Big Data and Reasoning

Daniele Avolio

Anno 2023/24

Contents

1 Teoria

In questi appunti ci saranno semplicemente gli appunti del corso di Big Data and Reasoning, tenuto dal professor Francesco Ricca presso l'Università della Calabria.

Non aspettatevi 100% di correttezza, sono appunti presi a lezione e quindi potrebbero contenere errori!

2

3 Lezioni di Laboratorio

Domande dell'esame possibili

In questa sezione ci sono domande che mi vengono fuori quando leggo i PDF e che quindi mi aspetto che possano essere inserite all'interno delle prove intermedie orale.

Buona fortuna a tutti!

4 Domande: Big Data Introduction

Domanda 4.1. *Cosa sono i Big Data?*

I Big Data sono un insieme di dati che viene differenziato dai comuni dati dal fatto che per essere gestiti non sono piu' abbastanza le vecchie tecnologie e metodi di gestione dei dati. I fattori che influenzano sono la velocita con cui vengono generati, il volume di dati che viene genegrato e la varieta' di dati che ci sono all'interno del dataset.

Domanda 4.2. *Quali sono le 5 V dei big data?*

Le 5 V dei big data fanno riferimento alle caratteristiche di essi, che sono:

- **Volume**: La quantita' di dati enorme che vanno gestiti. La scala che va fino ai Zettabyte.
- **Varieta'**: I vari tipi di danni che sono all'interno del database da analizzare e gestire. Possono essere immagini, audio, video, testo, ecc...
- **Velocita'**: La velocita' con cui i dati vengono generati e devono essere analizzati.
- **Veracita'**: L'incertezza e l'imprecisione dei dati che vengono generat e che quindi devono essere analizzati per capirne la *qualita'*.
- **Valore**: Il valore che i dati hanno per l'azienda che li possiede. Ad esempio vantaggi di business e analisi possibili da condurre.

Domanda 4.3. *Qual e' stata la prima rivoluzione nei database?*

Boh era quella iniziale dove praticamente si usava un modello gerarchico e veniva utilizzato Cobol. Diciamo che per gestirne uno sarebbe servito un esperto di database.

Domanda 4.4. *Qual e' stata la seconda rivoluzione nei database?*

L'introduzione del **modello relazionale** che porta i dati da essere tutti in unici tabelle a diverse con una struttura ben specifica.

Qui nasce anche il concetto di **transzione** che deve essere ACID:

- A: Atomica
- C: Consistente
- I: Isolata
- D: Duratura

Domanda 4.5. *Qual e' stata la terza rivoluzione nei database?*

Quando sono stati inventati i social network, grandissime moli di dati sono cominciate ad essere generate. Le strutture di un tempo non erano ancora abbastanza solide da poter gestire delle moli di dati tali. Per questo motivo **nuove tecniche** sono state inventate. Quindi nascono anche i database non relazionali. Ci sono state varie invenzioni come **lo sharding**. Sono stati creati altri tipi di database come quelli **(chiave, valore)**. Principalmente, pero', fu Google con il **distributed file system** a dare il via a questa rivoluzione.

Un concorrente a google fu Yahoo con HDFS (Hadoop Distributed File System).

5 Domande: GoodBye SQL

Domanda 5.1. *Differenza tra Scale Up e Scale Out*

Scale Up o scalare verticalmente significa *aggiungere risorse ad una macchina sola*. Ad esempio, significa potenziare una macchina tale da poter gestire piu carico di lavoro. Questo e' un approccio che e' stato utilizzato per molto tempo e che ha portato a creare macchine sempre piu potenti. Il problema e' che questo approccio ha un limite fisico, ovvero non si puo' andare avanti all'infinito. Inoltre, questo approccio e' molto costoso.

Scale Out o scalare orizzontalmente e' una cosa piu economicamente sostenibile, poiche' si basa sul concetto di *aggiugnere piu macchine che lavorano tra loro per gestire il carico di lavoro*. Questo e' notevolmente piu economico perche' i costi di una macchina sono molto piu bassi rispetto a quelli di una macchina potenziata.

Domanda 5.2. *Quali sono stati i primi tentativi di scalare orizzontalmente*

I primi tentativi furono quelli di costruire dei **Memcached server**, in cui gli utenti potessero effettuare letture senza andare a toccare i database principali. Un'altra tecnica era quella della **replicazione dei dati** su piu' databases.

Quando va bene questo? Quando le letture sono notevolmente maggiori rispetto alle scritture. Questo perche' se ci fossero tante scritture, ci sarebbe un bottleneck architetturale, dato che il database su cui vengono effettuate le scritture continua a rimanere 1 e 1 soltanto.

Domanda 5.3. *Cosa significa Sharding e come viene usato?*

Sharding significa **tagliare orizzontalmente un database** per splittarlo su diverse macchine. Questo comporta un aumento delle performance sopratutto in scrittura, poiche' aumenta il numero di macchine dove si puo' scrivere, ma aumenta anche la **complessita' di gestione del sistema.**

Bisogna anche effettuare il taglio con un criterio, poiche' poi quando i dati devono essere recuperati si vuole un tempo di accesso comunque accettabile senza dover effettuare una ricerca su tutti e n i database.

Problemoni:

- Complessita': Come gia' detto, gestire un sistema del genere diventa complesso e richiede una conoscenza elevata
- SQL: Non si puo usare SQL su diverse shard, quindi ci vuole un meccanismo dietro le quinte che gestisca il tutto
- ACID LOSS: Si perdono le transizioni ACID perche' su piu macchine non esiste, poiche' si ritornerebbe ad avere un bottleneck

Domanda 5.4. *Spiega il teorema di CAP*

Il teorema di CAP che sta per **Consistentcy, Availability e Partition Tolerance** dice che **non si possono avere tutti e 3 i requisiti** in un sistema distribuito. Provando ad ottenere tutti e 3 contemporaneamente, si andra' comunque a perdere 1 dei 3.

- Consistentcy: Ogni utente deve avere la stessa visualizzazione del contenuto in qualsiasi istante
- Availability: Ogni richiesta deve essere servita
- Partition Tolerance: Il sistema deve funzionare anche se alcune macchine non sono disponibili

Un esempio pratico: *Immagina di avere un servizio di database distribuito per un'applicazione di shopping online. In questo sistema, hai utenti che effettuano ordini e aggiornano il proprio carrello degli acquisti in tempo reale. Il database è distribuito su più server in diverse località geografiche per garantire la ridondanza e la tolleranza ai guasti.*

Coerenza (Consistency): Supponiamo che tu abbia implementato una forte coerenza dei dati, il che significa che ogni aggiornamento del carrello degli acquisti di un utente deve essere immediatamente riflesso su tutti i server. Questo assicura che tutti i server abbiano sempre una vista coerente dei dati. Tuttavia, quando si verifica una partizione di rete tra due gruppi di server, il sistema deve scegliere tra attendere la risoluzione della partizione (rendendo il servizio non disponibile per un certo periodo) o accettare il rischio di avere carrelli degli acquisti temporaneamente non coerenti tra le due parti del sistema.

Disponibilità (Availability): Se desideri massimizzare la disponibilità, il sistema dovrebbe continuare a permettere agli utenti di aggiornare i propri carrelli degli acquisti, anche se si verifica una partizione di rete tra i server. Tuttavia, ciò può comportare il rischio di avere dati non coerenti tra i due lati della partizione durante il periodo di separazione.

Tolleranza alla partizione (Partition Tolerance): Il sistema deve essere in grado di tollerare le partizioni di rete. Questo significa che il servizio dovrebbe funzionare in presenza di guasti di rete o partizioni geografiche, ma potrebbe comportare una temporanea mancanza di coerenza o disponibilità in situazioni di partizione.

Quindi, in questo esempio, puoi vedere che il teorema CAP si applica. Quando si verifica una partizione di rete, devi fare una scelta tra coerenza e disponibilità. Non è possibile garantire contemporaneamente entrambe le proprietà. Il sistema deve tollerare la partizione ma potrebbe sacrificare la coerenza o la disponibilità a seconda delle decisioni di progettazione prese.

Domanda 5.5. *Spiega: No Go Zone, Eventual Consistentcy, Strict Consistentcy*

- No Go Zone:

$$\text{Consistentcy} \cap \text{Availability} \cap \text{Partition Tolerance}$$

- Strict Consistentcy:

$$\text{Consistentcy} \cap \text{Partition Tolerance}$$

- Eventual Consistentcy:

$$\text{Availability} \cap \text{Partition Tolerance}$$

Domanda 5.6. *Come funziona Amazon Dynamo?*

- Database non relazionale alternativo.
- Accesso basato su chiave primaria.
- Dati recuperati da una chiave sono oggetti binari non strutturati.
- La maggior parte degli oggetti è piccola, sotto 1 MB.
- Dynamo permette di sacrificare la coerenza per garantire disponibilità e tolleranza alle partizioni.

Caratteristiche architetturali chiave

- Consistent hashing: usa l'hash della chiave primaria per distribuire i dati tra i nodi del cluster in modo efficiente.
- Coerenza regolabile: l'applicazione può bilanciare la coerenza, le prestazioni di lettura e scrittura.
- Versionamento dei dati: permette la gestione di più versioni di oggetti nel sistema, richiedendo la risoluzione delle versioni duplici da parte dell'applicazione o dell'utente.

Principali caratteristiche di Dynamo

- Hashing consistente: assegna le chiavi ai nodi in modo efficiente, consentendo l'aggiunta o la rimozione di nodi con minimo riassestamento.
- Coerenza regolabile: permette di scegliere tra coerenza, prestazioni di lettura e scrittura.
- Versionamento dei dati: gestisce più versioni di oggetti, richiedendo la risoluzione delle versioni duplici da parte dell'applicazione o dell'utente.

Domanda 5.7. *Qual e' la differenza tra SQL e NOSQL*

Nei database NOSQL si perde la strictness del modello relazionale e si parla piu' di **modello a documenti**. Questo significa che i dati sono salvati con una struttura diversa tra loro, non obbligatoriamente. Il formato con cui si salvano e' spesso **xml** o **json**.

Ogni documento possiamo dire che rappresenti una riga di un database relazionale. All'interno del documento ci potrebbero essere anche documenti innestati e liste.

6 Domande: Google Stack e Hadoop

Domanda 6.1. *Come funziona il file system di Google*

Il GFS o Google File System si basa sul concetto di avere costantemente i loro servizi attivi, avendo a disposizione un **numero enorme** di macchine e server che garantiscono l'**availability** costante.

La consistenza non e' un qualcosa che viene garantito, ma viene garantito una **fault tolerance** costante. Anche perche' avendo 1.000.000 di macchine, il fatto che 1 non funzioni diventa la normalita'. Ci si aspetta di avere anche dei sistemi che ripristino il sistmea quando questo accade.

- Faul tolerance: tante macchine che permettono di avere un'espansione orizzontalme
- Banda sostenuta: garantire sempre il servizio a discapito della latenza
- Atomicita' con minimi problemi di sincronizzazione
- **Availability**

Si basa su assunzioni di dover gestire tanti file dell'ordine di GB, di avere *poche letture random e tante letture a batch.*

Domanda 6.2. *Qual e' la struttura del GFS*

- Tanti client: accedono al servizio
- Un Master: Gestisce i metadati e comunica con i chunkserver
- Tanti chunckserver: Salva i dati e sul disco locale. Spesso ogni chunck viene duplicato per garantire la fault tolerance e availability.

Diciamo che la procedura di comunicazione dell'applicazione e' che:

1. Il client chiede al master quale chunckserver contattare
2. Il client contatta il chunck server
3. Vengono effettuate le operazioni sui chunk
4. Il master controlla lo stato dei chunckserver e aggiorna i metadati

Domanda 6.3. *Come funziona il paradigma MapReduce?*

Nota: IN un programma parallelo quali sono i maggiori bottleneck? **I LOCK**

Il paradigma si basa su due fasi:

1. Map: I dati vengono partizionati in chunks che saranno poi gestiti in parallelo

2. Reduce: Si combina l'output dei vari mappers per avere il risultato finale

Da notare che viene effetuato **con bruteforce** e il codice fa parecchio schifo mi sa.

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

Domanda 6.4. *Come funziona BigTable?*

BigTable e' un database distribuito che si basa su GFS e MapReduce. E' un database **non relazionale** che si basa su una struttura di **colonne** e **righe**.

Sparsa distribuita persistente multi dimensionale ordinata MAPPA

(riga: stringa, colonna:stringa, tempo: int64) $\rightarrow$ stringa

Le pagine web sono salvate con URL al contrario perche per distribuire i dati in modo uniforme sui server, BigTable deve utilizzare una chiave di partizionamento. In questo caso, la chiave di partizionamento è l'URL della pagina web.

Per evitare che tutti i dati vengano memorizzati su un unico server, BigTable utilizza una funzione di hash per trasformare l'URL in una chiave di partizionamento. Tuttavia, se l'URL fosse utilizzato come chiave di partizionamento così com'è, ci sarebbe il rischio che tutti i dati relativi a un singolo sito web finiscano sullo stesso server.

Per evitare questo problema, Google salva gli URL al contrario. In questo modo, l'hash della chiave di partizionamento viene distribuito in modo più uniforme sui server, migliorando le prestazioni del database.

- Righe: sono stringhe arbitrarie che identificano la riga
- Colonne: sono stringhe arbitrarie che identificano la colonna. Si chiamano famiglie. Spesso i dati della stessa famiglia sono dello stesso tipo.
- Timestamp: Viene usato per poter avere una versione dei dati e poter fare rollback

Domanda 6.5. *Implementazione di BigTable*

Ha 3 elementi principali.

Il client che usa le API per comunicare. Un **Master** che gestisce i tablet servers e garantisce il load balancing. **I tablet servers** che gestiscono un insieme di tablets.

I tablet servers gestiscono il partizionamento dei dati e le richieste in lettura e scrittura.

Notare che i **dati del client non passano dal master ma dai tablet servers**

Domanda 6.6. *Come funziona Hadoop*

Hadoop e' un framework open source che permette di gestire grandi quantita' di dati in modo distribuito. Si basa su HDFS e MapReduce.

Lo **schema on read** permette di non dover progettare uno schema per i dati. Lo schema dipende dai dati che si andranno a gestire. Anche questo si basa sull'aggiungere hardware comodo ed economico per scalare.

- Scale out
- Chiave VALORE
- Functional Programming
- Offline batch (non realtime)

Domanda 6.7. *La struttura di Hadoop*

- NameNode: gestisce i metadati e i file system (**UNO E UNICO**)
- DataNode: ogni macchina slave avra' un datanode che gestisce i dati. (**SONO TANTI**)
- DataNode secondario: Usato come backuo diciamo (**tipicamente uno**)
- JobTracker: gestisce i job e le richieste dei client (**Uno solo, master**)
- TaskTracker: gestisce i task sui vari slave node(**tanti, slave**)