Data Management

Alberto Filosa

6/1/2020

Contents

1 Data Modelling (Variety)

Il *data modelling* è uno strumento per descrivere parte del mondo reale che permette di immagazzinare dati, in inglese *write* data, ed interrogare i dati, in inglese *read* data. Esso è costituito da un modello concettuale, teorico, e da un linguaggio, anche grafico, per descrivere ed interrogare i dati. Inoltre, è sempre disponibile in un DBMS (**D**ata**B**ase **M**anagement **S**ystem) che supporta la semantica del modello. Si presentano le caratteristiche di un data modelling:

- *Machine readability*;
- *Expression power*;
- Semplicità, flessibilità e standardizzazione.

1.1 Data Lifecycle

I dati hanno un loro ciclo di vita: inanzitutto, bisogna definire il proprio obiettivo facendosi delle domande. Dopodichè, bisogna trovare e scaricare i dati. Esistono diversi modi per cercare i dati: possono essere interni, tramite interrogazioni di query, file di log o tramite strumenti IoT, oppure esterni, dai social network, dal web o dagli open data. I dati possono essere gratuiti o a pagamento; una volta trovati, però, bisogna identificare anche la modalità di cattura, tramite il download, una query SQL, utilizzando le API oppure tramite scraping. Appena scaricati, i dati sono grezzi, vanno manipolati e lavorati per ottenere delle ottime informazioni: le operazioni di pulizia sono varie e vanno utilizzati in base ai propri scopi. Si devono gestire gli errori di imputazione, gli eventuali NA (veri o i valori impossibili), gli outliers, ecc. Avere i dati puliti è necessario per ottener ottimi risultati. Generalmente si utilizzano più dataset da siti diversi per le proprie analisi: un metodo efficiente è l'integrazione dei dati. Esistono diversi approcci per come i dati sono stati progettati e per i propri scopri. Successivamente si compiono delle analisi esplorative e predittive. Infine, si presentano gli output tramite una rappresentazione grafica.

1.2 Relational Model

Il modello relazionale è il più conosciuto ed utilizzato al mondo. I principali vantaggi di questo tipo di architettura sono l'uso di uno schema rigido per immagazzinare i dati e sfrutta le proprietà ACID:

- ***A***tomicity: una operazione avviene per intero (commit) o restituisce un errore ed il database ritorna alla forma iniziale, con il fenomeno chiamato rollback;
- ***C***onsistency: i nuovi dati inseriti rispecchiano lo schema prestabilito, o l'operazione fallisce;
- ***I***solation: le singole operazioni non influiscono sulle altre. Di conseguenza, il database costruisce una coda dei processi in modo tale da non mutare il proprio stato;
- ***D***urability: la persistenza del file è garantita anche in caso di crash del sistema, tramite backup e file di log.

Essendo ancora ben sviluppato ed utilizzato, molti

database sono ancora salvati in questo formato. Tuttavia, i modelli relazionali presentano diversi svantaggi:

- Gli attributi possono avere solamente un valore;
- Non è compatibile con i nuovi linguaggi di programmazione ad oggetti;
- Non permette *loop* nei dati;
- La modifica delle tabelle è lunga e dispendiosa;
- Poco prestante in alcuni contesti.

In particolare, le performance, in questo caso inteso come velocità dei processi, dipendono dal numero di righe, dal tipo di operazione svolta, dall'algoritmo scelto e dalla struttura dei dati. Peer aumentare la velocità di esecuzione, è possibile compiere l'attività di *scaling*. Ne esitono di due tipi: ***Scaling Up***, nella quale si potenzia la singola macchina, ***Scaling Out***, nella quale si aggiungono più macchine in una rete di calcolatori. Quest'ultimo è molto difficile da attuare nei modelli relazionali.

1.3 NoSQL

Per risolvere i problemi dei database relazionali, sono stati costituiti i NoSQL (Not Only SQL). Pur essendo molto diversi tra loro, hanno caratteristiche comuni:

1. Non hanno alcuno schema prestabilito. Infatti, per aggiungere un nuovo attributo non è necessario modificare l'intero modello;
2. Si assume la concezione del mondo aperto: ciò che non è vero è sconosciuto, ma non per forza falso;
3. Seguono il teorema CAP, affermando che per un DBMS è impossibile fornire in modo simultaneo tutte e tre le proprietà:
   - ***C***onsistency: tutti i nodi vedono gli stessi dati allo stesso istante (se assente, si mostra quello più recente);
   - ***A***vailability: riceve una risposta in un tempo determinato per ogni richiesta;
   - ***P***artition Tolerance: il sistema funziona anche con dati frammentati su una serie di calcolatori.

   Il modello relazionale utilizza le proprietà CA, mentre i NoSQL generalmente CP o AP.
4. Seguono il principio BASE:
   - ***B***asic ***A***vailability: la consistenza non è garantita interamente, mostrando una parte dei dati disponibili;
   - ***S***oft State: i dati possono avere schemi diversi;
   - ***E***ventual Consistency: i sistemi NoSQL richiedono che i dati convergano ad uno stato consistente senza specificare quando. Prima ddella consistenza si possono avere anche valori non veritieri.

I modelli NoSQL si dividono in diverse macrocategorie:

- *Key-Value* (Dynamo, Voldemort): sono tabelle con chiavi che si riferiscono ad un certo dato. Questo modello è simile ai modelli Document Based;
- *Column Family* (Big Table, Cassandra): il modello è in grado di salvare grandi quantità di dati e la chiave-colonna si riferisse ad un certo dato raggruppato in colonna;
- *Document Based* (MongoDB, CouchDB): salvati in formato JSON, contiene insiemi di coppie <chiave-valore> (è facile ricercare dati in questo formato);
- *Graph Based* (Neo4J, FlockDB): sfrutta il concetto di grafo per archiviare i dati come nodi ed esaltare i legami tra di essi costruendo archi. Rapido nelle *query*, è difficile da scalare per quanto riguarda l'immagazzinamento.

La semplicità del modello permette di archiviare un maggior numero di dati. Inoltre, la ridondanza dei dati facilita le ricerche pur aumentando le dimensioni dei dati stessi: si sacrifica spazio di archiviazione a favore della velocità di esecuzione.

1.3.1 Document Based

MongoDB è un sistema di gestione dati basatoo sui documenti; i dati sono salvati in formato BSON, Binary JSON. La principale differenza con i modelli relazionali è che i Document Based non prevedono operazioni di *join*. Nel caso in cui si verifichi un massiccio caricamento di dati, è possibile apportare delle modifiche al processo; in particolare, è possibile disabilitare il riconoscimento dei dati, che fa parte del protocollo di comunicazione, e disabilitare la scrittura del file *log*. Per dataset grandi è utile l'utilizzo di indici, in modo tale da recuperare informazioni velocemente, ma così facendo rallenta l'inserimento di nuovi dati (il campo `_id` è sempre indicizzato).

Senza un indice, MongoDB analizza tutti i documenti, come avviene nei database relazionali.

L'indicizzazione evita proprio questo tipo di problema, organizzano il contenuto in una lista ordinata; è meglio usarne pochi per ogni *collection*. Il numero massimo di indici è pari a 64. L'aggregazione utilizza pipeline ed opzioni. La *pipeline* di aggregazione avvia l'elaborazione dei documenti della raccolta e passa il risultato a quella successiva per ottenre risultati. Inoltre, è possibile utilizzare lo stesso operatore in diverse *pipeline*.

1.3.2 Graph Based

Un grafo è una collezione di nodi ed archi, i quali rappresentano la relazione tra i nodi. In questo modo, è possibile utilizzarli in diversi ambiti, come *social media*, bio-informatica, sistemi di autorizzazione, ecc. Il modello a grafo con etichette, in inglese *labeled property*, presenta le seguenti caratteristiche: essa contiene nodi e relazioni, con i nodi che devono contenere delle proprietà ed essere etichettati con 1 o più tabelle. Le relazioni sono nominali e direzionali, cioè hanno un nodo d'inizio ed uno di fine; inoltre, anch'esse possono avere delle proprietà. Le proprietà delle relazioni possono assegnate con un criterio *fine-grained* o *generic*. Considerando il caso della relazione `ADDRESS`, si può fare distinzione tra:

- *Fine-grained*: `HOME.ADDRESS` o `WORK.ADDRESS` (etichette diverse);
- *Generic*: `ADDRESS:HOME` o `ADDRESS:work` (etichetta sempre la stessa, con un attributo che ne indica il tipo).

Generalmente è preferito il secondo approccio rispetto al primo, in quanto permette di scrivere *query* più facilemente. Un *Graph Database* può archiviare i dati nativamente com un grafo o secondo altri sistemi di archiviazione. Il primo ottimizza la gestione dei grafi, mentre il secondo archivia i dati in formato tabellare o documenti per integrare il database come se fosse un grafo. Ad esempio, il metodo tabellare archivia le relazioni su una tabella relazionale che può essere interrogata tramite *join bomb* (*join* con se stessa), tuttavia questo sistema degenera in fretta all'aumentare della distanza tra due nodi. Il database a grafo risolve quindi il problema dei database relazionali (e di molti altri database NoSQL) a gestire le relazioni interne ai dati. Infatti anche altri modelli NoSQL, indipendentemente dal modello in uso, soffrono perdite di prestazioni quando sono effettuate aggregazioni di dati (soprattutto non indicizzati) dal momento che i collegamenti non sono nativi nel modello e manca il concetto di prossimità. Il *DBA* (***D***ata***B***ase ***A***dministrator) in base ai suoi bisogni (integrazione con altre applicazioni) può benissimo decidere di usare un database a grafo con una gestione dei dati non nativa senza che questo impatti sulla qualità del prodotto finale.

Per eseguire una query nel modello a grafo, il tempo di risposta non dipende strettamente dal numero totale di nodi (che rimane più o meno costante) perché la *query* viene processata nella porzione locale del grafo connessa al nodo base, mentre nei modelli relazionali e altri modelli NoSQL peggiorano le prestazioni all' aumentare dei dati (spesso in una maniera lineare). Il processing engine usa *index-free adjacency*: i nodi connessi sono collegati fisicamente tra di loro, velocizzando il loro recupero da una *query*, ma l'efficacia di quelle che non sfruttano le proprietà del grafo viene peggiorata, ad esempio nel caso delle operazioni di *I/O*. Neo4j implementa un linguaggio *Cypher* di tipo dichiarativo, usando una sintassi simile a quella di SQL, ma ottimizato per i grafi. Il linguaggio è di tipo *Pattern-matching*, facile da imparare, leggere e capire, espressiva ma compatto, di tipo dichiarativo, ovvero bisogna dichiarare ciò che si vuole. Esso è fortemente influenzato da altri linguaggi sopratutto SQL, ma rimane comunque diverso abbastanza da capire che il modello utilizzato è a grafo. Le principali funzioni sono tipo aggregativo e di ordinamento. Alla fine è necessario aggiornare il grafo.

1.3.3 Key-Value Model

Considerato il modello più semplice e flessibile, associa ad una stringa, o chiave, una sequenza di dati binari, chiamati *BLOB* (***B***inary ***L***arge ***OB***ject). Le unice operazioni lecite sono l'aggiunta e la rimozione di una nuova coppia <chiave-valore>, la modifica e la ricerca di un valore data la chiave. Bisogna tenere in considerazione che non è possibile cercare la chiave dato il valore. Grazie all'indicizzazione ad *hash*, il modello riesce a scalare orizzontalmente in modo efficiente. Esso è una funzione matematica che assegna un valore ad una chiave: $h(x) = v$. Questi valori possono essere facilemente distribuiti su una rete di calcolatori di dimensione arbitraria. Per recuperare il dato, è necessario interrogare il nodo in cui è presente la chiave ed attendere che invii il valore.

1.3.4 Column Family

I dati sono raggruppati insieme nello spazio di memoria in colonne. In questa tipologia di database i valori

di una colonna diventano co-localizzati nello stesso blocco del disco. Il vantaggio principale è dato dal fatto che il processo di ricerca tramite una *query* è ottimizzato perchè tutti i valori sono nello stesso spazio del disco. Esistono due importanti tipi di database: Wide Column e Big Table, i quali sono una evoluzione dei Key-Value database. In particolare, quando aumenta la complessità dei dati, si immagazzinano in grandi colonne.

Big Table, introdotta da Google, è una mappa multidimensionale ed è possibile accedere ai valori tramite una chiave colonna o riga. I dati sono organizzati in tabelle e ognuna di esse è composta da database Column Family che includono colonne. Sono presenti due chiavi, rispetto alla riga e alla colonna. Le colonne contengono i dati ed il loro contenuto. Il timestamp supporta differenti versioni del databse in modo da osservare il cambiamento dei dati nel tempo ed è possiile accedervi all'ultimo senza alcuna confusione. Il numero seriale è unico per ogni riga, utilizzando come predefinito il timestamp. Questo odello è utile per le mappature i *-to-Many.

Cassandra, introdotta da Facebook, è molto imile ad HBase, con la differenza di storage dei dati e gli algoritmi di programmazione. La partcolarità è che ha una sola Column Family ed utilzza un linguaggio CQL (**C**assandra **Q**uery **L**anguage), simile ad SQL. Il *Keyspace*, il database in RDBMS, possiede nativamente un fattore ed un algoritmo di replicazione, ma rimane solamente un raggruppamento di Column Families. Esso non permette operazioni di *join* o chiavi esterne, quindi il database prevede l'uso di dati ridondanti e denormalizzati per gestire i collegamenti.

2 Data Quality

La qualità dei dati è definita secondo diversi aspetti, tra i quali Accuracy, Completeness, Currency e Consistency.

L'accuratezza, in inglese *Accuracy*, di un valore è definito come la vicinanza tra il valore stesso e la corretta rappresentazione del valore nel mondo reale. Esistono due tipi di accuratezze: *sintattica*, ovvero il grado per cui i vaori sono rappresentati correttamente, e *semantica*, ovvero il grado per cui i valori sono rappresentati nel mondo reale. Con essa è possibile misurare l'accuratezza dei valori alfanumerici. La metrica basata sulle stringhe adotta una funzione di distanza per la quale si misura la distanza tra il valore e quello del dominio di referenza. Questo processo è chiamato *edit distance*; la ED non normalizzata è definita come il numero di inserimenti/cancellazioni/sostituzioni di simboli alfanumerici. La ED normalizzata non misura la distanza, benì il suo complemento ad 1, cioè l'accuratezza di v1 rispetto a v2.

$$ED_{normalizzata}(v1, v2) = 1 - fracED(v1, v2)n \in [0, 1]$$

La completezza, in inglese *Completeness*, è una metrica della qualità del dato nella quale il fenomeno osservato è rappresentato nell'insieme dei dati. La misura di completezza per le tabelle è definita attraverso una riga, attributo, tabella od oggetti senza alcun valore mancante nelle righe. Fino ad ora si è assunto l'ipotesi di mondo chiuso, ovvero tutto ciò che è rappresentato è vero, il resto è falso. Una alternativa a questo è il mondo aperto, ovvero tutto ciò ancora non rappresentato non è falso.

La diffusione, in inglese *Currency*, misura con quale rapidità i dati sono aggiornati rispetto al corrispondente fenomeno del mondo reale. Come prima misura si prende in considerazione il ritardo temporale tra il tempo t_1 dell'evento del mondo reale che ha provocato la variazione e l'istante t_2 della sua registrazione, operazione molto costosa. Una seconda metrica utilizzata è la differenza tra il tempo di arrivo alla organizzazione ed il tempo in cui è stato effettuato l'aggiornamento tramite i file di *log*.

La coerenza, in inglese *Constistency*, dei dati presenta due diverse definizioni: la prima definisce la coerenza come con vincoli di integrità definiti sullo schema; la seconda, invece, definisce la coerenza come diverse rappresentazioni di uno stesso. I vincoli di consistenza possono includere un singolo o più attributi, attributi in più relazioni e vincoli espressi in termini di probabilità.

Il *Tradeoff* è il compromesso di utilizzo di tutte queste tecniche sulla qualità dei dati:

1. La consistenza e completezza nei modelli relazionali possono non essere conciliabii quando si vuole rispettare l'integrità referenziale;
2. Indiversi domini si privilegia l'accuratezza o completezza rispetto alla tempestività;
3. Nel Web la tempestività è fondamentale rispetto all'accuratezza o completezza.

2.1 Quality Assessement and Improvement

L'obiettivo principale del miglioramento dei dati è la pulizia di essi in modo da migliorare le performance successive. In questo ambito, si adottano due tipi di startegie: la prima, chiamata *data-driven*, nella quale un dataset di partenza viene modificato con una comparazaione di altri dataset simili. In questo modo è utile pulire le sorgenti dei dati, in modo tale da non modificare ogni volta dataset diversi, anche se è un processo molto complicato. La seconda, chiamata *process-driven*, ridisegna il processo di creazione dei dati. Esistono due principali tecniche utilizzate:

- *process control*, nella quale si inseriscono de controlli nel processo dei dati per prevenire la degradazione e la propagazione dell'errore;
- *process redesign*, nella quale si rimuovo le cause di bassa qualità dei dati.

3 Data Distribution

Un sistema di database centralizzato è un sistema in cui un database è localizzato, memorizzato e mantenuto in un'unica posizione. Se il processore fallisce, allora succede anche a tutto il sistema. Se è necessario avere i dati in luoghi diversi è possibile distribuirli o replicarli, in modo tale da eseguire anche operazioni in parallelo.. In un *Database Distribuito Omogeneo* tutti i siti hanno identico software concordano tutti a collaborare per l'elaborazione delle richieste dell'utente, che appare come unico sistema. In un *Database Distribuito Eterogeneo* diversi siti possono usare differenti schemi e software, creando però maggiori problemi per query e transazioni. Con la replicazione il sistema mantiene multiple copie di dati, che sono memorizzate in diversi siti, per un maggiore recupero e tolleranza degli errori con la frammentazione dei dati. Dividendo e frammentando il database in n diversi database, sorge il fenomeno *bottleneck*. I vantaggi della frammentazione sono molteplici:

- *Availability*, ovvero l'errore di un sito non determina l'indisponibilità dei dati, dato che una replica di essa si trova in un altro posto;
- *Parallelism*, ovvero le query sono processate da diversi nodi parallelamente più volte;
- *Reduced Data Transfer*, ovvero esistono repliche dei dati nel nodo stesso.

Esistono due principali svantaggi, ovvero dato che ogni replica deve essere aggiornata, i costi degli update aumentano; inoltre, si incrementa la complessità del controllo della rapidità di aggiornamento.

3.1 Fragmentation

Per *frammentazione* dei dati si intende la divisione di relazioni in frammenti che contengono sufficienti informazioni per ricostruire la relazione. Essa può essere *orizzontale* o *verticale*, l'importante è che contenga le informazioni per ricostruire la relazione. Nel primo caso, ogni tupla deve essere assegnata ad 1 o più frammenti, in modo da permettere processi paralleli sui frammenti e dividere i dati per allocare le tuple in posti di accesso facilitato. Nel secondo caso, lo schema è diviso in due diversi schemi più piccoli che devono contenere tra loro una chiave comune ed effettuare join successivamente. La frammentazione verticale permette alle tuple di essere divise in modo da allocarle in luoghi frequenti.

Le due frammentazioni possono essere utilizzate nello stesso database contemporaneamente e anche successivamente frammentati fino a quanto si vuole. Un aspetto molto importante è la *trasparenza* del dato che indica il grado per cui un utente non è consapevole di come i dati siano stati archiviati. Dal punto di vosta dei costi, il sistema centralizzato ha come criterio principale il conteggio degli accessi dei dischi. Invece, in un sistema distribuito bisogna considerare il costo per la trasmissione dei dati sulla rete e anche il potenziale guadagno in termini di performance. I principali vantaggi di un database distribuito sono il miglioramento di determinate caratteristiche, come la condivisione, l'autonomia locale, la disponibilità, ecc., e la riduzione dei costi dell'hardware. I principali svantaggi, invece, sono che l'architettura e la progettazione sono complesse e costose, la sicurezza e l'integrità dei dati sono difficilmente controllabili e si ha una mancanza nello standard nell'utilizzo.

Esistono 3 tipologie di strutture di DDBMS (***D***istributed ***D***ata***B***ase ***M***anagement ***S***ystem):

1. *Shared Everithing*: dominata dal mercato delle architetture fino al 2000, è presente un grande e costoso database che condivide ogni cosa, per cui molto lento; è il sistema centralizzato;
2. *Shared Disk*: i dati sono salvati su diversi dischi connessi tra loro. I dischi sono connessi e comunicano con le CPU;
3. *Shared Nothing*: ogni cosa è separata, ogni disco ha la propria CPU e non comunicano a vicenda.

Solo alla fine i processori vengono connessi insieme per combinare i risultati da ognuna di essi.

3.2 Replication

Un *log* è un file sequenziale immagazzinato in memoria stabile, archiviando tutte le attività in ordine cronologico. Con esso è possibile definire un checkpoint in modo da non definire un set di transazioni in un dato tempo. Il dump è una piena copia dello stato di un database in una memoria stabile. La sua esecuzione è offline e genera un backup; solamente terminato il dump le esecuzioni verranno scritte nel log. Esistono diverse strutture di replica:

- One2Many: 1 sorgente ed alcuni target;
- Many2One: alcune sorgenti e 1 target;
- Peer2Peer: dati replicati in molteplci nodi che comunicano tra loro;
- Bi-directional: Peer2Peer con solo 2 Peer;
- Multi-tier Staging: replicazione divisa in più fasi.

Per creare una replica è possibile prendere i dati dal master, che sono i backup o i dati stessi, e muoverli se non accessibili dall'esterno. Oppure si usa il file di log: per mantenere la replica uguale alla sorgente durante il trasferimento, si usa il log per esguire gli stessi comandi sulla sorgente.

4 Data Architecture (Volume)

La prima definizione di Big Data è stata coniata da Gartner nel 2012: > Big Data is high Volume, Velocity and/or high Variety information assets

Ad oggi è possibile aggiungere altre *V*, come Variability e Veridicity. Analizzare database in un singolo server è un processo lento e costoso. La soluzione è distribuire su più hardware meno costosi il lavoro di calcolo, in modo tale da avere risultati più veloci. I principali problemi, però, sono legati a diversi aspetti, come la presenza di punti morti (in inglese *deadlock*), la larghezza della banda e la coordinazione tra i nodi ed i casi di fallimento del sistema. I vantaggi nel distribuire su più computer i database sono la possibilità di scalare i dati linearmente, con il fenomeno chiamato *scale out*, l'attività di calcolo è rivolta ai dati e non viceversa e gestisce i casi di fallimento. Di seguito si presentano le varie architetture di distribuzione dei dati.

4.1 HDFS

L'HDFS (***H***adoop ***D***istribuited ***F***ile ***S***ystem) è un file sistem distribuito progettato per girare su hardware base. La differenza principale con altri file system è di essere progettato per girare grandi moli di dati su macchine poco costose e di essere fortemente tollerante agli errori. Esso consiste in un singolo namenode e in server master che gestisce il file system, chiamato namespace, regolando gli accessi ai file dei client. Sono presenti un elevato numero di datanode che gestiscono lo storage legato ai nodi su cui vengono eseguiti. In particolare, un *namenode* esegue operazioni nel file system, ad esempio apertura/chiusura e rinominazione, e determina la mappatura dei blocchi reindirizzando le richieste di operazioni ai *datanode*. Questi ultimi memorizzano i file frammentati, di solito ogni 128 MB, e creano/eliminano/replicano dei blocchi sotto istruzioni del namenode. Infine, il *transaction log* registra le operazioni compiute sui file.

La strategia di piazzamento dei blocchi dei file tra i datanode è la seguente:

1. Si compie una replica sul nodo locale;
2. Si compie una seconda replica su una collezione di nodi remoti, chiamati *rack*;
3. Si compie una terza replica sullo stesso rack remoto;
4. Si compiono delle repliche distribuite in modo casuale.

I client leggerano successivamente il fie della replica più vicina a loro. Esiste un sistema per verifica la correttezza dei dati; infatti, HDFS implementa un controllo *checksum* del file generato per ogni blocco del file. Quando un client richiede l'accesso ad un file, verifica che i dati ricevuti da ogni datanode combacino con quello associato. Altrimenti, verrà recuperato da un altro datanode. Il namenode verifica inoltre che i datanode siano tutti funzionanti e gestisce gli eventuali malfunzionamenti. HDFS può gestire diversi tipi di dati, come i file *.csv*, *.txt*, *.json*, *.arro* e *.parqet*.

4.2 Map Reduce

Map reduce è un motore di calcolo distribuito scritto in stile funzionale ed eseguito in parallelo. Essa prende in considerazione due tipi di funzioni: la funzione di *map*, che spezza in partizioni i file per svolgere il compito assegnato più velocemente e la funzione di *reduce*, che combina assieme i risultati del

file frammentato. In questo modo, si risolvono diversi problemi di gestione/processo dei Big Data quali:

- Assegnare i lavori dei singoli worker;
- Cosa succede se ci sono più lavori che worker;
- Come aggregare i risultati parziali;
- Come scoprire se i worker hanno finito i propri task;
- Cosa succede se un worker muore.

L'architettura di Map Reduce è di tipo *master-slave*: il *master* gestisce la coda dei lavori, li suddivide in vari blocchi e ne notifica il termine; gli *slave* gestiscono i singoli lavori inviando informazioni al nodo master. La fase di Map esegue lo tesso codice su tanti record; i dati di input vengono trasformati in coppie <chiave-valore> e filtrati in un altro insieme di coppie dello stesso tipo. Quest'ultimo viene restituito al framework eseguendo delle operazini di raggruppamento di elementi con la stessa chiave. La fase di Reduce aggrega i risultati intermendi generando l'output. Il programmatore dovrà solamente definire le funzioni di map e reduce, mentre tutte le altre attività verranno gestite da mapred.

4.3 Hadoop

Hadoop è un framework che supporta applicazioni distribuite con elevato accesso ai dati ed unisce i sistemi di Map Reduce e HDFS. Le principali caratteristiche di questo tipo di distribuzione dei dati è di essere un modello economico di storage scalabile; infatti, il costo per TB è più basso rispetto ad ogni altra tecnologia. Hadoop immagazzina i dati in server multipli in modo da distribuirli successivamente su diversi computer migliorando le performance dei database. Inoltre, i dati non devono essere caricati in un formato strutturato: il fenomeno chiamato *schema-on-read* permette ad Hadoop di immagazzinare i dati di diverso formato velocemente. L'imposizione della struttura può essere posticipato fino a quando si vuole accedere ai dati.

L'ecosistema di Hadoop include divere utility e applicazioni sempre in espansione. *Pig* è uno scripting language che esegue operazioni di analisi di dati; il linguaggio utilizzato è chiamato Piglatin, che viene tradotto il mapred. *Hive*, invece, ha una interfaccia simile a SQL per dati memorizzati in HDFS, rappresentando perciò un datawarehousing basato su Hadoop. Questo applicativo è utilizzato per report giornalieri, misure di attività degli utenti, data e text mining, machine learning e per attività di business intelligence come pubblicità e individuazione degli spam. Tuttavia, il sistema di MapRed ha dei limiti: è difficile da comprendere, i task non sono riutilizzabili, è incline agli errori e per analisi complesse richiede molti lavori di MapReduce. In Hadoop 1.0 dunque, ogni jobtracker deve gestire molti compiti: gestire le risorse computazionali, scandire i task dello stesso lavoro, monitorare la fase di esecuzione, gestire i possibili "failure" e molto altro ancora. La soluzione a questo problema è stata splittare la fase di gestione in gestione dei cluster e gestione dei singoli lavori. Questo sistema è stato messo in pratica da YARN (Yet Another Resource Negotiator) in cui è presente un ResourceManage globale e un ApplicationMaster per ogni applicazione.

4.4 Data Warehouse

I dati vengono raccolti da fonti diverse e devono essere successivamente aggiunti al database. Per fare ciò esistono due soluzioni:

- Integrazione (vista in precedenza), aggregando i dati in diverse modalità. I problei principali di questo metodo sono il costo e la non efficienza per query complesse e l'impossibilità di utilizzo dei dati storici;
- Data Warehouse, nella quale l'informazione è integrata prima del suo utilizzo e memorizzata per consentire analisi ed interrogazioni complesse.

La seconda soluzione è quella adottata dalla maggior parte delle aziende. Anche se i dati non possono essere aggiornati, è possibile modificare e ristrutturare l'informazione memorizzata sui Data Warehouse; inoltre, si ha un maggior controllo di sicurezza e può contenere informazioni storicizzate.

> Un Data Warehouse è un singolo, completo e coerente store di dati ottenuti da varie risorse e rese disponibili agli utenti in modo comprensibile ed essere utilizzati in ambiti economici (Barry Devlin)

4.4.1 Architecture

I Data Warehouse possono avere diverse architetture: l'architettura a *2 livelli* permette ai dati esterni ed operazionali di essere uniti in un unico data warehouse, successivamente diviso in piccoli database (chianmati in inglese *data mart*) che contengono le informazioni

rilevanti per una particolare area. Alla fine si compiono analisi di dati ed altre procedure. I vantaggi principali di questa architettura sono la continua formazione di dati di buona qualità e l'essere organizzata in una logica basata sul modello multidimensionale prendendo in considerazioni diverse aree. Esistono delle tecniche specifiche per ottimizzare il software utilizzato per lo store dei dati. L'architettura a *3 livelli* ha una struttura simile alla precedente, ma dato che gestire in unico datawarehouse grandi moli di dati non è così efficiente, i dati vengono riconciliati, quindi puliti ed integrati alla sorgente. Il vantaggio principale è quello di creare un modello di dati comune e di riferimento per l'intera azienda. Un altro approccio è la costruzione di *data mart indipendenti* per ridurre la complessità di un unico data warehouse e quindi anche il tempo di costruzione. In particolare, si hanno delle dimensioni conformi di come rappresentare i dati nella stessa maniera; l'unico svantaggio è l'incoerenza nelle definizioni nei dati (una informazione è coerente in un data mart, ma forse non in un altro).

4.4.2 Multidimensionality:

Il data warehouse si basa sul concetto di *multidimensionalità* dei dati: una volta che i dati sono stati ripuliti, integrati e trasformati bisogna capire come trarne il massimo vantaggio. Esistono 3 approcci differenti: la *reportistica*, orientato agli utenti che hanno necessita di accedere periodicamente ad informazioni strutturate; *data mining*, utilizzando modelli predittivi per prevedere informazioni future; *OLAP* (***O***n***L***ine ***A***nalytical ***P***rocessing), che consente di analizzare ed esplorare interattivamente i dati con una interffaccia semplice da utilizzare e flessibile. Una sessione OLAP consiste in un percorso di navigazione tramite query o via grafica per svolgere analisi sotto diversi aspetti e a diversi livelli di dettaglio. Ogni passo di sessione di analisi è scandito dall'applicazione di un operatore OLAP che trasforma l'ultima interrogazione formulata in una nuova integrazione. I fatti di interesse sono rappresentati in *cubi* in cui ogni cella contiene misure che quantificano il fatto da diversi punti di vista, ogni asse una dimensione di interesse e ogni dimensione può essere la ridce di una gerarcia di attributi per integrare i dati memorizzati in cubi base. Le informazioni sono categorizzate e le categorie seguono delle gerarchie in base alle proprie esigenze. Inoltre, si possono compiere operazioni di aggregazione dei dati per successive analisi. Le funzioni base di uno strumento OLAP sono diverse:

- *Pivoting*: operazione di rotazione delle dimensioni di analisi per analizzare i dati in dimensioni diverse;
- *Slicing*: riduzione della dimensionalità del cubo fissando un valore per ogni dimensione;
- *Dicing*: riduzione della dimensionalità del cubo attraverso un criterio di selezione;
- *Roll-up*: aumenta l'aggregazione dei dati eliminando un livello di dettaglio da una gerarchia;
- *Drill-accross*: collegamento tra 2 o più cubi per comparare i dati;
- *Drill-down*: diminuzione di aggragazione dei dati aumentando il livello di dettaglio di una gerarchia.

4.4.3 DFM

Il DFM (***D***imensional ***F***act ***M***odel) è un modello concettuale grafico per data mart pensato per supportare il progetto concettuale e creare un ambiente su cui formulare intuitivamente interrogazioni dell'utente. La rappresentazione concettuale generata consiste in un insieme di schemi di fatto: gli elementi di base consistono in un *fatto*, un concetto di interesse per il processo nel corso del tempo; una *misura*, una proprietà numerica di un fatto che ne descrive un aspetto quantitativo; una *dimensione*, una proprietà con dominio finito di un fatto; un *attributo dimensionale*, le dimensioni e gli eventuali altri attributi che le descrivono; una *gerarchia*, un albero direzionato i cui nodi sono attributi dimensionali e gli archi modellano le associazioni tra le coppie di attributi. Invece, gli elementi avanzati possono o meno essere presenti nello schema di un DWS: un *attributo descrittivo* contiene informazioni aggiuntive su un attributo dimensione; un *attributo cross-dimensionale* è un attributo dimensionale o descrittivo il cui valore è determinato dalla combinazione di 2 o più attributi; la *convergenza* è un fenomeno in cui due attributi dimensionali possono essere connessi da due o più cammini direzionati distinti; la gerarchia condivisa è una abbreviazione usata per replicare più volte una porzione di gerarchie nello schema; un arco multiplo modella una associazione molti-molti tra due attributi dimensionali; una gerarchia incompleta è una gerarchia in cui per alcune istanze risultano assenti 1 o più livelli di aggregazione.

Da un modello di questo tipo, con uno schema concettuale, è possibile passare ad un modello logico relazionale. Si basa su principi quali la ridondanza dei dati e la denormalizzazione delle relazioni. Le principali operazioni da svolgere durante la progettazione sono le seguenti:

1. Si sceglie lo schema logico da utilizzare, ad esempio Star, Snowflake, ecc.: si rappresentano tutte le dimensioni e gerarchie come un tabella dimensionale. Per limitare la ridondanza dei dati, è possibile utilizzare il modello a fiocco di neve;
2. Si traducono gli schemi concettuali;
3. Si scelgono le gerarchie dinamiche;
4. Si scelgono le viste da realizzare;
5. Si applicano varie forme di ottimizzazione.

Le gerarchie sono adeguatamente modellate dalle dipendenze funzionali, che implicano una visione statica della realtà che può modificarsi nel tempo. La dinamica delle gerarchie influenza la rpgettazione logia del DWS. In funzione di come sarà utilizzata dalle interrogazioni, si prendono in considerazione 4 scenari:

- Oggi o Ieri: ciascun evento è riferito al valore delle gerarchie nell'istante di tempo in cui si è verificato;
- Ieri per Oggi: ciascun evento è riferito al valore inziale delle gerarchie;
- Oggi per Ieri: gli eventi sono inclusi alla situazione delle gerarchie alla data odierna;
- Oggi e Ieri: si considerano solo gli eventi riferiti a valori immutati della gerarchia

4.4.4 Data Lake

Un *Data Lake* è un repository utilizzato per analizzare grandi quantità di dati in diversi formati nel loro formato nativo; la sua architettura è basata per contenere tutti i tipi di dati, come i dai prodotti dalle macchine, dalle persone e da quelli tradizionali. Infatti, lo schema ed i requisiti dei dati non sono definiti in partenza, ma vengono delineati al momento dell'interrogazione, processo chiamato *schema on read*. In questo modo si riduce la quantità di dati inseriti nel data warehouse. Il data lake può essere utilizzato come un'area di atterraggio per la fase di store dei dati nel DWS. Per implementare un data lake si possono utilizzare diverse tecnologie, come HDFS che è la più utilizzata, NoSQL, ecc. Lo schema di un data lake è il segeunte:

- I dati sono presi ed immagazzinati nel loro formato nativo (*schema on read*);
- I dati vengono analizati per determinare il proprio valore in base al tipo di analisi che si vuole fare con processi scalabili paralleli;
- Si catturano i dati originali e li si memorizzano sotto forma di analisi usata precedentemente (*schema on write*).

Le principali fasi di un data lake sono le seguenti:

1. *Transient load zone*: I dati sono presi ed immagazzinati nel loro formato nativo per un controllo della loro qualità;
2. *Raw Data Zone*: una volta finito il controllo delle performance i dati vengono caricati in questa zona;
3. *Curated Data Zone*: i dati venogno puliti ed organizzati per successive analisi o per integrarli;
4. *Analytics Sandbox*: si compiono delle analisi esplorative e predittive.

La principale differenza tra un data lake ed un DWS è che per acquisire i dati la prima utilizza meno memoria, viceversa la seconda, mentre per il recupero dei dati è più veloce per la seconda rispetto alla prima. I due sistemi coesistono per una performance migliore. La principale architettura utiizzata tra questi sistemi è chiamata *Lambda*: una volta trovato valore ai dati vengono immagazzinati nel DWS. I data sciencist, in questo modo, possono provare a giocare con i dati senza che vengano sovrascritti e fino a quando non si estrae una importante informazione non vengono immagazzinati in DWS.

La memorizzazione è basato sul recupero ottimale, quindi veloce e chiaro, dei dati con un pattern autoesplicativo. L'organizzazione è frequentemente basato sull'area del soggetto, sul tempo di partizionamento e sull'accesso dei dati da parte dell'utente.

5 Velocity

Per *velocità* si intende quel processo in cui i dati devono essere acquisiti processati velocemente. Per farlo, si utilizzano software chiamati Online Data Analytics. I dati devono essere processati velocemente perchè ogni decisione tardiva è una opportunità mancata e quindi un minore guadagno da parte della azienda; inoltre, può portare a gravi conseguenze da parte dell'utente finale, come nel caso del healthcare. Bisonga tenere in considerazione la differenza tra dati raccolti in tempo reale e dati veloce: i primi riguardano un contesto applicativo per il tipo di analisi che si vogliono compiere. Il processo di cattura dei dati avviene nel seguente modo: da uno streaming di dati si creano flussi continui. L'obiettivo è catturare i dati interessati e immagazzinarli in repository determinando il valore di cattura. Dopodichè si possono usare diversi software per compiere analisi predittive, se si ha tempo per scaricarli e caricarli su HDFS.

Il software più utilizzato per la data stream è *Kafka*: è un meccanismo a coda nella quale un producer vuole pubblicare un evento di interesse e lo scrive in una coda, non preoccupandosi di chi leggerà il dato, pubblicando il messaggio (fase di *publish*). Quando un consumatore è interessato all'evento del precedente producer, kafka avvisa il consumatore che un dato è disponibile, tramite il processo di *subscribe*, e il messaggio viene visualizzato dal consumer.

Esistono diverse architetture per l'acquisizione di dati in tempo reale: l'architettura *Lambda*, nella quale da Kafka si formano due layer: uno di raccolta di dati, chiamato batch, nella quale si memorizzano in HDFS con calma; un layer in cui i dati vengono processati in tempo reale. Infine, i due layer vengono uniti per formare una query. La principale limitazione di lambda è che la applicazione logica è implementata due volte; per risolvere questo problema è stato introdotta l'architettura *Kappa*: all'inzio si raccolgono e analizzano i dati velocemnte e una volta terminata l'analisi vengono immagazzinati in un DWS.