ammontare di istruzioni e il codice non si è ancora fermato autonomamente. La EVM è quindi una macchina quasi turing completa che definisce una soglia computazionale dinamica all'esecuzionedel codice. Questa soglia è il gas una unità di misura sia della computazione massima sia dello storage massimo occupabile. L'esecuzione di uno smart contract non è gratis ma necessita di un pagamento di ethereum pari al costo in gas delle varie istruzioni.

- GAS: Unità di misura delle fee per ogni computazione.

- GAS limit: Sovrastisma del gas totale necessario per l'esecuzione del codice.
- GAS price: Quanto si è disposti a pagare per ogni unità di gas.
- GAS limit * GAS price: Ammontare di gwei che il mittente è disposto a pagare per l'esecuzione del codice

Il pagamento di questa somma è la ricompensa al miner per l'esecuzione del codice e la validazione della transazione. Più è alto il gas price più è alta la probabilità che i miners selezionino la transazione (Recentemente questo meccanismo è stato sostitutito con una fee base e una mancia). Quando una transazione viene eseguita:

1. La quantita di ether definita da GAS limit * GAS price viene rimossa dall'account del mittente
2. Il miner inizia ad utilizzare il GAS fino alla fine dell'esecuzione o finchè non finisce il GAS a disposizione
3. Alla fine della transazione se è rimasto del GAS questo viene restituito al mittente allo stesso prezzo iniziale
4. Se invece finisce il gas a disposizione prima del termine dell'esecuzione si applica un REVERT STATE in cui ogni cambiamento viene ripristinato. Non vengono però restituiti gli ether, in quanto il miner ha comunque speso potere computazionale per eseguire il codice

5.4 Macchina a stati Ethereum

Lo stato di Ethereum è lo stato di tutti i suoi account, l'intero network è in accordo sullo stato di ogni account (sia EOA sia CA). Lo stato della rete è aggiornato quando un blocco viene minato.

5.4.1 Mining

Miners aggiornano lo stato della macchina distribuita aggiungendo blocchi alla rete eseguendo le transazioni inserite in esso e i relitivi codici dicontratto. Un blocco contiene l'hash del blocco prcedente, il set di transazioni, la radice del nuovo stato e la ricevuto del nuovo stasto e il nonce. Il mining di un blocco prende in input lo stato corrente e producono un nuovo stato. Ogni nodo deve essere in accordo sul nuovo stato.

5.4.2 Macchina distribuita

Essendo una macchina a stati distribuita ogni replica possiede la macchina stati e ogni replica deve essere nello stesso stato dato lo stesso input. La transazione permette di prendere in input uno stato e produrre un nuovo stato (state transition function)

5.4.3 Stato account

- nonce (account)
- balance
- storageRoot: hash del nodo radice di un albero Merkle Patricia (struttura dati hashing tree) usato per la codifica degli hash dei contenuti dell'account
- codeHash: contiene il codice del contratto (utilizzato solo per CA)-

5.4.4 Propagazione delle tranzazioni

Si utilizza un protocollo di flooding per la propagazione delle transazioni. Una transazione validata da un nodo viene trasmesa a tutti i nodi ethereum direttamente connessi al nodo originario (in media viene propagata a 13 nodi). Ogni nodo esegue a sua volta le stesse azioni.

Ricevuta Conserva i cambiamenti portati dalla transazione e gli eventi da essa generati

5.5 Mining e consenso

5.5.1 Ethash

Il protocollo di consendo di Ethereum necessita di essere resistente alle ASIC (Application Specific Integrate Circuit) processori special purpose per il mining della moneta che permettono di ridurre l'energia consumata. Per risolvere questo problema la funzione di hashing di Ethereum (ethash) richiede l'accesso ad una struttura dati a grafo molto grande chiamata DAG (directed acyclic graph) che crea un collo di bottiglia sulla memoria necessaria a calcolare l'hashing. La computazione passa quindi a GPU con requirement di memoria molto grandi (maggiori di 3-4 GB)

1. Creo seed a partire dall'header block
2. Credo una cache da 16 MB (array di dati a 64 bit)
3. Creo un dataset di 1 GB a partire dalla cache
4. Prendo random slices del dataset
5. Hashing degli slice
6. Successivamente posso utilizzare la cache per generare i dati utilizzati del DAG.

5.5.2 GHOST

Ethereum riduce il rateo di creazione dei blocchi da 10 minuti a 10 secondi portando ad una grande quantità di fork e aprendo la strada a possibili attacchi di tipo double spending e scartando una grande quantità di fork non minati, portando a molti miner che sprecano le loro risorse senza vedere i propri blocchi aggiunti alla rete. Il criterio di scelta della catena più lunga non può essere utilizzato in quanto serve molto meno de 51% del potere computazionale totale per creare una catena più lunga illecita. Si utilizza il ghost procotol (Greedy Heaviest Obeserved SUbtree):

- La catena viene scelta in base al sottoalbero più pesante. Più lavoro computazionale svolto contando anche il lavoro svolto nei fork vicini alla catena principale
- La ricompensa del mining viene data anche agli ommer block, stale block che sono inclusi nel calcolo per la scelta della catena ma fanno parte di un fork scartato.

Viene quindi imposto un limite alla profondita dell sottoalber più pesante con un numero massimo di ancestor e ommer block per ogni blocco minato. In questo caso il limite necessario per considerare un blocco immutabile è di 60 blocchi (1h)

Ommer Blocco che non può essere un ancesto del blocco minato, non può essere stato incluso comee ommer in passato, e deve essere un figlio dell'ancesto del nodo minato a profodintà da 2 a 7. Riceve il 87.5% del premio che avrebbe ricevuto ma non le fees (il nephew riceve la restante parte).

5.5.3 Proof of Stake

Nasce dalla necessità di ridurre il consumo di energia del mining proof of work, anche chiamato virtual mining, dipende non dal lavoro svolto dai minre da uno stake economio messo a disposizione dal validatore. Utilizza la stessa moneta come risorsa per prevenire gli attacchi.I validatori mettono a disposizione i loro token (stake) per entrare nella lotteria per la produzione del prossimo blocco La probabilità di essere selezionato è proposioznale al numero di token messi a disposizione (stake)La coing age (da quanto tempo si hanno a disposizione i token) determina la difficoltà del proof of work. Necessario un valore minimo di coinAge per essere un validatore.Coin Age si resetta dopo aver validato un blocco permettendo di creare una rotazione in chi valida la catena (prevenzione di attacchi goldfinger). La scelta del miner avviene in base a:

- Estrazione casuale pesata
- Proof of work con difficoltà dinamica (basata su coin age)

I token messi a disposizione sono chiamati stake in quanto se la transazione che ho validato è illecita e non viene accettata dagli altri nodi della rete tutti i token messi a disposizione vengono persi.

5.6 Codice

Il linguaggi di programmazione adottato è Solidity (ma esistono alternative) un linguaggio ad oggetti simile a Javascript. Viene compilato in bytecode EVM prima di essere caricato sulla blockchain

5.6.1 Dettagli codice

- Non si possono usare float per evitare differenze di approssimazione.
- Costruttore non chiamabile.
- Il public indica che il valore è direttamente leggibile dall'esterno, spesso si utilizzano dizionari e insiemi.
- I dizionari ritornano 0 per chiavi non esistenti.
- Numeri random hanno come seed l'id del blocco, in questo modo tutte le macchine hanno lo stesso id e generano lo stesso numero.

5.7 Curiosità e dettagli

Strumenti come METAMASK permettono funzionalità aggiuntive quali:

- Rimozione di una transazione effettuando una transazione nulla verso noi stessi con stesso nonce ma fee più alta
- Velocizzare la transazione corrente (viene emessa una nuova transazione con stesso nonce ma fee più alta)

Capitolo 6

MongoDB

6.1 Introduzione

6.1.1 Partitioning e sharding

Il partitioning di un database è il processo di divisione delle tabelle di un database in parti più piccole. Dividere una tabella in più tabelle di dimensioni inferiori consente di memorizzare un volume di dati che eccede le capacità di una singola macchina, ma anche di avere query più veloci e di diminuire quindi i tempi di risposta del sistema. Il partitioning si divide in orizzontale e verticale: il partitioning orizzontale consiste nella suddivisione dei dati su più macchine, il partitioning verticale consiste nella suddivisione di una tabella in più partizioni, ogni partizione avrà come colonne un sottoinsieme proprio delle colonne della tabella originale. Lo sharding è un tipo di partitioning orizzontale, consiste nel replicare lo schema su più macchine e dividere i record in shard, assegnando ogni record ad uno shard a sulla base della "shard key".

6.1.2 Scaling di un database

- Scaling UP: Macchina più grande
- Scaling OUT: Partizionamento dei dati su più macchine.

6.1.3 MongoDB

Database document-oriented, non relazionale creato sullo scaling out e partizionamento orizzontale. Basato su coppie chiave-valore simili ad un json. I vantaggi principali sono avere documenti che corrispondono ad oggetti di tipi di dato nativi in molti linguaggi di programmazione, l'utilizzo di documenti e array embedded riduce la necessità di join complessi e gli schemi dinamici supportano il polimorfismo (schema less, il codice definisce lo schema).

- Collection: Insieme di documenti
- Document: Unità atomica, corrisponde alla riga di una tabella. I documenti in una collezione possono avere diverse forme.
- Field: Valore conenuto nel documento, corrisponde alla colonna di una tabella. Valori espressi in formato BSON.
  - _id: Identificatore univoco, può essere qualsiasi tipo (default ObjectID). Automaticamente generato se non presente utilizzando timestamp, un valore randomico e un counter incrementale.

6.2 Operazioni

6.2.1 Inserimento

Tutte le operazioni di scrittura sono atomiche a livello di singolo documento. Nell'inserimento MongoDB controlla solo l'unicità dell'id e la grandezza.

- insertOne: inserisce un singolo documento
- insertMany: inserisce documenti multipli in una collezione

6.2.2 Ricerca

find: Ricerca di documenti nella collezione, ritornaun cursore (iterable), un puntatore al documento che soddisfa i requisiti della query.

- Criterio di selezione:
  - Equality: $\{< field >:< value >\}$
  - In: $\{< field >: \{\$in : [< value >, ..., < value >]\}\}$
  - Altro: $\$eq, \$gt, \$gte, \$le, \$ne, \$nin(notin)$ etc, etc
  - Criteri di selezione multipli si dividono con virgola $\{< field >:< value >, < field >:< value >, ...\}$
  - Operatori logici: $\{\$logop : [\{< field >:< value >\}, \{< field >:< value >\}]\}$
  - Array sono cittadini di prima classe, l'operazione di equivaglianza di un array è una contains, per usare un matching esatto usare le parentesi quadre.
- Projection: Permette di includere field specifici (set a 1) o rimuovere field specifici (set a 0)
- Modifiche al cursore:
  - .sort: Ordinamento
  - .limit: Limita risultati. Utilizzato per paging
  - .skip: Non stampa primi n risultati. Utilizzato per paging.

6.2.3 Aggiornamento

- update: Ricerca un documento in base al nome e lo sostituisce interamente con il documento nel secondo parametro. Se modificare un field utilizzare $\$set$
- $inc: Incrementa il field del documento trovato.
- upsert: Parametro che permette di creare un documento se questo non viene trovato

6.2.4 Aggregazione

La pipeline di aggregazione permette di trasformare e combinare documenti in una collezione. Gli operatori vanno inseriti in un array separati da virgola e verranno eseguiti in ordine. Ogni operatore riceve uno stream di documenti. Una volta raggiunto l'ultimo operatore nella pipeline vengono mostrati i risultati.

- $match: Filtra i documenti
- $group: Raggruppa i documenti sulla base di un id (effettua aggregazioni)
- $lookup: Simile ad un left outer equi join. Permette di eseguire un join di collezioni quando sono presenti documenti embedded.

Ulteriori operatori:

- .count: Numero di documenti nella collezione che soddisfano la query
- .distinct: Tutti i valori distinti per una data chiave

6.3 Replication

Replicazione dei dati su più nodi per forninre ridondanza dei dati e incrementare l'availability. Un replicaset è un gruppo di processi mongod che mantengono lo stesso dats set. Un replica set contiene una sere di nodi che contengono dati e un nodo arbitro, uno di questi nodi è primario e gli altri sono secondari:

- Primario: riceve operazioni di scrittura e produce OPLOG,
- Secondario: Riceve gli OOPLOG e applica le operazioni ai dataset. Per le operazioni di lettura è possibile definire delle preferenze sul server secondario.

Membri di un replica set si inviano dei hearthbeat ogni due secondi, se qusto ping non arriva entro 10 secondi quel nodo è definito inacessibile. Se il nodo inacessibile è un nodo primario inizia una elezione per decidere chi è il nodo primario. Si possono settare nodi secondari in modo che non possano diventare nodi primari. Un nodo arbitro è unm nodo che non ha copie del dataset e non può diventare primario, partecipa però nell'elezione del primario, usato per far funzionare le elezioni con un numero pari di server.

6.4 Sharding

Scaling orizzontale dei cluser. Se la replication serve all'alta disponibilità questo permette di effettuare una distribuzione del carico. Uno shard contiene un subset del dato partizionato (è un replica set). I mongos sono dei router delle query, creando una interfaccia tra client e shard. I server di configurazione (sempre replica set) contengono metadata e i setting del cluster. Un database può avere

un mix di sharded e unsharded collections, quest'ultime sono caricate sullo shard primario. MongoDB partiziona il dato a livello di collezione, distribuendo i dati della collezione tra i diversi shard. Questo avviene utilizzando la shard key un field indexato che viene diviso in range non sovrapponibili, successivamente ogni range è associato ad un chunk di dati. MongoDB cerca di distribuire i chunk in maniera equa tra tutti gli shard del cluster.

- Ranged Sharding: Dati divisi in range basati sui valori della shard key. Con questo metodo un range di key che sono vicini tra loro hanno più probabilità di finire nello stesso chunk, questo permette di dirigere le operazioni solo verso gli shard che contengono i dati richiesti. Può portare ad una distribuzione non uniforme dei dati.
- Hashed Sharding: Consistent Hashing

Capitolo 7

Apache Spark

7.1 Introduzione

7.1.1 MapReduce

MapReduce è un paradigma di programmazione introdotto da Google ispirato alle funzioni map e reduce utilizzate nella programmazione funzionale. In un programma scritto utilizzando questo paradigma ci saranno quindi due funzioni principali: Map e Reduce. Nella Map vengono processate coppie chiave-valore e restituite coppie chiave-valore; nella Reduce vengono processate le coppie chiave-valore prodotte dalla Map. Questo paradigma viene utilizzato nei sistemi distribuiti perché consente di separare la computazione in due momenti: nella Map ogni nodo lavora in autonomia e produce risultati intermedi che vengono poi aggregati insieme ai risultati intermedi degli altri nodi nella Reduce.

- MAP: Prende in input chiave e valore e produce un valore intermedio. Map invocabile anche con chiavi distribuite su più nodi, ogni nodo lavora in modo autonomo.
- REDUCE: Aggrega i valori intermedi in un set più piccolo di valori. I valori intermedi vengono dati in input sotto forma di iteratori.

In sistemi MapReduce come Hadoop le chiavi intermedie vanno nel file system.

7.1.2 Spark

Cluster computing engine progettato per i big data e il in memory processing. Rispetto alla libreria MapReduce permette di avere pattern generalizzati, lazy evaluation dei lineage graph, minore overhead dei starting job e ordinamenti meno costosi. Spark permette di risolvere il bottleneck di Hadoop (pesanti operazioni su disco) distribuendo tutto in memoria centralie.

7.2 Resilient Distributed Datasets (RDD)

Struttura dati principale utilizzata in spark. Rappresenta una collezione di oggetti partizionata in cluster, salvata in memoria o su disco. Collezione di dati omogenei che non contiene duplicati che non segue un ordinamento. Robusto alla perdita di nodi, spark permette di ricostruire autometicamente le partizioni perse. Gli RDD operano in sola lettura, la modifica di un RDD comporta infatti la creazione di un nuovo RDD, che sarà automaticamente distribuito perché ogni partizione avrà il risultato dell'operazione di modifica sui dati in suo possesso. Le operazioni che possiamo effettuare su un RDD sono di due tipi: trasformazioni, che trasformano un RDD in un nuovo RDD e azioni, che applicano le trasformazioni sull'RDD e restituiscono i risultati.

7.2.1 Trasformazioni

Le trasformazioni sono operazioni che trasformano un RDD in un nuovo RDD. Usano la "lazy evaluation", ossia le operazioni definite dalla trasformazione non vengono effettuate finché questo non è realmente necessario, ossia finché non viene chiamata un'azione che necessita dei risultati delle trasformazioni richieste. Le trasformazioni in Spark sono:

- MAP: Prende in input un RDD e una funzinoe e restituisce un nuovo RDD contenente un item per ogni item dell'RDD originale. Questi item saranno gli item dell'RDD di input a cui sarà applicata una funzione.
- FILTER: Prende in input un RDD e una funzione di filtraggio. Restituisce un RDD i cui elementi sono gli elementi dell'RDD di input su cui la funzione ha restituito true.
- DISTINCT: Prende in input un RDD restituisce un RDD con i duplicati rimossi. Grande lavoro in quanto i worker dovranno comunicare per scambiare informazioni sui duplicati.
- FLATMAP: Come la map ma in questo caso ogni item puà essere mappato a zero o più item in output. Permette di applicare una funzione map e una di filter ma l'unione di queste due non permette di replicare l'output di una flatmap.

7.2.2 Trasformazioni binarie

- UNION: Unione di due RDD. Mantiene i duplicati.
- INTERSECTION: Intersezione di due RDD. Rimossi i duplicati.
- SUBTRACT: Sottrazione di due RDD. Mantiene i duplicati.
- CARTESIAN: Prodotto cartesiano tra i due RDD

7.2.3 Trasformazioni Key-Value

Trasformazioni per RDD di tipo key-value, chiamiti Paired RDD (gli elementi in value devono essere omogenei):

- GROUPBYKEY: $(K, V) \rightarrow (K, iterable(V))$ Raggruppa gli item per chiave.
- REDUCEBYKEY: $(K, V) \rightarrow (K, V)$ Come la groupbykey ma necessit in input di una funzione di aggregazione che restituisce un valore a partire da una coppia di valori $(V_1, V_2) \rightarrow V_3$. La funzione deve essere commutativa e associativa. Il nuovo valore è una aggregazione dei valori di quella chiave. Usando la groupbykey un worker dovrà mantenere tutti i valori per una specifica chiave. Questo metodo permette di effettuare riduzioni successive su ogni nodo.

- AGGRGATEBYKEY: come reduceByKey, ma consente di restituire risultati di tipo diverso da quello di partenza.
- SORTBYKEY: utilizzata per ordinare gli item in base alla chiave, ovviamente gli item saranno ordinati per worker.
- JOIN: : chiamata su due dataset con item di tipo $(K, V)e(K, W)$ produce un dataset di tipo $(K, (V, W))$ contenente per ogni chiave le corrispondenti coppie di valori dai due dataset.

7.2.4 Azioni

Le azioni sono operazioni che applicano le trasformazioni sull'RDD e restituiscono i risultati. Le possibili azioni in Spark sono:

- REDUCE: applica una funzione a tutti gli elementi dell'RDD e restituisce il risultato
- TAKE: prende in input un parametro n; restituisce i primi n elementi. Se questi non sono stati ordinati in precedenza saranno a caso.
- COLLECT: è come una take ma senza parametro, restituisce tutto l'RDD nel master in una struttura dati non parallelizzata.
- FOREACH: esegue la funzione passata in input su ogni elemento dell'RDD.

7.2.5 Ulteriori funzioni

- BROADCAST: Invia grandi valori read only a tutti i worker.
- ACCUMULATOR: Aggrega i valori dei worker e invia al driver. Accessibile solo al driver.