POLITECNICO DI MILANO

Facoltà di Ingegneria
Scuola di Ingegneria Industriale e dell'informazione
Dipartimento di Elettronica, Informazione e Bioingegneria

Corso di Laurea Magistrale in
Computer Science and Engineering

Progettazione di un'architettura distribuita per l'aggregazione di sorgenti dati streaming

Relatore:

PROF.SSA CHIARA FRANCALANCI

Correlatori:

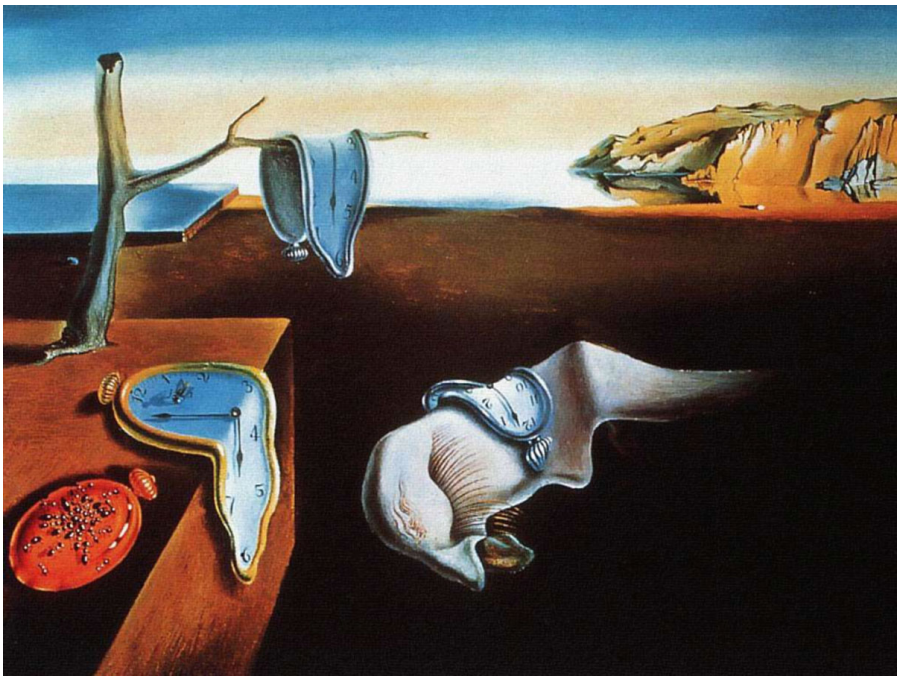
ING. PAOLO RAVANELLI
ING. ANGELA GERONAZZO

Tesi di Laurea Magistrale di:

DANIELE BARTOLI
Matricola n. 836326

VINCENZO RICCARDI
Matricola n. 879082

Anno Accademico 2016-2017



La persistenza della memoria - Salvador Dalí

Ai miei genitori...

INDICE

LISTA DELLE IMMAGINI

LISTA DELLE TABELLE

SOMMARIO

Lo scopo principale di questo elaborato è illustrare i passaggi chiave che hanno portano alla costruzione di un'architettura distribuita per l'integrazione di dati. Per la realizzazione del progetto, sono stati necessari due componenti, ovvero il database e il processing system. Dopo un'attenta analisi tra le tecnologie più diffuse in ambito Big Data, e delle conferme attraverso alcune prove sperimentali, abbiamo selezionato rispettivamente MongoDB e Apache Spark. Nello specifico l' integrazione riguarda un caso di DEM, cioè di Direct Email Marketing, in collaborazione con un'azienda del settore. Le sorgenti dati sono database contenenti informazioni riguardanti gli utenti. I dati sono ricevuti attraverso alcuni file CSV aventi la stessa struttura, ma con la presenza di alcuni duplicati e/o errori ortografici e/o attributi non validi. Allo stato attuale l'azienda non riesce a sfruttare pienamente i dati ricevuti. L'obiettivo del progetto è stato processare i dati in input e inserirli in un database integrato facilmente interrogabile in modo da fornire ai possibili clienti un comodo metodo per ottenere dati segmentati secondo specifici criteri. Inoltre, in parallelo al processing, è stato aggiunto un data lake contenente i dati grezzi per permetterne sviluppi futuri, cioè per favorire l' adozione di una struttura completamente diversa da quella attuale, senza dover riorganizzare completamente la base di dati esistente. L'architettura è stato pensato per essere flessibile ed efficiente, questi aspetti sono sono stati presi in considerati durante tutte le fasi della progettazione.

ABSTRACT

The main purpose of this work is to illustrate the steps that have led to the construction of a distributed architecture for data integration. In order to fulfil the project requirements, two components were needed, namely the database and the processing system. After a careful analysis of the most widespread technologies in the Big Data field, and confirmations through some experimental tests, we have selected MongoDB and Apache Spark, respectively. Specifically, the integration concerns a case of DEM, that is, Direct Email Marketing, in collaboration with a company in this sector. Data sources are databases containing information about users. The data are received through some CSV files having the same structure, but with the presence of some duplicates and / or incorrect spelling and / or not valid attributes. Now the company can not fully exploit the data received. The project objective was to process the input data and insert it into an integrated database that can be easily queried in order to provide to potential customers with a convenient method to obtain segmented data according to specific criteria. Moreover, in parallel to the processing, a data lake was added to accommodate raw data to allow future improvements, such as the adoption of a completely different structure from the current one, without completely reorganizing the existing database. The architecture has been conceived to be flexible and efficient, these aspects have been taken into account during all the design phases.

1 INTRODUZIONE

L'obiettivo di questo progetto è quello di costruire un'architettura integrata, facilmente accessibile che permetta di avere una visione unificata e corretta di eventuali errori sui dati inseriti. Le Direct Email Marketing sono aziende che si stanno sempre più evolvendo negli ultimi anni, infatti con l'aumento della mole di dati raccolti, è cresciuta la necessità di personalizzare la campagna pubblicitaria in modo da raggiungere solamente i clienti che sono maggiormente interessati a un determinato prodotto piuttosto che a un altro. Il caso principale di utilizzo di questa architettura è una campagna di Direct Email Marketing che utilizza i dati provenienti da varie sorgenti. I dati in ingresso allora, devono necessariamente garantire una certa qualità, affinché tutte le informazioni siano pienamente sfruttabili al fine di massimizzare l'efficacia della campagna.
Un ruolo centrale, per la riuscita della campagna pubblicitaria, viene quindi assegnato al database.
L'architettura realizzata è costituita da un database integrato con diverse caratteristiche: scalabilità, disponibilità, coerenza e tolleranza ai guasti. Le tecnologie che vengono utilizzate e descritte nei vari capitoli sono alcune tra le più diffuse nell'ambito Big Data.
Big Data è un termine ampiamente usato, e indica la capacità di estrapolare, analizzare e mettere in relazione un'enorme mole di dati eterogenei, strutturati e non, per scoprire i legami esistenti tra essi e cercare di prevedere quelli futuri. Per poter parlare di Big Data quindi, il volume dei dati deve essere correlato alla capacità del sistema di acquisire le informazioni così come arrivano dalle differenti sorgenti dati che sono adoperate.
Quindi, un sistema diventa "big" quando aumenta il volume dei dati e allo stesso tempo aumenta la velocità/flusso di informazioni che il sistema deve poter acquisire e gestire per secondo. Questo è quindi lo scoglio principale che il nostro progetto deve superare, l'efficienza infatti sta proprio nel ridurre al minimo il tempo di processing dei dati in ingresso. Il progetto si compone di due elementi principali, il DBMS e il Process System. Dopo un'attenta analisi delle varie tecnologie presenti sul mercato, considerando le caratteristiche più significative del nostro specifico caso al seguito anche di vari test eseguiti variando le tecnologie usate e i componenti dell'architettura, sono state prese delle decisioni.

Nella scelta del DBMS abbiamo tenuto conto dei due principali modelli esistenti, il modello relazionale e quello non relazionale, evidenziandone vantaggi e svantaggi nell'implementazione di uno rispetto all'altro. Scelto il modello si è focalizzata l'attenzione su quella precisa categoria di DBMS, mettendo in risalto i principali elementi distintivi di ogni singolo DBMS.

Per quanto riguarda il Process System, invece l'analisi è stata condotta sui due frameworks che meglio rispondono alla descrizione di Process System ideale di questo specifico progetto. L'analisi è stata quindi accompagnata dall'implementazione e dai successivi test sull'efficienza, che hanno infine posto in evidenza i difetti e i pregi delle varie implementazioni.

2 STATO DELL'ARTE

All' interno di questo capitolo si analizzano i due componenti principali del progetto, ovvero il DBMS e il processing system. Si parte quindi da un analisi ad alto livello sui database, ad esempio le varie tipologie e differenze tra DBMS, fino ad arrivare una descrizione e confronto tra principali tecnologie. In secondo luogo, si sposta l'attenzione sui processing systems in particolare su tre dei principali sistemi, ovvero Apache Storm, Apache Spark, Hadoop MapReduce.

2.1 I DATABASE

I dati digitali al giorno d'oggi sono raccolti in quantità enorme senza precedenti e in molti formati in una varietà di domini, ciò è stato possibile negli ultimi anni per l'incredibile crescita per la capacità degli strumenti di archiviazione dei dati e per la potenza di calcolo dei dispositivi elettronici, nonché dall'avvento del computing mobile e pervasivo, del cloud computing e del cloud storage.
Nel settore dell'IT trasformare i dati disponibili in informazione e far si che il business delle aziende abbia un vantaggio da tali informazioni, è un problema di lunga data. Nell'era dei "Big Data" questo problema è diventato ancora più complesso e difficoltoso, ma allo stesso tempo affrontare la sfida può essere meritevole, dal momento che la massiccia quantità di dati adesso disponibile può consentire risultati analitici mai raggiunti prima.
I database relazionali sono nati negli anni settanta e successivamente si sono evoluti fino ad arrivare alla massima diffusione negli anni duemila. La tabella è l' elemento base dei database relazionali, ne esiste una per ogni informazione da trattare, ognuna costituita da attributi, uno per ogni aspetto dei dati.
Una tabella di solito ha uno o più attributi che svolgono il ruolo di chiave primaria la quale identifica univocamente una certa tupla.
Tra le tabelle di un database relazionale possono esistere alcune relazioni, ad esempio: una tupla di una tabella A può far riferimento ad un altra tupla di un'altra tabella B e ciò può essere espresso inserendo la chiave primaria tra gli attributi della tabella B.
A partire dagli anni duemila e quindi in concomitanza con la nascita del web 2.0, si verifica una notevole diminuzione del costo dei sistemi di memorizzazione, un'enorme diffusione dell'e-commerce e dei social media, e una crescita esponenziale dei dati prodotti.
Tutto ciò porta, con il crescere dei dati, alla necessità di scalare, e quindi alla nascita dei database NoSQL.
Ovviamente, le caratteristiche e le proprietà che possiedono quest'ultimi sono differenti rispetto ai database relazionali, in particolare rispetto alle proprietà ACID su cui si fondano i relazionali.

Allora ancora oggi, bisogna trovare in ogni soluzione un' equilibrio tra le proprietà ACID (Atomicity, Consistency, Isolation e Durability) rispettate e il teorema CAP (Consistency, Availability e Partition Tolerance) che caratterizza i NoSQL. Tutto questo quindi porta, a seconda del progetto che si sta sviluppando, alla perdita di alcune proprietà ACID e contemporaneamente all'acquisizione di quelle del teorema CAP o viceversa.

2.1.1 *Differenze tra DBMS SQL e NoSQL*

I sistemi di basi di dati NoSQL non sono dotati di un modello come quello nel caso relazionale. Ci sono molte implementazioni, ognuna diversa dall'altra, nessuna è migliore dell'altra in modo assoluto ma può esserlo in una particolare situazione.

In sintesi, le principali differenze tra le due tipologie sono:

STRUTTURA E TIPI DI DATO I database relazionali che utilizzano SQL per memorizzare i dati necessitano di una struttura con degli attributi definiti, a differenza dei database NoSQL che possiedono una struttura più libera.

QUERY In modo indipendente dalle loro licenze, tutti i database relazionali implementano in una certa misura il linguaggio standard SQL, possono cioè, essere interrogati utilizzando SQL. Ogni database NoSQL implementa invece un modo proprietario e differente per operare con i dati che gestisce.

SCALABILITA' Entrambe le soluzioni possono scalare verticalmente. Le soluzioni NoSQL di solito offrono strumenti che permettono di scalare orizzontalmente molto più facilmente, essendo applicazioni più moderne e più semplici.

SUPPORTO I DBMS hanno una storia molto più lunga ed è molto più facile trovare supporto gratuito o a pagamento.

DATI COMPLESSI I database relazionali per natura sono la soluzione di riferimento per l'esecuzione di query complesse e per le problematiche sulla conservazione dei dati.

Nei DBMS NoSQL a documenti, è rilevante l'assenza delle relazioni. Esistono due meccanismi con cui vengono collegate le informazioni:

EMBEDDING Significa annidare un oggetto JSON all'interno di un altro. Questa tecnica sostituisce molto spesso le relazioni 1-a-1 e 1-a-molti. È tuttavia sconsigliabile utilizzarla quando i documenti (quello annidato e quello che lo contiene) crescono di dimensione in maniera sproporzionata tra loro, oppure se la frequenza di accesso ad uno dei due è molto minore di quella dell'altro;

REFERENCING Somiglia molto alle relazioni dei RDBMS, e consiste nel fare in modo che un documento contenga, tra i suoi dati, l'id

di un altro documento. Molto utile per realizzare strutture complesse, relazioni molti-a-molti oppure casistiche non rientranti tra quelle consigliate per l'embedding al punto precedente.

2.1.2 *Proprietà ACID*

In ambito informatico un insieme di istruzioni di lettura e scrittura sulla base di dati viene definito transazione. Possiamo definire alcune proprietà che un DBMS dovrebbe sempre garantire per ogni transazione che vengono identificate comunemente con la sigla ACID, che deriva dall'acronimo inglese Atomicity, Consistency, Isolation, e Durability. [1]

Atomicità

L'atomicità di una transazione è la sua proprietà di essere eseguita in modo atomico, ovvero al termine della transazione gli effetti di quest'ultima devono essere totalmente resi visibili oppure nessun effetto deve essere mostrato. Questa proprietà viene spesso garantita dal DBMS attraverso le operazioni di UNDO per annullare una transazione e di REDO per ripetere una transazione. Se questa proprietà viene rispettata il database non rimane mai in uno stato intermedio inconsistente, infatti un qualsiasi errore fatto prima di un commit dovrebbe causare un UNDO delle operazioni fatte dall'inizio della transazione.

Consistenza o Coerenza

Questa proprietà garantisce che al termine dell'esecuzione di una transazione, i vincoli di integrità sono soddisfatti. Infatti se questa proprietà viene rispettata il database si trova in uno stato coerente sia prima della transazione che dopo la transazione. Con un piccolo esempio possiamo capire meglio cosa intendiamo per consistenza. Consideriamo una transazione bancaria tra due conti correnti, il principale vincolo di integrità che la transazione deve rispettare è che la somma dei due conti correnti prima e dopo la transazione deve essere uguale. Se la transazione rispetta questo vincolo allora possiamo dire che è coerente/consistente.

Isolamento

La proprietà di isolamento garantisce che ogni transazione deve essere indipendente dalle altre, ovvero l'eventuale fallimento di una o più transazioni non deve minimamente interferire con altre transazioni in esecuzione. Quindi affinchè l'isolamento sia possibile, ogni transazione deve sempre avere accesso a una base di dati consistente. I livelli di isolamento principali sono 4:

- Read Uncommitted, che consente di eseguire transazioni in sola lettura, senza quindi bloccare mai in lettura i dati.
- Read Committed, prevede il rilascio immediato dei dati in lettura e ritarda quelli in scrittura.
- Repeatable Read, vengono bloccati sia i dati in scrittura che quelli in lettura ma solo sulle n-uple della tabella coinvolta.
- Serializable, vengono bloccate interamente gli accessi alle tabelle in gioco, spesso questo è un livello di isolamento inefficiente.

Durabilità o Persistenza

Questa proprietà garantisce che i risultati di una transazione completata con successo siano permanenti nel sistema, ovvero non devono mai più essere persi. Ovviamente c'è un piccolo intervallo temporale tra il momento in cui la base di dati si impegna a scrivere le modifiche e la scrittura di quest'ultime, questo intervallo è un vero e proprio punto debole e quindi dobbiamo sempre garantire, per esempio con un log, che non si verifichino perdite di dati dovuti a malfunzionamenti. Per garantire questa proprietà quasi tutti i DBMS implementano un sottosistema di ripristino (recovery), che garantisce la durabilità anche a fronte di guasti ai dispositivi di memorizzazione, per esempio attraverso l'uso di back-up su supporti diversi oppure journaling delle transazioni.

2.1.3 *Teorema CAP*

Il teorema CAP è stato pensato inizialmente da Eric A. Brewer (da cui il nome di Teorema di Brewer), fondatore di Inktomi e chief scientist di Yahoo!, docente di Informatica presso UC Berkeley.[2] In particolare questo teorema afferma che per un sistema informatico distribuito è impossibile garantire tutte e tre le seguenti garanzie:

- Coerenza. Un sistema è completamente coerente quando è in grado di garantire che, una volta memorizzato un nuovo stato nel sistema, questo è utilizzato in ogni operazione successiva fino alla prossima modifica dello stesso. Quindi, tutte le richieste effettuate tra uno stato e quello successivo, forniscono il medesimo risultato.
- Disponibilità. Un sistema è completamente disponibile quando è sempre in grado di soddisfare le varie richieste oppure erogare i propri servizi.
- Tolleranza alle partizioni. La tolleranza alle partizioni è definita come la proprietà di un sistema di continuare a funzionare correttamente anche in presenza di una serie di fallimenti dell'infrastruttura, fino a che l'intero network fallisca.

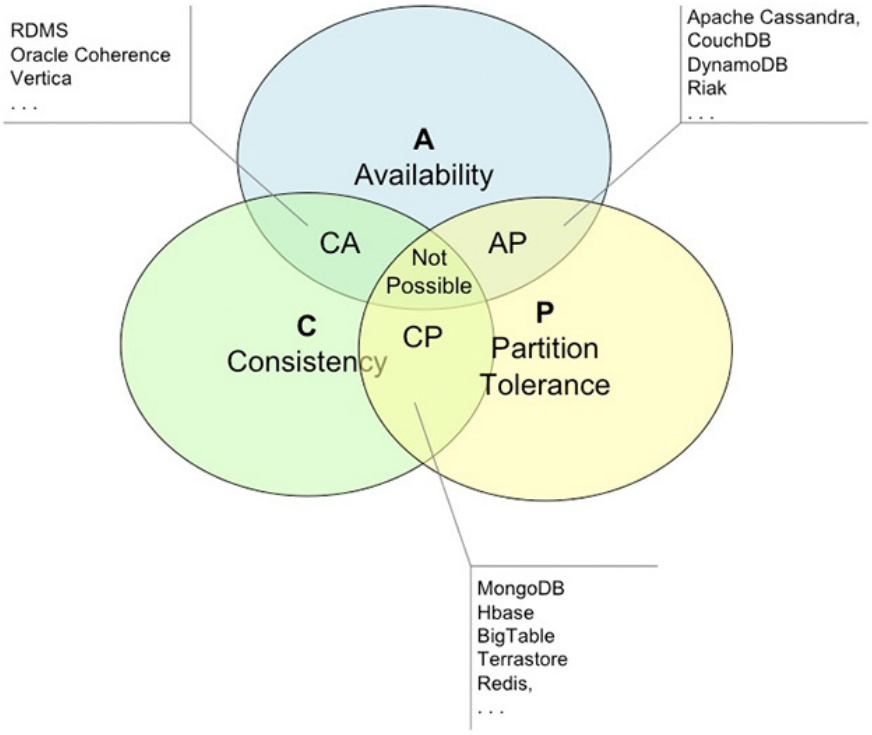


Figure 2.1: Proprietà garantite dal teorema CAP nei principali database

Le soluzioni che in generale prevedono una configurazione in cui un nodo agisce come master e gli altri come slave, ad esempio MongoDB, HBase, e Redis, garantiscono una elevata coerenza e sono molto tolleranti alle partizioni. Infatti i dati sono mantenuti in maniera coerente in tutti i nodi del cluster, e inoltre viene garantita la tolleranza partizioni evitando che i dati possano desincronizzarsi. Altre soluzioni NoSQL, per esempio Cassandra e CouchDB, garantiscono invece una continua disponibilità e la piena tolleranza alle partizioni. Infatti i nodi rimangono sempre online anche quando momentaneamente non lavorano e sarà il processo a risincronizzare i dati e risolvere eventuali conflitti.

2.2 I DATABASE NOSQL

Un punto chiave su cui si è focalizzata l' attenzione è l' analisi delle caratteristiche, dei limiti e dei punti di forza di alcune tra le più utilizzate soluzioni NoSQL disponibili sul mercato al fine di valutare la migliore scelta in riferimento agli obiettivi del problema in analisi. Le scelte riguardano principalmente:

- Formati di file supportati
- Linguaggi supportati
- Indicizzazione
- Principali utilizzatori e casi d'uso
- Costi relativi a licenze

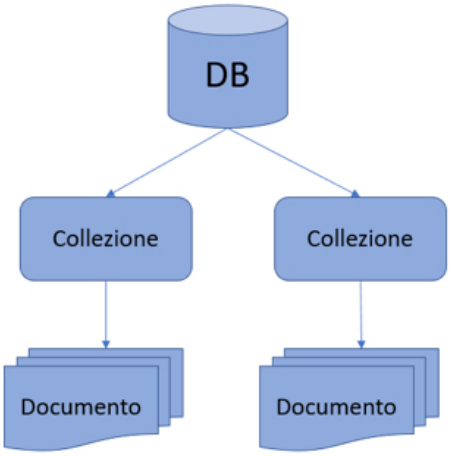


Figure 2.2: Struttura MongoDB

I principali database analizzati sono MongoDB, HBase, Cassandra e Redis.

2.2.1 *MongoDB*

MongoDB è un DBMS non relazionale orientato ai documenti. MongoDB si allontana dalla struttura tradizionale basata su tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico (MongoDB chiama il formato BSON), rendendo l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce. L'unità base di MongoDB è il documento, equivalente alla tupla nel database relazionale.[3]

BSON estende il modello JSON per fornire tipi di dati addizionali, campi ordinati e per essere efficiente nella codifica e decodifica con diversi linguaggi di programmazione. MongoDB utilizza i documenti per memorizzare i records, come tabelle e tuple memorizzano records in un database relazionale. Il formato JSON conferisce facilità d'uso e flessibilità mentre BSON aggiunge velocità e una codifica binaria che occupa poco spazio.

Ogni documento ha una chiave identificativa univoca, chiamata "id", assegnata al momento della creazione del documento. I documenti sono un insieme ordinato di chiavi con valori associati, più documenti possono essere raccolti in collezioni.

Una singola istanza di MongoDB può contenere uno o più database indipendenti, in generale i database contengono tutti i dati relativi ad un'applicazione.

MongoDB supporta C, C++, C#, Java, Node.js, Perl, PHP Python, Motor, Ruby, Scala attraverso i driver ufficiali, inoltre sono presenti driver non ufficiali per integrare gli altri linguaggi.[4]

Un'altra caratteristica di MongoDB è il supporto di indici secondari generici, permettendo una varietà di query veloci e fornendo capacità di indexing uniche, composite, geospaziali e full text.

Un gran numero di aziende oggi utilizzano MongoDB, le più significative sono:

- SAP: usa MongoDB in SAP platform-as-a-service.
- FORBES: memorizza articoli e dati societari in MongoDB.
- SOURCEFORGE: usa MongoDB come storage per i propri dati.
- FOURSQUARE: implementa MongoDB su Amazon AWS per memorizzare località e le registrazioni degli utenti nelle località.
- EBAY: usa MongoDB per i suggerimenti della ricerca e per State Hub, il Cloud Manager interno.

Per un progetto di una piccola impresa i costi iniziali per MongoDB ammontano a 166.000 $, mentre per Oracle ammontano a 820.000 $ I Costi annuali per MongoDB sono in media 129.000 $ invece per Oracle 286.000 $, questo perchè in MongoDB i costi relativi alla licenza sono azzerati. [5]

2.2.2 *Redis*

Redis è un progetto di database open-source. La persistenza non è obbligatoria e quindi di default gli elementi vengono immagazzinati solo nella main memory. Esso si basa infatti su una struttura key/-value: ogni valore immagazzinato è abbinato ad una chiave univoca che ne permette il recupero. Redis supporta diversi tipi di strutture di dati astratti, quali stringhe, elenchi, mappe, set, gruppi ordinati, hyperlogogs, bitmap e indici spaziali. Nonostante il vasto utilizzo dello storage in RAM, l'attuazione di una vera persistenza su memoria di massa resta un compito importante di Redis per garantire il mantenimento del database in casi di riavvio del server, interruzione dell'erogazione di energia elettrica e guasti hardware.[6]

Redis dispone di due meccanismi di persistenza:

- RDB: consiste in una sorta di snapshot che crea un salvataggio su disco dei dati immagazzinati nel server.
- AOF: crea un file di log continuo ove vengono registrate tutte le operazioni di modifica inviate al server.

RE-JSON è un modulo di Redis che implementa lo standard JSON come tipo di dato nativo. Permette di memorizzare, aggiornare e recuperare i valori JSON dalle chiavi Redis (documenti), ReJSON utilizza la sintassi tipo JSONPath per selezionare i documenti all' interno dei documenti.

I documenti sono memorizzati come dati binari in una struttura ad albero, permettendo un accesso veloce ai sotto elementi. Per tutti i tipi dei valori JSON le operazioni atomiche sono tipizzate.

Redis non è esattamente una memorizzazione chiave-valore dato che i valori possono essere anche strutture complesse. Redis offre

soltanto un accesso alla chiave primaria. Comunque, dato che Redis è un server di struttura dati può essere utilizzato per creare indici secondari di differenti tipi incluso gli indici multi colonna.

Twitter, Pinterest sono due delle molteplici aziende che usano Redis, questo perchè è una tecnologia molto flessibile che trova ambiti di applicazione un po' ovunque.Altri utilizzatori sono: Snapchat, Craiglist, Digg, StackOverflow, Flickr. [7]

2.2.3 *HBase*

HBase è una base di dati distribuita open source modellata su BigTable di Google e scritta in Java. Fu sviluppato come parte del progetto Hadoop dell'Apache Software Foundation ed eseguito su HDFS (Hadoop Distributed File System), fornendo capacità simili a quelle di BigTable per Hadoop. HBase è un archivio di dati a valore chiave a colonna ed è stato idolizzato ampiamente a causa del suo rapporto con Hadoop e HDFS, infatti utilizza come dispositivo di storage HDFS ed è adatto per operazioni di lettura e scrittura veloci su grandi insiemi di dati con elevata velocità di trasmissione e bassa latenza di input/output. HBase può essere eseguito su un cluster Hadoop e fornisce un accesso casuale in real-time ai nostri dati.

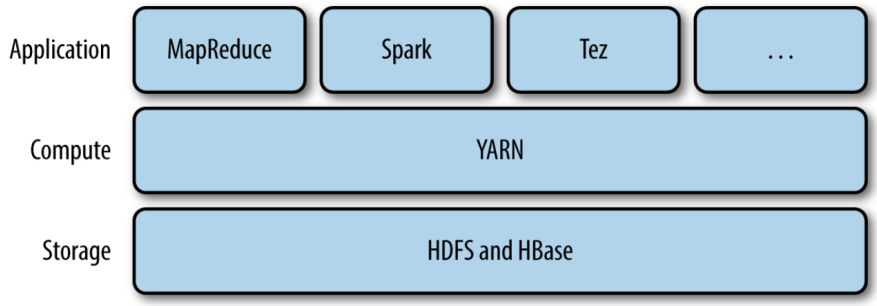


Figure 2.3: I vari livelli dell'architettura Hadoop. Come possiamo vedere HDFS e HBase sono utilizzati come storage.

La scelta tipica è HDFS a causa della stretta integrazione tra HBase e HDFS. Ciò non vuol dire che HBase non può funzionare con nessun altro filesystem, ma non è stato dimostrato che in produzione scala bene come HDFS. [8] HBase fornisce le seguenti caratteristiche:

- Bassa latenza di accesso a una piccola quantità di dati all' interno di un grande dataset.
- Data Model flessibile per lavorare con i dati indicizzati dalla row key
- E' scalabile.

Hbase è interamente basato sul concetto di colonna, anche se non supporta script SQL, possiamo comunque creare una struttura dati che ricorda molto le tabelle SQL, infatti una tabella in Hbase è rappresentata come un insieme di colonne. Una o più colonne possono appartenere a una column family. Una colonna contiene per ogni

riga la chiave identificativa, questo permette all'utente di ricercare dati all'interno della tabella in modo rapido e preciso. Uno dei principali difetti di Hbase, considerando il nostro progetto, è la mancanza di API per il supporto JSON. È pur vero che possiamo creare noi il supporto JSON, per esempio creando un mapper in java, ma rimane comunque un qualcosa di non nativo in Hbase.

Per quanto riguarda l'indicizzazione, HBase non supporta nativamente indici secondari, pero come nel caso di JSON è possibile implementare alcuni indici secondari autonomamente. Ovviamente gli indici secondari richiedono un' ulteriore elaborazione, questo potrebbe così influire notevolmente sulle performances. Inoltre un indice secondario va periodicamente aggiornato quindi si potrebbero usare anche tabelle secondarie che vengono periodicamente aggiornate tramite un apposito processo MapReduce. I linguaggi supportati da HBase sono Java, Python, C/C++, PHP e Ruby. [9] I principali punti di forza di HBase sono la scalabilità (lineare e modulare), la facilità d'integrazione di sorgenti dati che utilizzano differenti strutture e schemi, report interattivi e molto altro riguardante l'analisi dei dati real-time. Proprio per queste ultime caratteristiche menzionate, Hbase è molto usato da diversi siti web orientati ai dati, per esempio la piattaforma di messaggistica Facebook. [10] Altri utilizzatori da menzionare sono:

- Airbnb che utilizza HBase come parte del suo workflow in tempo reale di AirStream
- Imgur usa HBase per alimentare il suo sistema di notifica
- Spotify utilizza HBase come base per i processi di Hadoop e di machine learning.
- Adobe usa HBase come storage, in particolare per contenere i dati utilizzati da flash player.

2.2.4 *Cassandra*

Cassandra è un database management system non relazionale distribuito con licenza open source e ottimizzato per la gestione di grandi quantità di dati. Il codice di Cassandra è stato inizialmente sviluppato all'interno di Facebook (per potenziare la ricerca all'interno del sistema di posta). Cassandra è stato rilasciato poi da Apache Software Foundation per gestire grandi quantità di dati dislocati in diversi server, fornendo inoltre un servizio orientato alla disponibilità. Una delle principali caratteristiche di Cassandra è la tolleranza ai guasti, gestita replicando automaticamente i dati su più nodi. La replicazione dei dati viene gestita da un nodo coordinatore, inserito tipicamente in una struttura ad anello. L'elevata tolleranza ai guasti aumenta quindi la durabilità e la scalabilità di questa tecnologia.[11]
Cassandra fornisce una struttura di memorizzazione chiave-valore, alle chiavi corrispondono dei valori, raggruppati in column family: una column family è formata da un nome che la identifica e da

un array di coppie chiave-valore. Ogni coppia chiave-valore identifica una row, il valore contiene a sua volta una lista di valori. Una tabella in Cassandra è una mappa multi-dimensionale, distribuita, indicizzata da una chiave; il valore è un oggetto altamente strutturato. La tupla in una tabella è una stringa senza restrizioni sulla lunghezza, tipicamente lunga da 16 a 36 byte. Le famiglie di colonne messe a disposizione da Cassandra sono due: tipo semplice e tipo super. Il tipo super può essere rappresentato come una famiglia contenuta in un'altra famiglia, il gradino più alto di questa gerarchia è il keyspace, ogni keyspace contiene il set di column family specifico di un'applicazione; Le applicazioni possono specificare il tipo di ordinamento delle colonne all' interno di una famiglia di colonne super o semplici. Il sistema consente di riorganizzare le colonne in ordine cronologico. L'ordine cronologico è sfruttato, ad esempio, dalle applicazioni di posta, dove i risultati sono sempre visualizzati dal più recente al più vecchio oppure da sensori e applicazioni IOT ad esempio applicazioni meteorologiche, in cui il timestamp è fondamentale. Ogni colonna all'interno di una famiglia è accessibile usando la convenzione column-family.

A partire dalla versione di Cassandra 2.2 è stato introdotto un notevole miglioramento che permette e rende più facile lavorare con i documenti JSON, il miglioramento consiste nella modifica delle funzioni SELECT and INSERT con l'inclusione di una variante incentrata su JSON, inoltre due funzioni native sono state aggiunte per convertire da e verso JSON. Le librerie sono open source e forniscono un supporto contro i fail, dovuti principalmente a guasti hardware, errori di rete oppure di memoria esaurita, inoltre forniscono la capacità di creare pool di connessioni. Gli elenchi dei driver più recenti possono essere reperiti attraverso il sito ufficiale di Cassandra. Ecco un elenco di alcuni dei principali linguaggi supportati: Java, Python, Node.js, PHP, C++, NET, Ruby.[12]
Gli indici integrati in Cassandra rivestono un importantissimo ruolo nel momento in cui abbiamo una tabella con molte righe che contengono il valore indicizzato. Ad esempio, supponiamo di avere una tabella di playlist con un miliardo di canzoni e di voler cercare alcune canzoni di un artista. Molte canzoni condividono lo stesso valore della colonna artista. La colonna dell'artista è quindi un'ottima candidata per un indice. Ovviamente non sempre è consigliabile usare gli indici, per esempio su colonne aggiornate o eliminate frequentemente oppure su colonne di grossa cardinalità perché si interroga un'enorme quantità di records per un numero ridotto di risultati.

Tra i principali utilizzatori citiamo [13]:

- Facebook: usa Cassandra nella Posta in Arrivo nel motore di ricerca, con oltre 200 nodi distribuiti.
- Twitter: passa a Cassandra perché può essere eseguito/lanciato su diversi cluster server ed è capace di mantenere un'innumerevole quantità di dati.

- Cisco's WebEx: usa Cassandra per memorizzare il feed dell'utente e l'attività in tempo reale.
- IBM ha sperimentato un sistema scalabile di email basato su Cassandra.
- Netflix: usa Cassandra per gestire i dati dei suoi sottoscrittori.

2.2.5 *Tabella Comparativa*

Le due tabelle poste sotto rappresentano una comparazione tra i database selezionati. Si può notare che tutte le tecnologie supportano i principali linguaggi di programmazione e l'indicizzazione, nella maggior parte dei casi, è fornita nativamente tramite indici secondari. Le licenze per le tecnologie analizzate sono tutte di tipologia open source.

Table 2.1: Tabella comparativa tecnologie

Tecnologia	JSON	Ling.Supportati	Indicizzazione
MongoDB	SI	Javascript, Python, C, C++, C#, Java, PHP, Ruby,	Si, secondari indexing
Redis	SI	NodeJS, PHP	SI, indici secondari
Cassandra	SI	C#, C++, Java, Javascript, PHP, Python, Perl	NO
CouchDB	SI	C, C#, Java, Javascript, PHP, Python	SI
HBase	SI	C++, C#, Java, PHP, Ruby	NO, secondari indici da default

Table 2.2: Tabella comparativa tecnologie

Tecnologia	Utilizzatori	Licensing
MongoDB	SAP, Forbes, Sourceforce, Foursquare, eBay	OpenSource
Redis	Twitter, Github, Weibo, Pinterest Snapshat, Craiglist,Digg, StackOverflow, Flickr	BSD
Cassandra	Facebook, Twitter, Netflix, Reddit, IBM	Apache
CouchDB	Akamai, Cisco, Linkedin, McGrowHill, Ryanair, Sky, Verizon, Viber, Nielsen, eBay	Apache
HBase	Facebook, Adobe, Twitter, Yahoo!, HP	Apache

2.3 I PROCESSING SYSTEMS

Alla base del nostro progetto, oltre al database NoSQL, dobbiamo utilizzare un sistema per processare i dati. Fondamentale è la scelta di un sistema che processa i nostri dati in maniera rapida ed efficiente. A tal proposito analizziamo due tecnologie diverse che però vanno a svolgere le stesse funzioni, mettendo in evidenza di ognuno di essi la struttura, i vantaggi e gli svantaggi che l'implementazione può portare. Questa scelta dipende anche dal DBMS che andremo a scegliere in quanto deve essere perfettamente compatibile in modo da formare con esso un' applicativo funzionale.

2.3.1 *MapReduce*

MapReduce è un framework per la creazione di applicazioni che elaborano grandi quantità di dati in parallelo. Alla base c'è il concetto di programmazione funzionale. A differenza della programmazione

multithreading dove i thread condividono i dati delle elaborazioni presentando così una certa complessità nel coordinare l'accesso alle risorse condivise, nella programmazione funzionale invece non si ha la condivisione dei dati che sono passati tra le funzioni come parametri o come valori di ritorno. Il concetto base è "divide et impera" dividendo l'operazione in diverse parti processate in modo autonomo. Una volta che ciascuna parte del problema è stata calcolata, i vari risultati parziali sono "ridotti", cioè ricomposti, a un unico risultato finale. La gestione di tutto questo processo viene fatta da MapReduce stesso, il quale si occupa di esecuzione dei task di calcolo, del loro monitoraggio e della ripetizione dell' esecuzione in caso qualcosa andasse storto.[14]

COMPUTE NODE sono dei nodi di calcolo che si trovano assieme ai DataNode di HDFS: lavorando insieme a HDFS, MapReduce può eseguire i task di calcolo sui nodi dove i dati sono già presenti, avendo così un' efficienza maggiore.

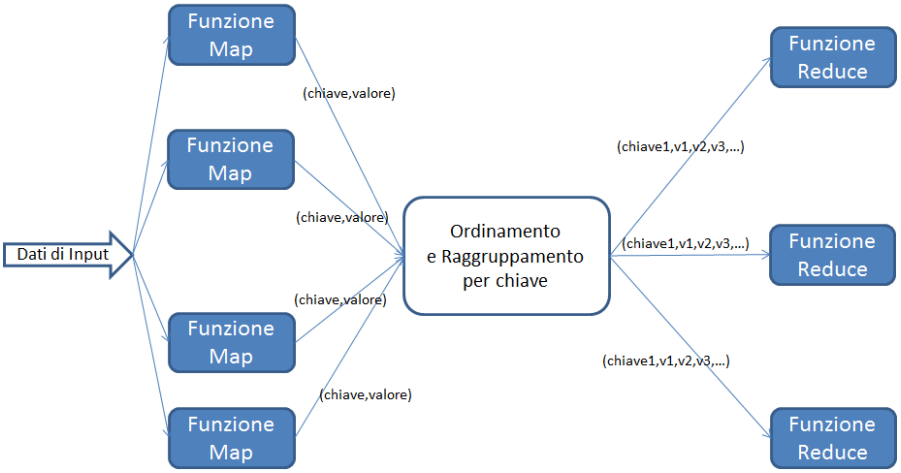


Figure 2.4: Esempio di un Job su MapReduce

Componenti principali di un job MapReduce:

- Dati di input su HDFS
- Funzione Map, che trasforma i dati in input in una serie di coppie chiave/valore.
- Funzione Reduce che per ogni chiave, elabora i valori associati e crea come output una o più coppie chiave valore. Prima di essere eseguita la reduce devo raccogliere le coppie chiave/valore prodotte dalla Map.Le coppie sono ordinate per chiave e i valori con la stessa chiave sono raggruppati.
- l'output scritto su un file HDFS.

I componenti architetturali di MapReduce sono:

- Job Tracker è il componente che gestisce le risorse, cioè CPU e memoria, e del ciclo di vita del job MapReduce. Il JobTracker distruisce i lavori utilizzando una semplice politica: privilegia i

nodi più vicini che contengono dati da elaborare. Se un nodo non può ospitare il task, il Job Tracker stesso si occupa della schedulazione del lavoro e della ripetizione dell'esecuzione di singoli task di MapReduce che hanno qualche errore.

- Task Tracker è il componente che gira sul singolo nodo e che esegue effettivamente il task seguendo gli ordini del JobTracker.

Per utilizzare il framework MapReduce si deve fornire gli input, i file di output, le funzioni map e reduce come implementazioni di interfacce o classi astratte Java. Il client Hadoop fornisce il job come archivio .jar e le configurazioni al JobTracker il quale si occupa di distribuirli ai vari nodi per l'esecuzione. Il JobTracker determina il numero di parti in cui l'input deve essere distribuito e attiva alcuni TaskTracker in base alla loro vicinanza ai nodi che contengono i dati di interesse.

I TaskTracker estraggono poi i dati dalla parte di loro competenza e attivano la funzione map che produce coppie chiave/valore. Dopo che si è conclusa la fase di map, i TaskTracker notificano al JobTracker il completamento del loro lavoro. Il JobTracker attiva così la fase di reduce, nella quale i TaskTracker ordinano i risultati dei mapper per chiave, li aggregano ed eseguono la funzione reduce al fine di produrre l'output, salvato in un file diverso per ciascun TaskTracker.

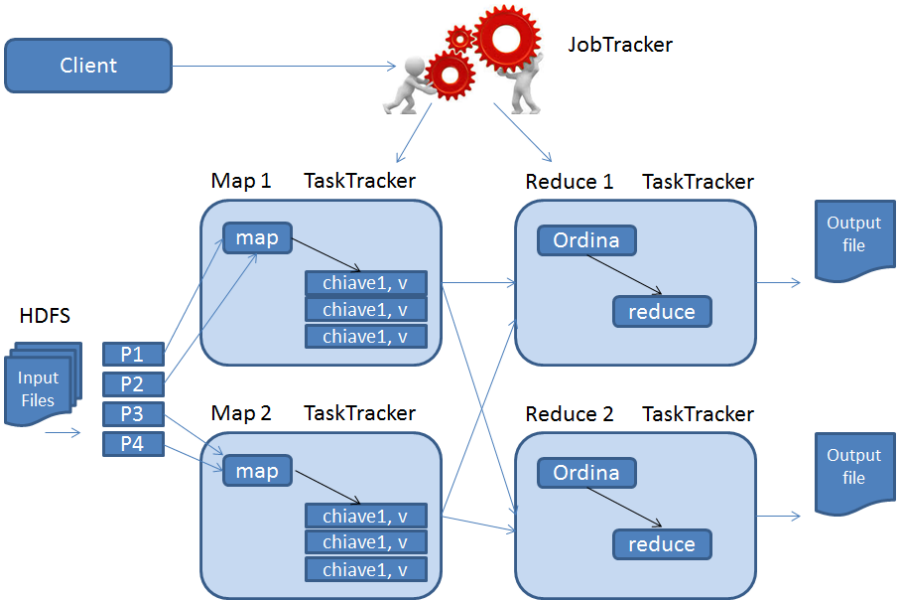


Figure 2.5: Architettura MapReduce

Scenari di utilizzo MapReduce

MapReduce può essere conveniente quando si devono effettuare numerose operazioni sui dati. Esempi di utilizzo sono:

- Creazione di liste di parole da documenti di testo, indicizzazione e ricerca. Ad es. Conteggi,somme estrazione di liste univoche di valori e applicazione di filtri ai dati.

- Analisi di strutture dati complesse,come grafi
- Data mining e machine learning
- Esecuzione di task distribuiti come calcoli matematici complessi e analisi numeriche
- Correlazioni, operazioni di unione, intersezione, aggregazione e join. Ad es. analisi di mercato,analisi predittive e previsione dei trend.

2.3.2 *Apache Spark*

Introduzione

Spark è stato creato nel 2009 come progetto all'interno dell'AMPLab all' università della California, Berkeley. Sin dall' inizio era ottimizzato per essere eseguito in memoria per aiutare a processare dati molto più velocemente che gli approcci alternativi come MapReduce di Hadoop il quale tende a scrivere dati su disco tra ogni stage di processing. Secondo il paper [15] l' esecuzione in memoria può essere molto più veloce rispetto a MapReduce.

Table 2.3: Differenze prestazionali tra Spark e Map Reduce

Piattaforma	Spark	MR	Spark	MR	Spark	MR
Dimensione Input (GB)	1	1	40	40	200	200
Numero di map task	9	9	360	360	1800	1800
Numero di reduce tasks	8	8	120	120	120	120
Tempo di esecuzione (Sec)	30	64	70	180	232	630
Mediana tempi map tasks (Sec)	30	64	70	180	232	630
Mediana tempi reduce tasks (Sec)	6	34	9	40	9	40

I risultati che vediamo nell'immagine sono stati ottenuti utilizzando per entrambe le tecnologie la stessa configurazione hardware, nello specifico è stato utilizzato un cluster con 4 nodi per un totale di 128 core, 760 GB di RAM e 36 TB di storage. La versione utilizzata di JAVA è la 1.7.0, inoltre è stato usata la versione 2.4.0 di Hadoop e la versione 1.3.0 di Spark. I risultati ottenuti rappresentano i tempi per svolgere un operazione di word count su tre diverse quantità di dati, in particolare, 1 GB, 40 GB e 200 GB. Dalla tabella possiamo notare come Spark sia 2.1x , 2.6x e 2.7x più veloce di Map Reduce nell'elaborazione di 1 GB, 40 GB e 200GB di dati. La differenza sostanziale tra le due tecnologie è che, nelle fasi intermedie, Spark non scrive su disco, mentre Map Reduce si.

Nel 2014 Spark fa parte della fondazione Apache ed è adesso uno dei loro progetti di punta. Spark è un motore di data processing general-purpose, utilizzabile in un ampio range di circostanze. Di solito sono associati a Spark interrogazioni interattive su grandi insiemi di dati, il processing di streaming di dati da sensori o sistemi finanziari e processi di machine learning. Gli sviluppatori possono

anche utilizzarlo per supportare altri processi di data processing, beneficiando dell' insieme estensivo di librerie e di API, comprensive del supporto per i linguaggi come Java, Python, R, Scala. Spark è spesso utilizzato accanto a HDFS, il modulo di storage di Hadoop, ma può anche essere integrato altrettanto bene con altri supporti di memorizzazione come HBase, Cassandra, MongoDB, Amazon S3.

Caratteristiche Chiave di Spark

Le caratteristiche principali di Spark sono tre:[16]

SEMPLICITÀ Le capacità di Spark sono accessibili tramite un ricco insieme di API, tutte progettate specificatamente per interagire velocemente e facilmente con i dati in scala.

VELOCITÀ Spark è progettato per la velocità, operando sia in memoria che su disco. Le performance di Spark possono essere anche migliori quando supporta query di dati interattivi memorizzati in memoria.

SUPPORTO Spark include il supporto nativo per una perfetta integrazione con soluzioni di storage dell' ecosistema Hadoop. In aggiunta la Apache Spark Community è grande, attiva e internazionale. Tutti i principali providers commerciali hanno soluzioni basate su Spark.

2.3.3 *Apache Storm*

Nella decade passata c'è stata una rivoluzione nei sistemi di data processing. MapReduce, Hadoop e le tecnologie relative hanno reso possibile memorizzare e processare dati di un ordine di grandezza prima impensabile. Sfortunatamente queste tecnologie di data processing non sono sistemi in real-time nè sono pensati per esserlo. Non ci sono modi per trasformare Hadoop in un sistema realtime poichè il realtime ha un insieme differente di requisiti rispetto al batch processing. Il realtime data processing in scala massiva sta diventando una necessità per le aziende. La mancanza in Hadoop del realtime è colmata da Storm. Apache Storm legge da una sorgente o più sorgenti flussi di dati grezzi in real-time, successivamente li elabora tramite una sequenza di piccole unità di processamento ed emette in uscita l'informazione processata ritenuta utile.

Struttura Storm

La struttura base di Storm è composta da:

- Tupla: È la struttura dati principale in storm, è una lista di elementi ordinati che supporta tutti i tipi di dato. È modellizzata come valori separati da virgole.
- Stream: Sequenza non ordinata di tuple.

I moduli in Storm sono classificati in:

- Spouts: sono le sorgenti dello stream, esse accettano dati da sorgenti di dati grezzi, come ad esempio Twitter Streaming API, oppure file CSV.
- Bolts: sono le unità di processamento logiche. Gli Spout passano i dati ai bolt, i quali processano i dati e creano un nuovo stream in output, il quale può essere processato da uno o più bolt. Possono essere eseguite svariate operazioni tra cui filtering, aggregation, joining, interazione con data sources e databases. L' interfaccia chiave che implementa i bolts è IBolt (in alternativa possono essere utilizzate IRichBolt, IBasicBolt).

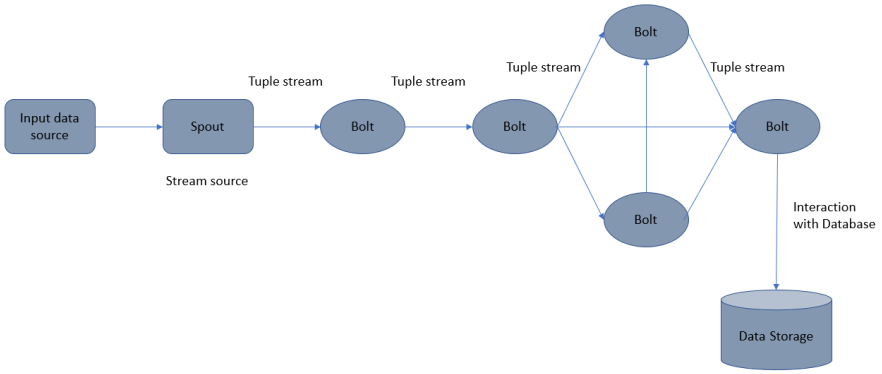


Figure 2.6: Struttura Storm

I Bolt e gli spout connessi insieme formano una topologia, ovvero una struttura rappresentabile come un grafo diretto dove i vertici sono le computazioni e gli archi sono lo stream di dati. La logica applicativa in real-time è specificata nella topologia all'interno di storm.[17]

Una semplice topologia inizia con gli spouts. Lo Spout emette i dati ad uno o più bolts. Un bolt rappresenta un nodo nella topologia avente la più piccola processing logic. Storm mantiene la topologia sempre in esecuzione fino a quando non decido di chiuderla. Il lavoro principale di storm è di eseguire la topologia, inoltre diverse topologie potranno essere eseguite in un tempo stabilito. I Bolt e gli Spout devono essere eseguiti in un certo ordine per funzionare correttamente. L'esecuzione di ogni singolo spout e bolt da parte di Storm è chiamata Task. Ad un certo tempo ogni spout e bolt possono avere più istanze in esecuzione su thread multipli separati, inoltre la topologia è eseguita in maniera distribuita su worker nodes multipli. Storm ha il compito di distribuire i tasks uniformemente su tutti i worker nodes. Il ruolo del worker node è restare in ascolto per i jobs e iniziare, o stoppare i processi ogni volta che un nuovo job arriva.

Struttura interna Storm

Apache Storm internamente ha due tipi di nodi:

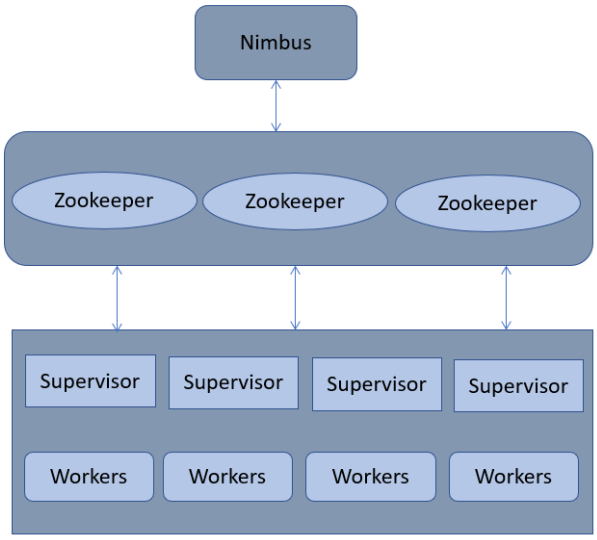


Figure 2.7: Struttura Storm

- Nimbus, ovvero il componente centrale di Storm, il suo compito principale è eseguire la topologia di storm. Esso analizza la topologia e raccoglie il task da eseguire, successivamente distribuisce il task ad un supervisor disponibile. Esiste solo un Nimbus, tutti gli altri nodi sono chiamati Workers nodes.
- Il supervisor ovvero il nodo che avrà uno o più worker process, delegherà i task ai worker processes. Il worker process genererà tanti executors quanti ne ha bisogno per eseguire il task, inoltre esso eseguirà task relativi a una specifica topologia. Per eseguire un task il worker process crea diversi executors e chiede loro di eseguire i task, in questo modo un singolo worker process avrà molteplici executors.

La struttura interna di un cluster tipico è formata da:

- Un Nimbus
- Zookeeper (utilizzato per coordinare Nimbus e i supervisors)
- Uno o più supervisor

ZOOKEEPER FRAMEWORK è un servizio utilizzato da un cluster (gruppo di nodi) per coordinarsi tra di loro e mantenere condivisi i dati con tecniche di sincronizzazione robuste. Il Nimbus è stateless quindi dipende dal Zookeeper per monitorare lo stato del working node, inoltre aiuta il supervisor a interagire con il nimbus. (mantenendo lo stato del nimbus e del supervisor)

Storm è stateless, anche se questa caratteristica può sembrare uno svantaggio, in realtà compone il principale vantaggio di storm, in quanto gli permette di processare i dati in real time nel miglior modo possible e in maniera veloce.

Tipologie di Grouping

Parte della definizione di una topologia sta nello specificare per ciascun Bolt quali flussi deve ricevere come input. Un grouping di stream definisce come deve essere partizionato quel flusso tra le attività dei Bolt. In storm ci possono essere 8 tipologie di grouping, inoltre è possibile creare un raggruppamento di stream personalizzato implementando l'interfaccia di CustomStreamGrouping. I 4 principali grouping sono:

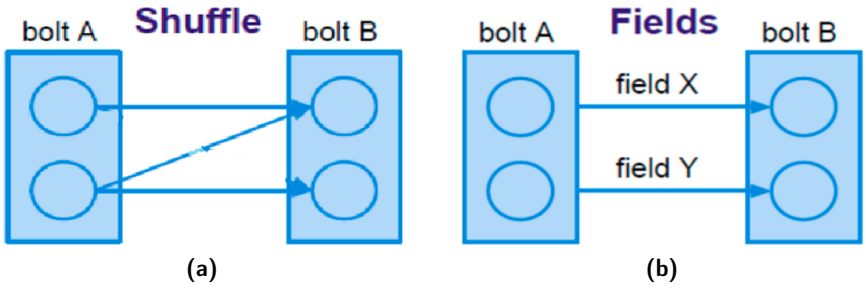


Figure 2.8: Shuffle e Fields Grouping

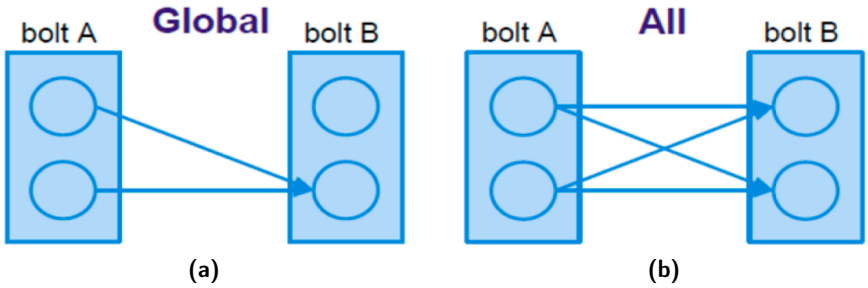


Figure 2.9: Global e All Grouping

SHUFFLE GROUPING Un numero uguale di tuple è distribuito random su tutti i workers che eseguono i bolts.

FIELD GROUPING I campi con gli stessi valori nelle tuple sono raggruppate insieme e le tuple rimanenti rimangono fuori. Le tuple con lo stesso campo sono inviate allo stesso workers che esegue i bolts.

GLOBAL GROUPING Tutti gli streams possono essere raggruppati e inoltrati a un bolt, il quale invia le tuple generate da tutte le istanze di una sorgente a un singola target instace.

ALL GROUPING Invia una singola copia di ogni tupla a tutte le istanze del bolt ricevente. E' utilizzato per inviare segnali ai bolts, utile in operazioni di join.

Workflow tipico

I principali step di un workflow tipico di Apache Storm, possono essere:

- Nimbus aspetta che gli si sottometta la Storm Topology
- dopo aver ottenuto la topologia, la analizza e ottiene tutti i tasks che devono essere avviati e l'ordine di esecuzione.
- Il nimbus eventualmente distribuisce i tasks ai supervisor disponibili.
- In un time interval particolare tutti i supervisor inviano un segnale per informare che sono ancora vivi.
- Quando un supervisor muore senza aver inviato il segnale al nimbus, quest'ultimo assegna i tasks ad un altro supervisor.
- Quando il nimbus muore i supervisor funzionano lo stesso sui task già assegnati
- Quando tutti i task sono completati il supervisor aspetterà che un nuovo task entri.
- Nel frattempo il nimbus morto sarà riavviato automaticamente dal service monitoring tools.
- Il nimbus riavviato continuerà da dove si era fermato.
- In modo simile il supervisor morto può anche essere riavviato automaticamente.
- Dato che sia il nimbus che il supervisor possono essere riavviati automaticamente, ed entrambi continueranno come prima, Storm sicuramente processa tutti i task almeno una volta.
- Una volta che tutte le topologie sono processate, il nimbus aspetta che arrivi una nuova topology e similarmente il supervisor aspetta nuovi tasks.

Modi di Esecuzione Storm

Di default ci sono due modi in un cluster Storm:

LOCAL MODE Questo modo è utilizzato per sviluppo, testing e debugging perchè è il modo più veloce per vedere lavorare insieme tutti i componenti della topologia. In questo modo possiamo regolare i parametri che ci fanno vedere come la nostra topologia viene eseguita in differenti ambienti di configurazioni. Nel local mode, le storm topologies sono eseguite in una macchina locale in una singola JVM.

PRODUCTION MODE In questo modo, inviamo la nostra topology al working storm cluster, il quale è composto da molti processi, di solito eseguiti su differenti macchine. Un working cluster verrà eseguito indefinitamente fino a quando non è spento.

Distributed Messaging System

Il Distributed Messaging è basato sul concetto dell' accodamento di messaggi affidabili. I messaggi sono messi in coda in modo asincrono tra le applicazioni client e i messaging systems. Un distributed messaging system fornisce i benefits di affidabilità, scalabilità e persistenza. La maggior parte dei pattern di messaging seguono il modello publish-subscribe (Pub-Sub), dove i mittenti dei messaggi sono chiamati publishers e coloro che ricevono i messaggi sono chiamati subscribers. Una volta che il messaggio è stato pubblicato dal mittente, i subscribers possono ricevere il messaggiio selezionato con l' aiuto di un opzione di filtering. Di solito abbiamo due tipi di filtro il topic-based filtering e il content-based filtering. Si noti che il modello pub-sub può comunicare solo via messaggi. Un esempio nella vita reale è Dish TV, i quali pubblicano differenti canali come sport, film, musica etc... e ognuno si può sottoscrivere a quelli che ritiene più interessanti e ottenerli ogni volta che sono disponibili.

Use Cases di Storm

- L'infrastruttura di Twitter, inclusi i sistemi di database (Cassandra, Memcached, ecc.)
- L'infrastruttura di messaggistica, Mesos e i sistemi di monitoraggio / avviso
- Yahoo! sta sviluppando una piattaforma di nuova generazione che consente la convergenza di elaborazione di grandi dati e bassa latenza.
- Groupon utilizza Storm per analizzare, pulire, normalizzare e processare grandi quantità di data points non unici con una bassa latenza e un elevato throughput.
- Alibaba utilizza Storm per elaborare il registro delle applicazioni, modificare i dati nel database e per fornire statistiche in tempo reale per le applicazioni di dati.

3

ANALISI PROBLEMA

In questo capitolo inzialmente viene fatta una overview sulla DEM (Direct Email Marketing) analizzandone caratteristiche, costi ed efficacia. Successicamente viene presentata la situazione attuale dell' azienda DEM Joiner mettendo in luce le problematiche di integrazione dei dati, la qualità dei dati, il data lake e i requisiti.

3.1 DIRECT EMAIL MARKETING

Il problema principale è stato quello di integrare dati, con uno stesso schema, provenienti da sorgenti differenti. Nello specifico l'integrazione riguarda un caso di DEM, cioè di Direct Email Marketing.

L'Email Marketing è una strategia di marketing utilizzata per inviare messaggi di posta elettronica per promuovere servizi o prodotti. I messaggi inviati possono riguardare anche altri aspetti che riguardano un brand e quindi eventi e iniziative particolari o, semplicemente, nuovi articoli pubblicati su un blog. Per poter inviare dei messaggi, si deve ovviamente disporre di una lista di iscritti e dei loro indirizzi di posta elettronica. Una volta che si è in possesso della lista di iscritti, si potrà iniziare la campagna di email marketing ed inviare messaggi riguardanti i prodotti che si intende promuovere.

Partiamo dall' elenco dei benefici che una campagna di email marketing può avere:[18]

- Riduzione del tempo e della difficoltà di gestione della comunicazione con i possibili client.
- Comunicazione in tempo reale.
- Personalizzazione e segmentazione.
- Riduzione dei costi.
- Coinvolgimento degli iscritti.
- Possibilità di tracciare l'efficacia della comunicazione.

Con l'email marketing i tempi e le difficoltà di gestione vengono abbattuti. Infatti, si può organizzare una comunicazione efficace in meno di un'ora, in generale le campagne di marketing condotte tramite posta tradizionale, volantini, giornali, riviste e tv devono essere fatte con lo schema "uno e per tutti". Si deve cioè veicolare un messaggio comprendente il più vasto target possibile. Questo significa che non può essere personalizzato a dovere per ogni singolo destinatario. L'obiettivo del progetto è fornire dei dati che permettano di creare messaggi con target specifici, avendo una maggiore efficacia.

Per esempio Groupon basa il contenuto dei propri messaggi secondo quattro criteri principali, segmentando i dati in base a:[18]

- Geografia. Groupon si preoccupa di inviare degli annunci sulle "cose da fare". Il livello di segmentazione è talmente alto che Groupon è in grado di inviare degli annunci mirati su eventi, concerti, o ristoranti che operano in una determinata area geografica.

- Tipo di prodotto. In questo caso si parla di viaggi, elettronica e corsi di formazione. Quindi offerte non strettamente legate all'area geografica.

- Tempo trascorso dall'iscrizione. Groupon gestisce i nuovi iscritti tutti allo stesso modo. Per esempio, i nuovi iscritti ricevono una serie di 12 messaggi nei primi 45 giorni di iscrizione. L'obiettivo in questo caso è di rendere entusiasti e coinvolgere i nuovi iscritti con il brand.

- Dati personali. Questo è il livello più profondo della segmentazione adottata. Infatti, Groupon, non solo invia dei messaggi personalizzati secondo l'età, il sesso e la città di residenza, ma utilizza le informazioni che si è fornito acquistando servizi di terze parti, in base ai click e a quello che i clienti hanno acquistato in precedenza sul loro portale.

L'evoluzione recente si sta concentrando sempre più sulla qualità del contatto (profilazione delle utenze, cura della soddisfazione del cliente), rispetto agli invii massivi di posta che avevano caratterizzato l' e-mail marketing degli esordi.

Costo ed efficacia

Il costo è davvero basso perchè consiste solamente nell'invio di messaggi a utenti selezionati secondo target specifici, inoltre l'efficacia è sempre maggiore rispetto alla sponsorizzazione di una newsletter. Il costo però, potrebbe aumentare nel momento in cui si ha l' accesso a un database con una qualità dei dati maggiore da cui è possibile estrarre dei target sfruttando dei criteri di selezione più profondi. In ogni caso il costo di una DEM è sempre maggiore rispetto all'acquisto di uno spazio pubblicitario in una newsletter.

L'efficacia di una DEM è strettamente legata a tre aspetti principali: individuazione dei target, selezione e contenuto del messaggio pubblicitario. Inoltre, una personalizzazione del messaggio pubblicitario potrebbe accrescere molto l'efficacia della comunicazione. Le unità di misura dell'efficacia di una DEM generalmente sono due, il numero di click (CTR) e il numero di contatti generati (lead).

Data Quality

DATA QUALITY CONTROL è il processo che permette di controllare l' utilizzo di dati con misure di qualità note per un'applicazione o un processo.

Questo processo è di solito eseguito dopo il processo QA (Data Quality Assurance), il quale consiste nel trovare le inconsistenze e le correzioni da effettuare. I processi Data QA forniscono le seguenti informazioni ai Data Quality Control (QC):

- Livello di inconsistenza
- Completezza
- Accuratezza
- Precisione

In questo specifico problema, le sorgenti dati sono database contenenti informazioni riguardanti gli utenti. I database sono forniti da piccole realtà commerciali su tutto il territorio Italiano.

ACCURATEZZA è definita come la distanza tra un valore v e un valore v' considerato come la corretta rappresentazione del fenomeno reale che v intende esprimere. Considerando il valore v e v' allora abbiamo due tipi di accuratezza diversi:

- Accurratezza Sintattica. Si verifica che il valore dell'attributo v sia presente nell'insieme dei valori di dominio D. È facile immaginare che tale valore non risulterà presente in D, e si potranno quindi ottenere dei valori vicini (in questo caso la vicinanza può essere realizzata come il numero di lettere necessarie per rendere i due valori uguali, ma si possono definire altre metriche). Nel caso dell'accuratezza sintattica, però, non si è interessati alla valutazione del valore v con il valore reale v' (cioè con il vero nome dell'individuo che si vuole rappresentare) ma con l'insieme di tutti i valori di dominio dell'attributo v.
- Accuratezza Semantica. In questo caso si valuta l'accuratezza del valore v paragonandolo con la sua controparte reale v'. È chiaro che è fondamentale conoscere qual è il vero nome dell'individuo che l'attributo v vuole esprimere. Diversamente dall'accuratezza sintattica, che misura la distanza tra il valore osservato e il valore più vicino dei valori appartenti al dominio sottoforma di valore numerico, l'accuratezza semantica fornisce una valutazione dicotomica: o v è accurato quanto il valore reale o non lo è, indipendentemente dalla distanza tra i valori v e v'. Come conseguenza, grazie all'accuratezza semantica, si esprime intrinsecamente il concetto di correttezza del dato.

COMPLETEZZA è definita come il livello di ampiezza, profondità ed appropriatezza di un dato in funzione dello scopo che ha. [19] Per meglio descrivere la dimensione della completezza, a titolo esemplificativo, si può immaginare la struttura che memorizza i dati come una tabella (relazione): le colonne rappresentano gli attributi dell'oggetto che si vuole memorizzare, mentre le tuple della tabella rappresentano le diverse osservazioni dell'oggetto.

Ad esempio, nel caso dell'anagrafica comunale, si può immaginare il dato sulla popolazione anagrafica come una tabella in cui le colonne modellano le informazioni anagrafiche dei cittadini (es., nome,cognome, sesso, data di nascita, etc) mentre ogni riga rappresenta un cittadino diverso. È possibile quindi distinguere diverse tipologie di completezza del dato:

- La completezza di colonna, che misura la mancanza di specifici attributi o colonne da una tabella;
- La completezza di popolazione, che invece analizza le tuple mancanti in riferimento ad una popolazione osservata.

Risulta evidente che alcuni livelli di completezza sono difficili da valutare.

CONSISTENZA è definita come la violazione di una o più regole semantiche definite su un insieme di dati. Anche in questo caso, è possibile identificare vari livelli di consistenza:

- Consistenza di chiave: è la più semplice delle forme di consistenza e richiede che, all'interno di uno schema di relazione (una tabella), non vi siano due tuple con il medesimo valore di un attributo usato come chiave.
- Consistenza di inclusione: ne è un classico esempio è la "foreign key" di una relazione. Richiede che ogni valore di un attributo (colonna) di una relazione (tabella) esista come valore di un altro attributo in un'altra (o nella stessa) relazione.

La situazione allo stato attuale quindi non rispetta queste caratteristiche, in quanto i dati non vengono processati e potrebbero essere inconsistenti oppure incompleti.

3.2 DEM JOINER

La DEM Joiner è un'azienda che ha lo scopo di aggregare dati provenienti da diversi publisher.

PUBLISHER sono aziende che si occupano della raccolta dei dati e li mettono a disposizione di DEM Joiner.

I dati sono ricevuti attraverso vari file CSV che periodicamente vengono inviati dai publisher. DEM Joiner, quindi, riceve enormi quantità di dati aventi la stessa struttura, ma con la presenza di alcuni duplicati e/o errori ortografici e/o attributi non validi. Allo stato attuale DEM Joiner non sfrutta pienamente i dati ricevuti, infatti riesce soltanto a rendere univoco un utente presente in più publisher, tramite un algoritmo di deduplica. Ovviamente questo sistema, di tipo relazionale, presenta evidenti lacune e quindi dovrà essere migliorato, per esempio i dati processati si potrebbero inserire in un database integrato che sarà facilmente interrogabile in modo da fornire ai possibili clienti un comodo e facile metodo che permetta di ottenere dati segmentati seguendo specifici criteri.

Esempio

Azienda A : database utenti iscritti alla newsletter di un sito operante solo in Lombardia.

Azienda B: database di utenti partecipanti ad un evento in Emilia Romagna. Alcuni utenti dell'azienda A e dell'azienda B possono essere uguali.

L'azienda X operante su tutto il territorio nazionale vuole creare una nuova campagna marketing basata su DEM.

Sarebbe dispendioso dal punto di vista economico e del tempo andare a stringere accordi commerciali con le singole aziende possedenti i dati, quindi avere un'azienda con un database integrato diventa di fondamentale importanza.

3.3 INTEGRAZIONE

Il processo che porta alla creazione di un database integrato, presenta diverse problematiche che devono essere affrontate e risolte. Molte volte non c'è una soluzione univoca al problema, ma è il progettista a scegliere quella che ritiene più significativa. Di seguito riportiamo le principali problematiche che si sono venute a creare.

Validità dei dati

Analizzare la validità ed effettuare correzioni, se necessario.

L' analisi si divide in due tipologie: la prima si occupa della correzione di eventuali errori presenti nell'attributo dal punto di vista ortografico, spesso commessi in fase di inserimento.

ESEMPIO L' utente potrebbe sbagliare a inserire la città, scrivendo "Miano" al posto di "Milano".

La seconda invece si occupa di unificare sotto un'unica convenzione diverse rappresentazioni dello stesso attributo.

ESEMPIO Nel campo "sesso" in una sorgente il valore "Maschio" potrebbe essere indicato con il valore "M", mentre in un altra con il valore "0".

Il progettista deve scegliere una convenzione rappresentativa per memorizzare tale valore.

Dati Duplicati

E' possibile che in ingresso arrivino dati che sono già presenti nel database integrato, ovvero dati che hanno lo stesso identificativo. Ad esempio nel nostro caso possono arrivare dati con lo stesso "hash". Dobbiamo, quindi, provvedere all'integrazione, aggiornando gli attributi/valori già presenti e inserendo quelli assenti. L' aggiornamento

non prevede la perdita di informazione, ovvero non si sostituisce il nuovo dato con quello esistente, ma si tiene traccia degli attributi e della loro frequenza.

3.4 SCALABILITÀ

Il nostro sistema deve essere in grado di crescere o diminuire in base alle necessità e delle disponibilità.

La scalabilità può essere di due tipi:

SCALABILITÀ VERTICALE La scalabilità fatta in verticale fa riferimento all'aggiunta di risorse con l'obiettivo di aumentare, al contempo, la capacità. Per esempio: aggiungere più CPU al proprio server, oppure espandere la memoria del proprio storage o della propria memoria RAM.

SCALABILITÀ ORIZZONTALE fa riferimento all'aggiunta di nuove unità tra di nuove unità messe tra di loro in parallelo perché funzionino come una sola. Esempi di scalabilità orizzontale sono molto comuni: clustering, sistemi distribuiti e load-balancing. La scalabilità orizzontale si focalizza sia sull' aspetto hardware che sull' aspetto software.

Uno dei principali aspetti da considerare nella scelta della scalabilità da implementare è il costo, mentre la scalabiltà verticale è più facile da implementare, la scalabilità orizzontale risulta meno costosa, perchè non richiede l'acquisto di hardware costoso, ma al contempo richiede più lavoro.

Nel nostro caso è cruciale la scalabilità orizzontale, dobbiamo essere in grado di aggiungere o togliere unità in modo trasparente.

3.5 DATA LAKE

All' interno del nostro sistema prevediamo di inserire un data lake per permettere, in futuro, un processo integrativo diverso da quello effettuato.

DATA LAKE è un metodo per memorizzare i dati all'interno di un sistema o un repository, nel suo formato naturale, che facilita la collocazione di dati in varie forme di schema e struttura, di solito oggetti blob o file.

Il data lake è un archivio unico contenente tutti i dati i dati grezzi dell'azienda (che implica la copia esatta dei dati del sistema di origine). Il data lake comprende dati strutturati di database relazionali (righe e colonne), dati semi-strutturati (CSV, log, XML, JSON), dati non strutturati (email, documenti, PDF) e anche dati binari (immagini, audio, video) creando un archivio di dati centralizzato che accetta tutte le forme di dati. Si può pensare che le nozioni di data lake e data warehouse (e data mart) siano simili, ma c'è una distinzione fondamentale tra i termini. Il data lake immagazzina dati grezzi in qualsiasi formato venga fornito dalla sorgente dati. Non ci sono assunzioni

sullo schema dei dati e ogni sorgente può utilizzare qualsiasi schema si desidera. Con un data warehouse i dati in arrivo sono puliti e organizzati in uno schema unico e consistente prima di essere inseriti nel data warehouse, all' opposto con il data lake i dati in arrivo vanno in un "lake" in forma grezza, dopodichè vengono selezionati e organizzati singolarmente per ogni necessità. Molte iniziative di data warehouse non vanno molto lontano perchè hanno dei problemi di schema, cioè tendono ad avere lo schema unico per necessità di analisi, ma il data model unificato non è praticabile tranne per le piccole imprese.[20]

3.6 REQUISITI

L'efficienza è uno dei requisiti fondamentali che il sistema deve possedere, infatti i tempi di aggiornamento devono essere il più possibile ridotti quando la quantità di dati tende ad aumentare.

EFFICIENZA è la capacità di un sistema di utilizzare meno risorse possibili. I fattori che influenzano l' efficienza sono il tempo di utilizzo della CPU e lo spazio occupato dal programma e dai dati in memoria.

Potrebbe, per esempio, essere accettabile processare una tupla nel ordine di qualche millisecondo. In particolare le operazioni più onerose nel processamento sono quelle di lettura e scrittura sul database. Nell' architettura del progetto l'efficienza è il requisito fondamentale, perchè essendo enorme la mole di dati, i tempi di processamento potrebbero diventare grandi, rendendo il tutto poco, se non addirittura per niente, funzionale. Bisogna sempre tener conto delle condizioni in cui si effettuano le prove, in particolare bisogna considerare:

- I dati in ingresso, in particolare la loro dimensione che spesso viene utilizzata come funzione di costo per ottenere dei risultati oggettivi.
- Significatività dei dati in ingresso, e quindi diverse esecuzioni con diversi dati in ingresso.

4 SOLUZIONE PROPOSTA

In questo capitolo, dopo un attenta analisi dei vantaggi e degli svantaggi di ogni soluzione, vengono commentate e giustificate le scelte fatte. Dopo questa analisi, per questo specifico progetto, si è deciso di utilizzare MongoDB come DBMS e Spark come Data Processing System. Infine vengono descritti i flussi di lavoro dei vari moduli del progetto che sono stati implementati.

4.1 DATABASE

Il database è uno dei componenti più importanti, quindi si è cercato di ottenere il massimo profitto da ogni singola tecnologia utilizzata. Nella scelta del DBMS abbiamo tenuto conto di alcune caratteristiche fondamentali, il DBMS ideale infatti deve:

- Essere scalabile.
- Avere uno schema flessibile.
- Essere facilmente integrabile con i principali linguaggi di programmazione.
- Avere un costo ridotto.

Il DBMS che meglio rispecchia le caratteristiche sopra elencate è stato MongoDB, il paragone è stato fatto tra vari DBMS ma infine la scelta è stata fatta tra MongoDB e HBase.

4.1.1 *Soluzioni scartate*

Prima di arrivare a scegliere uno specifico DBMS, sono state analizzate quattro tecnologie. Come descritto nel capitolo 2, sono state considerate alcune caratteristiche e requisiti che il DMBS deve assolutamente possedere.

Cassandra

Un'altra alternativa che è stata scartata è Cassandra. Cassandra possiede gran parte dei requisiti richiesti e prendendo in considerazione il teorema del CAP garantisce la disponibilità e la tolleranza alle partizioni. La struttura è formata dalle column family e dalle row-key. A differenza di HBase la struttura è decentralizzata, quindi ogni nodo può eseguire qualsiasi operazione. Una delle principali limitazioni di

questa tecnologia riguarda la dimensione di ogni row, infatti se è maggiore di una quantità dell'ordine di dieci Megabytes si avranno problemi con i tempi di compattazione dei dati, ovvero i tempi necessari per ridurre lo spazio utilizzato su disco attraverso l'eliminazione dei dati non usati. Cassandra possiede una forte indicizzazione che però rappresenta anche una grande limitazione, infatti possiamo creare indici secondari sulle columns family, poco utili però nel caso in cui abbiamo colonne con una grande cardinalità.
Inoltre il partizionamento random dei dati, usato da Cassandra, rende l'aggregazione particolarmente difficile, spesso infatti bisogna ricorrere ad altre tecnologie, per esempio Storm o Hadoop.

Hbase

E' stato analizzato HBase come possibile soluzione il quale possiede una struttura molto simile a Cassandra basata sulle columns, columns family e sulle row-key. Per quanto riguarda l'interrogazione, con l'ausilio di Apache Phoenix, è possibile creare delle queries con una sintassi simile a SQL. Questa tecnologia non rispetta tutte le proprietà ACID, garantisce pero la disponibilità e la consistenza del teorema del CAP. A differenza di Cassandra, HBase ha una struttura centralizzata basata su un master server responsabile di monitorare tutti i vari region server presenti sul cluster. HBase è perfetto per creare real-time query, tale aspetto non è però significativo nel progetto. Un difetto di HBase è rappresentato anche dalla complessità dell'istallazione, infatti ha necessariamente bisogno di un server distribuito, Zookeeper, il quale gestisce le varie configurazioni e la manutenzione. Le performance di HBase sono ottime in lettura, ma non garantisce lo stesso per quelle in scrittura. La lettura nel database è resa particolarmente performante grazie al partizionamento ordinato dei dati. In parte l'aggregazione dei dati è supportata nativamente da HBase, per esempio le operazioni di MAX, MIN, SUM, AVG e STD. Inoltre, mentre Cassandra e MongoDB sono tecnologie fortemente tipizzate, quindi supportano i principali tipi per esempio string, integer, double e boolean, Hbase non possiede questa caratteristica.

Redis

Redis sarebbe stato una soluzione abbastanza valida, essendo l'intero storage in memoria RAM, sicuramente sarebbe stata la soluzione con le performance migliori. Questa tecnologia non possiede però API per permettere la definizione, da parte dell'utente, di metodi per le operazioni di Map Reduce, a differenza delle altre tecnologie che gestiscono questa caratteristica. Essendo tutto salvato in RAM è inoltre davvero difficile gestire la persistenza in memoria, bisogna quindi implementare vari metodi, come ad esempio snapshots oppure logs per garantire a pieno questa caratteristica. Redis è debolmente tipizzato, alcuni tipi supportati sono: strings, hashes, lists, sets and sorted

sets. Inoltre, la struttura di MongoDB, basata sul documento, è più adatta in questo progetto rispetto alla struttura ad albero usata da Redis.

4.1.2 *Soluzione scelta*

MongoDB è un database open-source basato sui documenti che fornisce performances molto buone, alta disponibilità e scalabilità. Inoltre questa tecnologia supporta diversi metodi che permettono la replicazione dei dati basata su una struttura master-slave, che crea quindi diverse copie dei dati su diversi server rendendo spesso l'integrazione dei dati, aspetto centrale nel nostro progetto, facile e veloce. Lo schema dei dati in MongoDB è libero, però spesso come nel nostro specifico progetto viene usato uno schema fisso per tutti i documenti appartenenti a una determinata collection.

Struttura

Il database ha una struttura flessibile, cioè in futuro può essere cambiata senza modificare i vecchi documenti. La struttura attuale è la seguente:

```
{
        "hash": "5288b98805a9b1d818d0386bfc9943c0",
        "citta": [{
                "dataIns": "2017-11-17",
                "valore": "MI",
                "dataUltimoIns": "2017-11-17",
                "freq": 3
        }],
        "CAP": [{
                "dataIns": "2017-11-17",
                "valore": "20130",
                "dataUltimoIns": "2017-11-17",
                "freq": 3
        }],
        "naz": [{
                "dataIns": "2017-11-17",
                "valore": "Montenegro",
                "dataUltimoIns": "2017-11-17",
                "freq": 3
        }],
        "sesso": [{
                "dataIns": "2017-11-17",
                "valore": "M",
                "dataUltimoIns": "2017-11-17",
                "freq": 2
        }, {
                "dataIns": "2017-11-17",
                "valore": "F",
                "datUltimoIns": "2017-11-17",
                "freq": 1
```

```
        }],
        "dataNascita": [{
                "dataIns": "2017-11-17",
                "valore": "1962-12-11",
                "datUltimoIns": "2017-11-17",
                "freq": 2
        }, {
                "dataIns": "2017-11-17",
                "valore": "1965-1-7",
                "datUltimoIns": "2017-11-17",
                "freq": 1
        }]
}
```


La struttura è formata da un hash, univoco, che codifica i dati personali di un individuo, in particolare dati sensibili che non possono essere condivisi per motivi di privacy. L' hash diventa di fondamentale importanza per identificare un documento e da la possibilità di creare delle query a favore di analisi successive. Ogni documento contiene una lista di attributi: CAP, nazionalità, sesso e data di nascità. E' stato assegnato ad ogni attributo non un unico valore, ma una lista di valori. Quest'ultima scelta è stata fatta per mantenere un certo livello di qualità del dato inserito. Introducendo una lista di valori ognuno con un campo frequenza che indica il numero di volte che un determinato valore è stato inserito, la probabilità che un valore non rispecchi la realtà diminuisce enormemente. Per esempio: un singolo individuo può avere inserimenti sul campo sesso contrastanti, alcuni possono affermare che l'individuo è maschio altri che è femmina, così mantenendo traccia della frequenza il problema viene quasi eliminato del tutto. Un altro importante campo che possiede un singolo valore è la data riguardante l'ultimo inserimento. La data dell'ultimo inserimento è importante perchè fornisce un livello di attendibilità basato sull'evoluzione del dato e come esso cambia nel tempo. Per esempio, vengono analizzati alcuni valori che confermano che un determinato individuo vive a Milano fino a una determinata data e successivamente a quest'ultima si trova una città inserita diversa da quella precedente, allora con una buona probabilità l'individuo si è trasferito.

La struttura allora, presenta le seguenti caratteristiche:

- Flessibile: infatti è possibile inserire nuovi attributi in modo trasparente, lasciando inalterati i vecchi documenti.
- Semplice: Per inserire un nuovo attributo basta aggiungerlo al nuovo documento che si vuole inserire.

  In modo analogo si può anche andare a rimuovere alcuni attributi che non sono più utilizzati, senza stravolgere il database. Si noti che in MySQL, ma anche in HBase, questo non può essere fatto a causa di una più rigida struttura basata sulle colonne.

Collezioni e Review Manuale

Il database è sostanzialmente composto da due collection: quella relativa agli utenti e quella per la review manuale. La collection "utenti" è centrale nel progetto, avendo all' interno tutte le informazioni relative all'utente con la struttura descritta sopra. La collection "review manuale" invece ha all' interno tutte le corrispondenze CAP-città errate e che richiedono una correzione manuale. Per esempio, se si inserisce come città "Miano" e come CAP "ooooo", ovviamente non esiste una città che si chiama in questo modo, inoltre il CAP è nullo e quindi non valido. In questo caso non si può inserire questa tupla nel database perchè non è valida, allora si scriveranno queste informazioni sulla collection "review manuale". Sarà poi una persona fisica a modificare manualmente le informazioni rendendole valide e pronte per essere scritte nella collection "utenti".

4.2 ELABORAZIONE

Un'altra scelta importante, durante la progettazione dell' architettura, è stata la scelta di un process system che, dopo aver ricevuto in ingresso alcuni dati, li processa e infine li scrive nel database. Il process system è stato scelto dopo un accurata analisi tra le tecnologie Spark e Apache Storm e MapReduce.

4.2.1 *Soluzione scartata*

Due sono state le soluzioni implementate, una in Spark e una in Storm. Alla fine però si è dovuto prendere una scelta e scartare la meno efficiente. Nella soluzione Storm è stata creata una topologia basata su diversi bolt che controllavano e modificavano i vari campi della tupla, uno spout per ogni publisher e un bolt writer con il compito di scrivere le tuple finali nel database. Ovviamente esistono varie forme di topologie che si possono implementare, si è optato per due differenti forme di topologia. Una formata dai vari bolt, e quindi dai vari moduli di controllo, posti in parallelo, mentre l'altra con i bolt disposti in serie.

In entrambi i casi abbiamo evidenziato i tempi di processamento medio per tupla e i tempi di inserimento/aggiornamento medio nel database. Questi tempi sono nettamente peggiori rispetto alla soluzione Spark, in entrambe le versioni di Storm.

Storm sarebbe stato migliore di Spark solo in termini di flessibilità, infatti la topologia e la struttura spout/bolt lo rendono uno dei DBMS più flessibili in assoluto. Se quindi le performance tra le due soluzioni fossero state simili, la flessibilità, quindi per esempio la maggiore facilità nell'aggiunta e nella gestione di un nuovo publisher e altri aspetti di questo tipo, avrebbero assunto un ruolo fondamentale nella scelta. Quindi proprio per questo gap in performance, è stato selezionato Spark come DBMS. Un altro punto di forza di Storm come abbiamo già detto nel capitolo sull'analisi della singole tecnologie è il processamento dei dati real-time. Questo aspetto è stato preso in con-

siderazione, ma non è importantissimo ai fini del progetto, infatti non abbiamo un continuo bisogno di processare i dati in real-time. Quindi con Spark possiamo selezionare un intervallo temporale periodico più o meno grande nel quale analizzare i dati che i vari publisher hanno caricato.

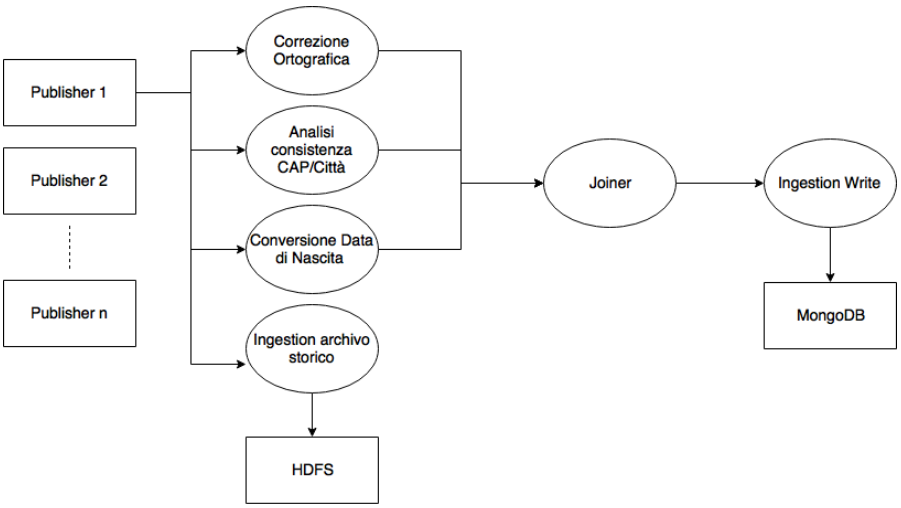


Figure 4.1: Topologia Storm con i vari moduli in parallelo

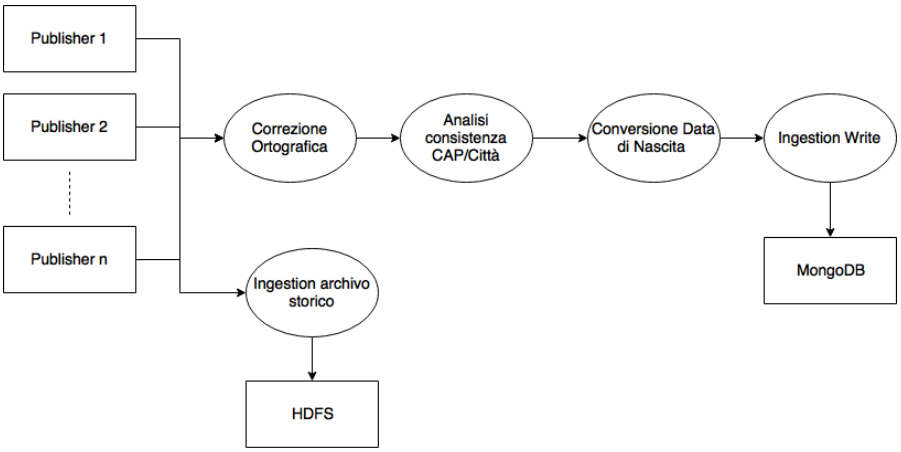


Figure 4.2: Topologia Storm con i vari moduli in serie

4.2.2 *Soluzione scelta*

Spark è un framework per l'elaborazione di dati su larga scala, molto performante, soprattutto in termini di velocità. E' caratterizzato da una notevole facilità d'utilizzo e funziona su tutti i framework più importanti, per esempio Hadoop e Mesos.

Programmazione funzionale

"To Iterate is Human, To Recuse, Divine"
L. Peter Deutsch

La programmazione funzionale è una delle principali differenze tra

Spark e Storm, infatti mentre Storm segue il paradigma tradizionale di programmazione, Spark segue un paradigma di programmazione il cui flusso di esecuzione del programma assume la forma di una serie di valutazioni di funzioni matematiche.

Un programma non è altro che una funzione, definita in termini di altre funzioni, che elaborano l'input fornito al programma e restituiscono il loro risultato finale. Le funzioni nella programmazione funzionale sono dette "funzioni pure", cioè avendo lo stesso input restituiscono lo stesso output, cioè in pratica non si hanno effetti indesiderati. Principali caratteristiche della programmazione funzionale:

- Le funzioni sono first class object, cioè sono considerate come un qualsiasi altro dato
- La ricorsione è la principale struttura di controllo, non c'è altro modo di iterare.
- Si focalizza principalmente sul processare liste.
- La valutazione delle funzioni è in modalità LAZY, cioè posso definire funzioni su liste infinite.
- Ogni variabile è read-only, cioè non può cambiare valore per evitare gli effetti collaterali.
- Si utilizzano le "high order function" cioè funzioni che lavorano su funzioni, che operano su funzioni etc...

Principali funzioni

In ingresso all' architettura si hanno i hanno dati che bisogna elaborare e restituire in uscita. I dati vengono presi in ingresso attraverso uno o più file CSV, ogni campo di ogni tupla passa allora attraverso una funzione che lo modifica correggendolo. Le funzioni principali sono:

- Correzione e analisi validità CAP/Città
- Correzione e analisi validità sesso
- Correzione e analisi validità data di nascita

Una volta applicate le funzioni sopra elencate, la tupla è pronta per essere scritta nel database integrato. Allora viene applicata un'ulteriore funzione per inserire oppure aggiornare la tupla nel database.

Correzione e analisi validità CAP/Città

Inizialmente vengono creati due dizionari indicizzati rispettivamente per CAP e per Città, il dizionario indicizzato per CAP ha come chiave il CAP e come valore un array di stringhe che rappresenta tutte le città che sono indicate da tale CAP. Viceversa, il dizionario indicizzato per città ha come chiave la città e come valore un array di tutti i CAP, questi due dizionari vengono usati per capire se un CAP appartiene a una determinata città o viceversa.

Inizialmente viene effettuato un controllo per vedere se il CAP è valido e quindi presente nel dizionario dei CAP, se il CAP è presente si passa allora a verificare che la città in input appartiene a quel determinato CAP. Se CAP e città hanno una corrispondenza allora la tupla è pronta per essere scritta nel database.
Nel caso in cui il CAP è presente nel dizionario dei CAP, mentre la città non è presente tra i valori corrispondenti a quel CAP, viene applicato l'algoritmo di Levenshtein, il quale ritorna un valore rappresentante la città che ha ortograficamente la percentuale maggiore di similitudine con la stringa data in ingresso. Ovviamente questa percentuale di similitudine deve essere maggiore di una certa soglia precedentemente stabilita. Se questo valore è maggiore della soglia allora la città selezionata dal dizionario viene scritta nel campo città della tupla in ingresso, altrimenti viene effettuato il controllo sul database per verificare se questa coppia CAP-Città è già presente o meno nella collezione "Review Manuale".
Se la coppia CAP-città è presente, allora inserisco la città corretta, altrimenti aggiungo un nuovo campo nella review manuale composto da una chiave formata dalla concatenazione del campo città con il campo CAP, e da un valore che rappresenta la correzione effettuata, il quale sarà inserito manualmente in un secondo momento.
Torniamo ora indietro e passiamo al caso un cui il CAP che abbiamo in ingresso non sia presente nel dizionario dei CAP. In questo caso la città assume un ruolo fondamentale, infatti si controlla se la città è presente nel secondo dizionario, quello delle città, se è presente viene preso il primo CAP corrispondente a quella determinata città. Se invece, anche la città non è presente nel dizionario delle città allora controllo se nella collezione "Review Manuale" la chiave formata dalla concatenazione di CAP-Città è presente, se è presente prendo il valore corrispondente, altrimenti viene inserita una nuova riga nella review manuale.

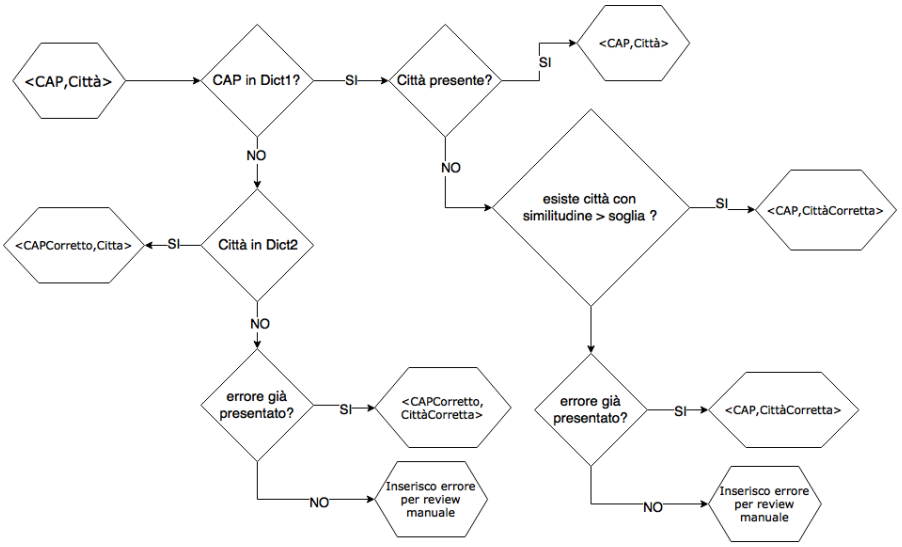


Figure 4.3: Modulo Correzione Città e CAP, Dict1: dizionario indicizzato su CAP Dict2: dizionario indicizzato su Città

Distanza di Levenshtein

Per implementare la distanza tra due stringhe, come ad esempio tra "Milano" e "Miano" è stata utilizzata la distanza di Levenshtein. La distanza di Levenshtein è una misura per la differenza tra due stringhe e serve per determinare quanto siano simili due stringhe. Viene utilizzata negli algoritmi di controllo ortografico e per fare ricerca di similarità tra immagini,suoni,testi etc... La distanza tra due stringhe A e B è definita come il numero minimo di modifiche elementari che consentono di trasformare la stringa A nella stringa B. Una modifica può essere la cancellazione di un carattere, la sostituzione di un carattere con un altro oppure l'inserimento di un carattere.
Esempio: per trasformare la stringa bar in biro ci vogliono due modifiche:

a) sostituzione di "a" con la "i"

b) inserimento di "o"

La distanza minima tra le due parole è di 2.

Correzione e analisi validità sesso

La funzione riguardante il controllo e l'analisi della validità del campo sesso, prende in ingresso il valore del campo sesso della tupla corrente. Si procede quindi, per prima cosa, a verificare se la stringa in input inizia con il carattere "m"/"M", se inizia per questo carattere allora si assume che il valore inteso sia maschio, altrimenti si verifica se la stringa inizia con il carattere "f"/"F". Se la stringa inizia con "f" allora si assume che il valore sia femmina, altrimenti si applica l'algoritmo di Levenshtein calcolando la distanza della stringa in ingresso da un array di stringhe rapprensentanti il sesso maschile e parallelamente la distanza da un array di stringhe rappresentanti il sesso femminile. Ottenute queste distanze sottoforma di percentuali di similitudini, viene presa la maggiore di ognuno dei due gruppi e inizio il confronto. Se la percentuale maggiore del gruppo maschile è maggiore di quella maggiore del gruppo femminile allora si assume maschio come valore giusto, altrimenti assumo femmina come valore giusto. Nel caso in cui le due percentuali fossero perfettamente uguali, allora viene preso un valore casuale e viene inserito quest'ultimo nel campo sesso.

Correzione e analisi validità data di nascita

La data di nascita viene fornita in ingresso divisa in tre campi, giorno del mese, mese dell'anno e anno. In uscita pero è stato deciso di unificare i tre campi sotto un unico campo "data di nascita", i controlli che andiamo a fare sui vari campi sono principalmente per verificare che i valori rientrino in dei range accettabili e logicamente validi. Le verifiche che si eseguono sono:

- Il giorno di nascita deve avere un valore compreso tra 1 e il numero massimo che rappresenta il numero di giorni che un

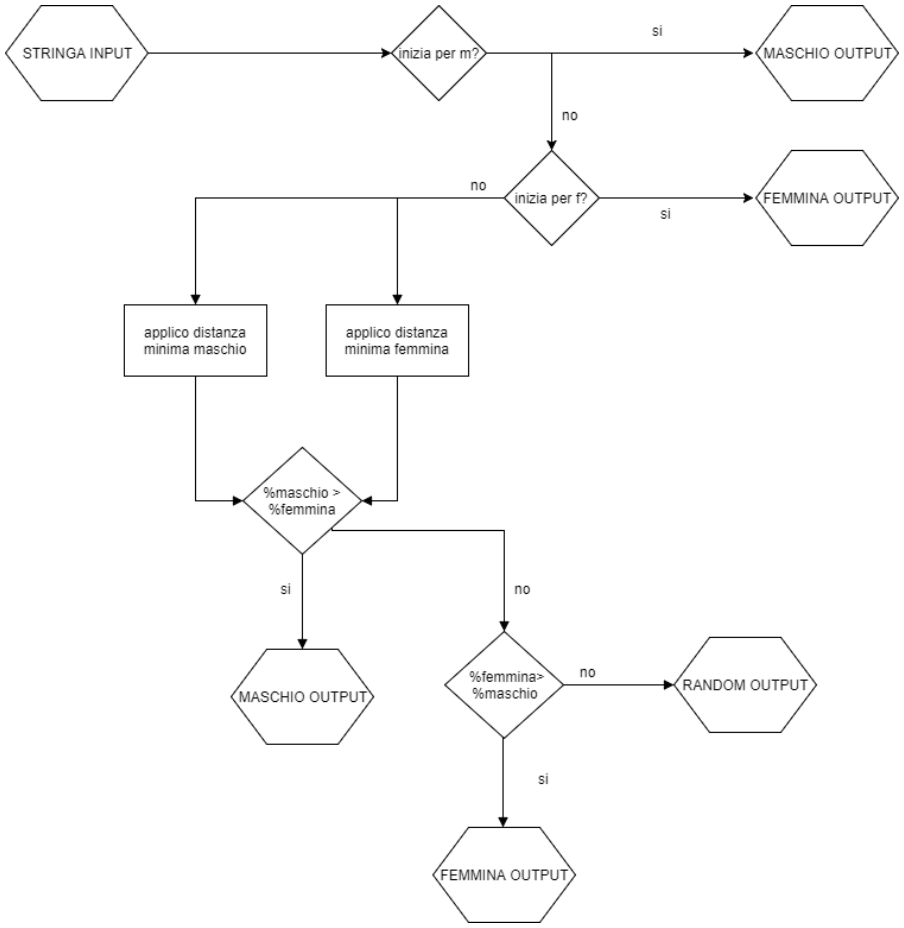


Figure 4.4: Schema correzione sesso

determinato mese ha, considerando anche il caso di febbraio il cui numero massimo di giorni varia a seconda del anno bisestile o non bisestile.

- Il mese di nascita deve avere un valore compreso tra 1 e 12.
- L'anno deve avere un valore logicamente valido, ovvero un individuo non può avere un anno di nascita maggiore dell'anno corrente, e inoltre un individuo non dovrebbe avere più di 150 anni di età. Quindi viene stabilito un range di valori che rispecchi la realtà.

Inserimento e aggiornamento

Una volta applicate le funzioni di correzione e analisi di validità dei vari campi, la tupla è pronta per essere scritta nel database. Il processo di inserimento oppure di aggiornamento viene eseguito attraverso tre funzioni, la prima funzione verifica se l'hash della tupla da inserire è già presente nel database, nel caso in cui è presente allora viene chiamata la funzione relativa all'aggiornamento, altrimenti viene chiamata la funzione relativa all'inserimento di un nuovo documento. L'inserimento di un nuovo documento nel database è sicuramente l'operazione più facile e veloce, infatti viene inserito un documento con l' hash della tupla come identificativo e come attributi i

vari campi. Ad ogni attributo viene aggiunto la data corrente come data ultimo inserimento, mentre il campo frequenza viene settato semplicemente a 1. L' aggiormento di un documento già esistente nel database richiede qualche passaggio in più. Inizialmente per ogni attributo verifichiamo se il valore della tupla è già presente nel database, se è presente viene incrementato il campo frequenza di uno e modificata la data ultimo inserimento con la data attuale. Se invece il valore non è presente nell' array dei valori, viene inserito il nuovo valore con frequenza 1 e con la data ultimo inserimento uguale alla data corrente.

4.3 RISULTATI SPERIMENTALI

Una volta implementate le varie soluzioni sono stati eseguiti dei test per verificare le performance, in particolare viene preso in considerazione il tempo di processamento, inserimento e modifica nel database. Dal punto di vista di configurazione hardware è stato utilizzato un singolo nodo con processore Intel i7 2.5 Ghz, RAM 16GB, SSD 512GB. Come funzione di costo è stata utilizzata la quantità di tuple in input, i test sono stati effettuati su due campioni di 27 mila e 90 mila tuple. I risultati nettamente migliori sono stati quelli della soluzione Spark, che impiega 4,25 minuti per processare 27 mila tuple, in confronto ai 6,28 minuti della soluzione Storm. Il tempo di processamento di 90 mila tuple è stato 22,57 minuti per quanto riguarda Spark, in confronto ai 32,5 minuti della soluzione Storm.

Table 4.1: Tempi processamento Spark e Storm

Numero di tuple	Spark	Storm
27000	4.25 min	6.28 min
90000	22.57 min	32.5 min

Il grafico in figura 4.1 mostra il tempo di inserimento e modifica nel database delle singole tuple. Si può osservare che le tuple che vengono inserite come nuovo dato nel database ci mettono decisamente meno tempo rispetto alle tuple che portano alla modifica di un dato esistente. Infatti gran parte delle tuple che vengono inserite impiegano un tempo inferiore a 30 ms, mentre le altre sono equamente distribuite in un intervallo temporale compreso tra 1 e i 200 ms.

Table 4.2: Tempi di processing medi

Modulo	Tempo medio per tupla
Controllo Data di Nascita	0.00607 ms
Controllo CAP/Città	0.01504 ms
MongoDB Modifica	12.54167 ms
MongoDB Inserimento	0.41406 ms
Processing completo	11.18 ms

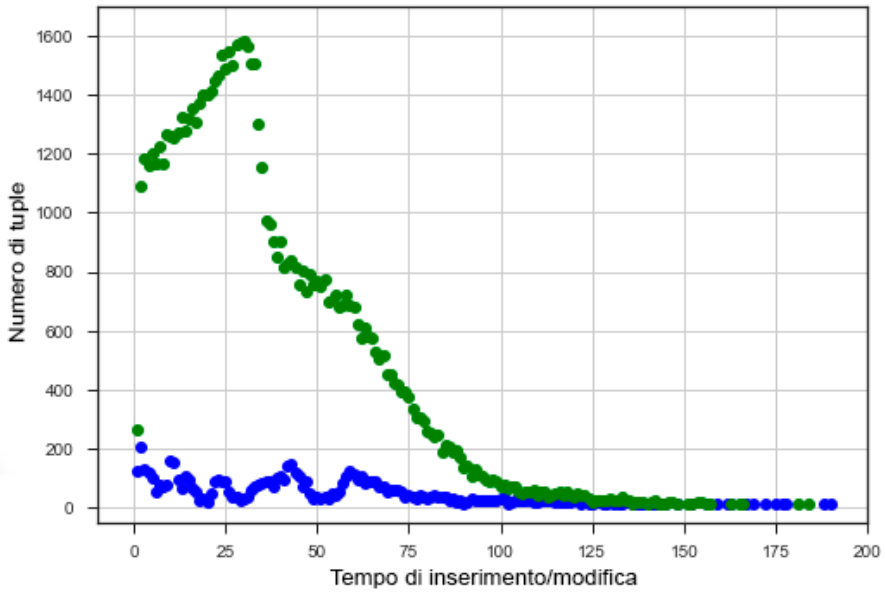


Figure 4.5: Grafico performance Storm, in blu le modifiche, in verde gli inserimenti nel database

5 CONCLUSIONE

Il progetto è incentrato principalmente nella scelta e nell' implementazione di due elementi: il Processing System e il DBMS. Il processing system che maggiormente rispecchia i requisiti del progetto è stato Spark poiché durante i passaggi intermedi di elaborazione Spark non scrive su disco, ed è quindi molto più veloce dei concorrenti. Mentre il database ritenuto migliore è stato MongoDB. La scelta è stata guidata dall' enorme flessibilità che MongoDB possiede, grazie alla struttura basata sul documento. La struttura proposta in Spark è capace di leggere da file CSV, e applicare funzioni a ogni singola tupla letta. Le funzioni applicate sono l'analisi CAP/Città, l'analisi del sesso, l'analisi della data di nascita. A questo punto la tupla è stata processata, quindi passa attraverso un'ultima funzione che ha il compito di scriverla nel database.

5.1 SVILUPPI FUTURI

Può essere interessante valutare un metodo differente per la similarità tra stringhe, nello specifico nel caso della correzione della città. Se spostiamo il punto di vista a livello aziendale, per ampliare il business, in futuro si può pensare di ampliare il database a tutta europa. Ciò porterebbe a una rivisitazione dell'architettura, in particolare il modulo CAP/città e il database dovranno essere adeguati alla nuova struttura.

Grazie alla scelta effettuata in fase di progettazione di scegliere MongoDB è possibile mantenere inalterata la struttura dei vecchi record. Per migliorare la segmentazione i publisher potrebbero aggiungere l' attributo "area di interesse" in modo tale da inviare email ad utenti maggiormente interessati. In futuro, inoltre, si potrebbero effettuare dei test con delle configurazioni hardware più performanti per valutarne meglio la scalablità.

L'ampliamento del cluster utilizzato e quindi una migliore distribuzione del carico di lavoro sui vari nodi potrebbero essere aspetti che in futuro andrebbero considerati.

BIBLIOGRAPHY

[1] Monica Mordonini. (). Cenni sulla gestione delle transazioni in dbms, [Online]. Available: http://www.dmi.unict.it/~ggiuffrida/db/Slides/DB-Sql%20Transazioni.pdf (visited on 11/28/2017) (cit. on p. 5).

[2] Luca Vetti Tagliati. (). Il teorema cap, [Online]. Available: http://www.mokabyte.it/2013/07/BrewerCAP-1 (visited on 11/28/2017) (cit. on p. 6).

[3] Amol Nayak, *Mongodb cookbook*. Packt Publishing Ltd, 2014 (cit. on p. 8).

[4] MongoDB Website. (). Utilizzatori di mongodb, [Online]. Available: https://www.mongodb.com/who-uses-mongodb (visited on 11/28/2017) (cit. on p. 8).

[5] ——, (). Costi mongodb, [Online]. Available: https://www.mongodb.com/collateral/total-cost-ownership-comparison-mongodb-oracle (visited on 11/28/2017) (cit. on p. 9).

[6] Redis Website. (). Redis intro, [Online]. Available: https://redis.io (visited on 11/28/2017) (cit. on p. 9).

[7] Redis website. (). Redis utilizzatori, [Online]. Available: https://redis.io/topics/whos-using-redis (visited on 11/28/2017) (cit. on p. 10).

[8] HBase website. (). Caratteristiche hbase, [Online]. Available: https://hbase.apache.org (visited on 11/28/2017) (cit. on p. 10).

[9] ——, (). Linguaggi supportati hbase, [Online]. Available: https://quabase.sei.cmu.edu/mediawiki/index.php/HBase_Query_Language_Features (visited on 11/28/2017) (cit. on p. 11).

[10] ——, (). Use case hbase, [Online]. Available: https://hbase.apache.org/poweredbyhbase.html (visited on 11/28/2017) (cit. on p. 11).

[11] html.it Website. (). Struttura cassandra, [Online]. Available: http://www.html.it/articoli/apache-cassandra-columnfamily-keyspace-e-le-similitudini-con-sql (visited on 11/28/2017) (cit. on p. 11).

[12] (). Cassandra support for programming languages, [Online]. Available: https://www.packtpub.com/mapt/book/big_data_

and_business_intelligence/9781782162681/9/ch09lvl1sec57/support-for-programming-languages (visited on 11/28/2017) (cit. on p. 12).

[13] (). Utilizzatori cassandra, [Online]. Available: https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781782162681/9/ch09lvl1sec57/support-for-programming-languages (visited on 11/28/2017) (cit. on p. 12).

[14] Mark Grover, Ted Malaska, Jonathan Seidman, and Gwen Shapira, *Hadoop application architectures: Designing real-world big data applications*. " O'Reilly Media, Inc.", 2015 (cit. on p. 14).

[15] Juwei Shi, Yunjie Qiu, Umar Farooq Minhas, Limei Jiao, Chen Wang, Berthold Reinwald, and Fatma Özcan, "Clash of the titans: Mapreduce vs. spark for large scale data analytics," *Proceedings of the VLDB Endowment*, vol. 8, no. 13, pp. 2110–2121, 2015 (cit. on p. 16).

[16] J Scott, *Getting started with apache spark*. 2015 (cit. on p. 17).

[17] Ankit Jain and Anand Nalya, *Learning storm*. Packt Publishing, 2014 (cit. on p. 18).

[18] (). Benefici direct email marketing, [Online]. Available: https://matico.io/direct-email-marketing (visited on 11/28/2017) (cit. on p. 23).

[19] (). Definizione di completezza, [Online]. Available: http://www.crisp-org.it/public/uploads/2014/12/Data_quality_Arifl-Crisp.pdf (visited on 11/28/2017) (cit. on p. 25).

[20] Martin Fowler. (). Datalake, [Online]. Available: https://martinfowler.com/bliki/DataLake.html (visited on 11/28/2017) (cit. on p. 29).