

# Architectures for big data

Francesco Tomaselli

31 gennaio 2021

## Indice

<b>1</b>	<b>Architetture</b>	<b>2</b>
1.1	Introduzione . . . . .	2
1.2	Motivazioni . . . . .	2
1.3	CDC e Datalake . . . . .	3
1.4	Jobs e scheduler . . . . .	4
<b>2</b>	<b>Apache Hadoop</b>	<b>5</b>
2.1	Elementi . . . . .	5
2.2	Processo . . . . .	5
2.3	HDFS . . . . .	6
<b>3</b>	<b>Spark</b>	<b>6</b>
3.1	Elementi . . . . .	6
3.2	Funzionalità e caratteristiche . . . . .	7
3.3	Processo . . . . .	8
<b>4</b>	<b>Deltalake</b>	<b>8</b>

# 1 Architetture

## 1.1 Introduzione

**Stile architetturale** Uno stile architetturale definisce la lista di design elements utilizzabili, oltre alle relazioni che sussistono tra essi. Nel contesto di sviluppo software uno stile incapsula le scelte importanti prese dagli architetti e ne coordina le interazioni. Esistono due problemi, erosion e drift, il primo consiste nell'usare l'architettura impropriamente, il secondo si manifesta quando un'architettura viene usata per scopi diversi da quelli originali.

**Elementi** Per quanto riguarda gli elementi di un architettura, essi si suddividono in:

1. *Data elements*: contengono informazione
2. *Connecting elements*: connettono diverse parti dell'architettura
3. *Processing elements*: trasformano i dati

**Materiali** Un materiale ha qualche peculiare caratteristica, è necessario scegliere materiali adatti agli scopi. Nel software, si parla di framework, linguaggi di programmazione, etc ...

**Point of views** È cruciale avere diversi punti di vista nella presentazione di un architettura, poichè esistono più attori con punti di vista differenti. Ad ognuno interessano o meno alcuni dettagli o aspetti dell'architettura totale.

Un esempio potrebbe essere una vista ad alto livello data al cliente, e una molto specifica data allo sviluppatore. Si può pensare poi ad una vista riguardo alle spese, che molto probabilmente non importerà ad esempio al team di sviluppo.

**Astrazione** Per astrazione si intende l'individuare un pattern, nominarlo, definirlo, analizzarlo e trovare un modo di invocarlo tramite il suo nome, così da evitare errori.

L'idea di base è quella di omettere i giusti dettagli nel contesto opportuno, semplificando l'interpretazione del risultato.

## 1.2 Motivazioni

Esistono vari motivi per cui si architetta un architettura software.

1. Framework per soddisfare richieste
2. Base tecnica per il design, ovvero, è la base per la definizione di un particolare design, che ha tra i suoi elementi un'architettura a supporto

3. Base per la stima dei costi di un sistema
4. Porta al riuso delle componenti
5. Porta alla centralizzazione dei dati
6. Aumenta produttività e sicurezza
7. Mitiga il rischio di lock-in

**Vendor lock-in** Condizione in cui un cliente dipende da un certo vendor. Tale dipendenza si manifesta se il costo di cambiare vendor è maggiore rispetto al tenerlo. Per esempio, deciso un cloud o un servizio, un'azienda potrebbe avere adattato il suo sviluppo a quel particolare ambiente di sviluppo. Un cambio di cloud porterebbe a costi di adattamento troppo importanti.

Esiste anche il knowledge lock-in, ovvero quando il costo di spiegare o impartire una conoscenza in qualcuno costa più di mantenere la persona attuale. Per proteggersi da vendor lock-in si può ricorrere ad *Adapter Pattern*.

**Design pattern** Un design pattern è la formalizzazione di una best practice per risolvere un problema comune. Ad esempio: singleton, SOA, REST, P2P, MapReduce ...

### 1.3 CDC e Datalake

**Datalake** L'idea di un Datalake è quella di avere un luogo in cui salvare dati strutturati, ottenuti da una certa sorgente in forma non strutturata. Su un datalake si salva solamente, non si elimina nulla, alla modifica di un dato si inserisce la sua versione modificata.

Tale approccio cambia il pattern *ETL*, poichè, si effettua una Extract, e una Load, per salvare su datalake, in un secondo momento si trasformano i dati in base al contesto.

**Change data capture** L'idea del CDC è quella di salvare solo i dati che sono nuovi o aggiornati dall'ultima raccolta dati. Ad esempio, nella raccolta giornaliera di una tabella SQL, è possibile, invece che salvare più volte la tabella intera, salvare solo le tuple nuove o modificate, riducendo di molto lo storage utilizzato e il costo di computazione seguente.

Esistono vari modi di implementare CDC:

- *Invasive database side*: timestamp su righe, numero di versione, indicatore di stato, trigger su tabelle
- *Invasive application side*: si basa su eventi, un'applicazione deve mandare informazioni sullo stato al cdc
- *CPU database*: si confrontano i log del database, oppure utilizzando log shipping, tipicamente coinvolto nel backup automatico

**Diff and Where** Si differenziano due tipi di tabelle, log e registry table. Nella prima esiste un chronoattribute, utilizzabile per ottenere i dati aggiornati tramite query, nella seconda si applica un approccio basato su hash per capire se le tuple sono state modificate dall'ultima volta.

**Move and rename** L'approccio consiste nel garantire transazionalità di un CDC. È necessario quindi rendere possibile un eventuale rollback, e non lasciare mai il datalake in uno stato inconsistente. L'idea è scrivere i file in formato tmp e rinominarli al termine del job cdc. All'iterazione successiva si effettua rollback dei file tmp.

**Adapter pattern** Per evitare vendor lock-in, si utilizzano classi wrapper per evitare di dipendere strettamente da metodi definiti da un vendor. Un esempio può essere un source o destination adapter, che espongono metodi read e write, implementati a seconda del contesto.

## 1.4 Jobs e scheduler

**Job** Un job è un elemento atomico che identifica una sequenza di passi o task da compiere. Può essere lanciato interattivamente o schedato. Si identifica con un singolo processo.

Un job è caratterizzato da un id, un nome e uno stato, può essere poi

1. Finito: possono completare, essere terminati o fallire
2. Online: possono essere solo interrotti quando terminano

**Scheduler** Governa l'esecuzione dei job, può essere basato sul tempo o su eventi. Può decidere a quale processore o server (sistema distribuito) mandare un certo job, scegliendo tra quelli liberi.

**Job queue** Coda dei job che sono da eseguire, esiste un concetto di priorità, riconducibile alle classiche priorità dei processi nei sistemi operativi.

**Job impersonation** Un thread che esegue un job può impersonare un determinato client. Ad esempio, dopo un login, un thread può agire impersonando l'utente loggato, in modo che il server possa rispondere nel modo opportuno.

**Concurrency** Due job possono essere sequenziali o concorrenti.

## 2 Apache Hadoop

Apache Hadoop è un framework open source. L'idea è quella di aumentare l'affidabilità di un sistema, basandosi sul fatto che l'hardware può fallire. Ha tre componenti: storage, resource management e computazione parallela.

### 2.1 Elementi

**Job Tracker** Servizio che decide dove una task deve essere eseguita.

**Task Tracker** Nodo che accetta la task data da un job tracker. Espone slot che possono eseguire task parallele, infatti, all'arrivo di una nuova task, uno slot libero viene utilizzato per eseguire una JVM. Manda segnali di heartbeat.

**Name node** Contiene informazioni su dove i dati sono scritti. Mantiene quindi una lista di risorse e i relativi nodi che le contengono. Rappresenta un single point of failure per Hadoop. Esiste un backup name node.

**Data node** Nodo che contiene i dati effettivi.

**Master e worker node** Un nodo master è composto da un job tracker, un task tracker, un data node e un name node. Mentre un nodo worker è composto da data node e task tracker.

### 2.2 Processo

*Moving computation is cheaper than moving data*

Un'applicazione manda un job asincrono al job tracker, aspettando in polling. Il job tracker cerca i dati nel name node e sceglie i task tracker più vicini con slot disponibili.

Successivamente il job tracker manda i job ai task tracker, e ne monitora lo stato attraverso i segnali di heartbeat. Nel caso di fallimento il job tracker può scegliere di aggiungere il task tracker a una blacklist.

**Confronto costi** Bisogna tenere presente che in un contesto come quello di Hadoop risulta più economico muovere la computazione piuttosto che i dati. Bisogna quindi scegliere i task tracker in modo opportuno, tenendoli vicini ai data node interessati, piuttosto che muovere i dati in sé ed eseguire le operazioni.

## 2.3 HDFS

HDFS è uno storage distribuito pensato per essere utilizzato su macchine poco performanti.

Un HDFS ha un architettura master/slave, dove esiste un name node per N data node. Il primo esegue operazioni sul file system, mentre i secondi si occupano di servire le richieste del primo e di gestire cancellazione, creazione e replica dei file.

**Data node** I file sono suddivisi in blocchi, le scritture non sono concorrenti sullo stesso blocco. Mandano segnali di heartbeat al name node.

**Replicazione dei dati** Si possono seguire varie strade per la replicazione dei dati in HDFS, ma bisogna essere rake aware, ovvero, bisogna replicare elementi dello stesso data node nello stesso posto, in modo da rendere più semplice la gestione di richieste multiple in caso di fallimento di un data node.

Un approccio è quello di replicare l'intero rack, diventa molto costoso perchè si sta scrivendo su rack differenti. Un'altro è quello di replicare 3 volte, una volta sul nodo che esegue la task, e altre due su due nodi di un rack differente. Le performance aumentano, essendoci 2 rack invece che possibilmente di più

**Quorum Journal Manager** I data node sono affidabili, viste le tecniche di replicazione, per estendere ai name node, essi potrebbero essere multipli, uno attivo e gli altri in standby. I data node manderebbero il battito a tutti questi nodi. Al fallimento del name node attivo, esso sarebbe rimpiazzato da un altro nodo.

## 3 Spark

### 3.1 Elementi

**Resilient Distributed Dataset** È un dataset distribuito in memoria su cui si possono effettuare operazioni in parallelo. Si può ottenere da un HDFS, oppure parallelizzando qualsiasi oggetto. L'idea è quella di distribuire gli elementi tramite una funzione di hash tra i nodi che offrono computazione, tale distribuzione può essere forzata con un *repartition*. L'operazione è necessaria poichè ci si potrebbe trovare in casi dove un nodo ha molti più elementi degli altri, oppure per raggruppare meglio le chiavi in seguito a una join.

**Driver** Processo responsabile dell'esecuzione di Spark, divide l'applicazione in task piccole computabili dai nodi worker, che eseguiranno. Racchiude lo SparkContext, che permette di utilizzare Spark, e i metadati per l'RDD.

**Master** È un nodo che contiene il driver, si occupa di orchestrare il lavoro tra i nodi worker, e ne monitora lo stato. Può avere un nodo di backup pronto in caso di fallimento.

**Spark Context** È il core di Spark, permette al driver di utilizzare il cluster. È singleton e manda segnali di heartbeat agli executor per monitorarne lo stato.

**Worker** Nodo che si occupa della computazione, contiene molti executors.

**Executors** Sono quelli che effettuano le computazioni vere e proprie. Hanno un id, ed è garantito il backup se falliscono.

## 3.2 Funzionalità e caratteristiche

**Pigrizia** Spark è pigro, non computa il risultato subito ma solo quando serve. È possibile quindi che un errore si verifichi solo all'esecuzione di una particolare riga dell'RDD.

**Map e reduce** Spark offre alcune funzioni di map e reduce. La prima permette di applicare una funzione ad ogni entry dell'RDD, mentre la seconda permette di eseguire riduzioni. Le seconde devono essere commutative, poiché non è garantito l'ordine con cui si eseguono le operazioni.

**DAG Scheduler** Ogni qualvolta che si effettua un'operazione su un RDD non si sta facendo altro che aggiugnere una task al DAG Scheduler. Il suo compito è quello di trasformare un execution plan logico in uno fisico, in modo da effettuare la computazione vera e propria. Ogni passo del DAG Scheduler è chiamato Stage.

**Task Scheduler** Uno stage su un sottoinsieme di righe è chiamata Task. Una Task è lanciata dal Task Scheduler ed eseguita da uno Worker attraverso il Cluster manager. Il numero di Task è proporzionale al numero di partizioni considerate nell'RDD.

**Shared Variables** All'esecuzione di una map o reduce, Spark distribuisce una copia delle variabili delle funzioni tra tutte le righe. Questo comportamento può portare a problemi, nel caso di variabili molto pesanti. Si introducono le shared variables, che possono essere:

- *Broadcast variables*: read-only, per esempio, dizionario condiviso in memoria su cui fare lookup
- *Accumulators*: write-only

### 3.3 Processo

Definiti elementi e caratteristiche di Spark l'esecuzione puo essere riassunta in questi passi:

1. Submit di un'applicazione utilizzando spark-submit utility
2. Allocazione risorse necessarie dal Resource Manager
3. Application master si registra al Resource Manager
4. Spark driver manda il codice all'Application Master, convertendo il codice in un DAG
5. Il Driver negozia con il Cluster Manager sulle risorse, e si creano gli Stage del DAG Scheduler
6. Gli Executors sono istanziati dagli Worker
7. Il driver tiene traccia dello stato degli Executors e manda le task al Cluster manager in base alla distribuzione dei dati
8. L'application Master crea una Container configuration per il Node Manager
9. È creato il primo RDD
10. Durante l'esecuzione il driver parla con l'Applicatin Master per monitorare lo stato, al termine si rilasciano le risorse

## 4 Deltalake