ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

SCUOLA DI SCIENZE
Corso di Laurea in Ingegneria e Scienze Informatiche

Progettazione e prototipazione di un sistema di Social Business Intelligence con Hadoop Impala

Relazione finale in
LABORATORIO DI BASI DI DATI

Relatore:
Prof. Matteo Golfarelli
Correlatore:
Dott. Enrico Gallinucci

Presentata da:
Antony Chiossi

III sessione di laurea
Anno Accademico 2014 - 2015

Ringraziamenti

Dedico questa tesi al mio medico e amico Cesar.

Indice

Elenco delle figure **5**

1 Perché Big data **11**
1.1 Principali aspetti dei Big Data 11
1.2 Acquisizione dei dati 13
1.3 Gestione e memorizzazione dei dati 13
1.4 Analisi e processamento dei dati 14

2 Introduzione al framework Hadoop **15**
2.1 Caratteristiche di Hadoop 15
2.2 HDFS: Hadoop Distributed Filesystem 17
2.2.1 Architettura 17
2.2.2 Replicazione dei dati 19
2.3 Hive 20
2.3.1 Architettura Hive 21
2.4 Parquet 22
2.4.1 Il modello 24
2.4.2 Il formato a colonne (Columnar format) 26

3 Impala: introduzione **28**
3.1 Vantaggi di Impala 29
3.2 Progettazione fisica dello schema 29
3.3 Impala Query Language 30
3.4 Architettura 30
3.4.1 Distribuzione dello stato 32
3.4.2 Catalog Service 33
3.4.3 Frontend 33
3.5 File formats 36

3.6 Best practice per l'esecuzione di query 38
3.6.1 Formato file Parquet 38
3.6.2 Partitioning 39
3.6.3 Considerazioni sulle prestazioni (JOIN) 40
3.6.4 HDFS Caching 41
3.6.5 Explain 41

4 Caso di studio sulla politica europea **44**
4.1 Social Business Intelligence 44
4.2 Architettura processo di SBI 46
4.3 WebPolEU 47
4.4 Progettazione della base di dati 49
4.5 Porting su Impala 54

5 Test delle performance di Impala **57**
5.1 Architettura Hardware 57
5.2 Query OLAP 58
5.3 Confronto con Oracle 62
5.4 Conclusioni 70

Bibliografia **72**

Elenco delle figure

2.1 Architettura HDFS - comprese operazioni di lettura, scrittura e resplicazione **[6]** 19
2.2 Principali componenti di hive e la loro interazione con Hadoop 21
2.3 Esempio grafico di un columnar storage **[8]** 23
2.4 Dati salvati in formato row-oriented **[8]** 23
2.5 Dati salvati in formato column-oriented **[8]** 23
2.6 Esempio rappresentazione liste **[8]** 25
2.7 Esempio rappresentazione mappa **[8]** 25
2.8 Elenco telefonico rappresentato come albero **[8]** 26
2.9 Metodo di rappresentazione dei dati in formato colonnare **[8]** 27

3.1 Singola istanza di un demone impalad con le sue principali interazione con gli altri denomi/processi **[12]** 31
3.2 Esempio di flusso di esecuzione distribuito 34
3.3 Esempio di costruzione di un piano di esecuzione in due fasi **[13]** 36
3.4 Piano di esecuzione della query COOCC con explain_level = 1 42

4.1 Architettura di un generico processo di Social Business Intelligence . . . 45
4.2 Principali dati presi in esame con rispettive cardinalità 46
4.3 Una rappresentazione UML dell'ontologia Topic per il progetto WebPolEU 48
4.4 porzione semplificata dello schema logico dell'ODS 49
4.5 orzione semplificata dello schema logico del Data mart 50
4.6 Schema logico dell'ODS relativo alla memorizzazione delle relazioni fra le entity all'interno delle clip 51
4.7 Schemi DFM dove sono rappresentate le gerarchie di Topic e Clip 52
4.8 la figura mostra i due principali fatti di interesse del sistema 52
4.9 Implementazione dei due fatti principali sul data mart 53
4.10 Implementazione dei Topic sul data mart 54

5.1 Output (SELECT_1) 58
5.2 Output (parte 1, query PRED_1) 59
5.3 Output (parte 2, query PRED_4) 59
5.4 Output (AGGREG1) 60
5.5 Output (N_JOIN5) 60
5.6 Output (N_COL1) 61
5.7 Output (COOCC) 61
5.8 Porzione tabella con i tempi relativi ai primi due gruppi di query 62
5.9 Porzione tabella con i tempi relativi agli ultimi 4 gruppi di query 63
5.10 Grafico con i tempi dei primi due gruppi di query della comparazione fra 7 demoni - 1 demone - Oracle 64
5.11 Grafico con i tempi gli ultimi 4 gruppi di query della comparazione fra 7 demoni - 1 demone - Oracle 65
5.12 Comparazione di alcune query cambiando la strategia di join - solo con configurazione Impala (7 demoni) 66
5.13 Tabella con i tempi (secondi) della comparazione fra Impala (7 demoni) - Impala Filtro (Unclassified) - ElasticSearch 68
5.14 Risultati del primi due gruppi di query della tabella 5.13 69
5.15 Risultati degli ultimi tre gruppi di query della tabella 5.13 69

Abstract

Il presente elaborato ha come oggetto l'analisi delle prestazioni e il porting di un sistema di SBI sulla distribuzione Hadoop di Cloudera. Nello specifico è stato fatto un porting dei dati del progetto WebPolEU. Successivamente si sono confrontate le prestazioni del query engine Impala con quelle di ElasticSearch che, diversamente da Oracle, sfrutta la stessa componente hardware (cluster).

Introduzione

Dall' avvento del cosiddetto web 2.0, i dati immessi nella rete attraverso i vari servizi web, sono in costante aumento. Di conseguenza la gestione e gli strumenti per analizzare questi ultimi hanno subito una grande innovazione. Sin dagli albori si è capito l'importanza che possono avere i dati, in quanto questi possono essere usati per innumerevoli tipi di analisi, consentendoci di capire meglio il mondo che ci circonda. Agli inizi, questo settore offriva poche e deboli tecnologie che consentivano solamente analisi statiche e mera estrazione dei dati raccolti. In tempi più recenti con l'avvento dei database relazionali e del linguaggio SQL, l'analisi dei dati diventa più dinamica grazie all'incremento delle operazioni eseguibili sui dati. Infatti SQL consente una estrazione dei dati semplice a qualsiasi livello di dettaglio. Le principali attività di analisi vengono effettuate su basi di dati operazionali, comunemente dette di tipo OLTP (On Line Transaction Processing), ottimizzati per attività di tipo transazionale (inserimento, modifica e cancellazione dei dati) anziche per attività di analisi e lettura massiccia di record. Con i sistemi OLTP risulta difficile effettuare una storicizzazione dei dati anche in presenza di dati storici in quanto risulta molto complesso ricostruire la situazione dei dati nel passato **[1]** . Oltre ciò, è sempre più in crescita il numero di contesti in cui vi sono numerose applicazioni, non condividenti la stessa sorgente, che manipolano dati replicati, non garantendo così coerenza e uniformità dei dati presi in oggetto. La necessità di effettuare analisi dei dati in modo semplici direttamente sulle fonti operazionali ha portato allo sviluppo di database progettati appositamente per l'analisi dei dati. Nascono così, agli inizi degli anni novanta, i primi Data Warehouse, database a tutti gli effetti (dati integri, consistenti e certificati) relativi a processi di businnes delle aziende. Grazie a questi ultimi vedono la luce le prime attività analitiche dei sistemi di Businnes Intelligence (BI). Per Businness Intelligence si intende un inseme di modelli, processi, metodi,persone e strumenti che rendono fattibile la raccolta e la riorganizzazione dei dati prodotti da un'azienda. Attraverso varie funzionalità, come analisi e aggregazioni, permette la conservazione, la trasformazione in informazioni, la reperibilità e la presentazione in forma semplice e flessibile, fornendo così uno strumento ideale per il supporto per i processi decisionali. Tuttavia consente solamente una valutazione a consuntivo di ciò che è accaduto nel passato, o al meglio, di ciò che sta accandendo nel presente. Circa un

decennio fa, sono emerse le prime necessità di effettuare analisi previsionali per anticipare gli eventi, ottenendo così notevoli fantaggi in termini di businnes. Nascono così, nei primi anni duemila, le tecniche di Data Mining che consentono di esplorare i dati e estrarre informazioni utili per cercare pattern, relazioni non note a priori o non immediamente indentificabili. Negli ultimi cinque anni, l'evoluzione dell'analisi dei dati e la BI hanno portato alla nascita di varie specializzazioni in questo campo:

- Business Analytics: insieme di tecnologie, practices, applicazionie e metodi che sfruttano modelli matematici e statistici per l'analisi dei dati. Sono spesso dotati di funzionalità di analisi visuale.
- Mobile BI e Reporting: si occupa della progettazone e produzione di applicazioni per l'esplorazione dei dati e la visualizzazione dei report su supporti mobili.
- Self-Service BI: consiste in software progettati per costruire in modo semplice report, analisi e nuovi modelli dati.
- Cloud Computing: complesso di tecnologie che permettono di usufruire di risorse hardware e software attraverso internet. Queste risorse sono spesso usate per analisi dati e nella BI.
- Big Data: insieme di fattori evolutivi volti all'analisi di grandi quantità di dati.

Nel primo capitolo viene fatta una panoramica sui Big Data, descrivendone le loro caratteristiche. Vengono poi descritte le tecnologie attualmente utilizzate per la loro memorizzazione, soffermandosi sulla più utilizzata, Hadoop, che offre un filesystem distribuito e un gestore di risorse attraverso i quali più applicazioni possono eseguire contemporaneamente computazioni sui dati.
Vengono in seguito descritte l'architettura e le caratteristiche principali di Hadoop per poi descrivere HDFS Hadoop Distributed Filesystem) e il modo in cui si effettuano letture e scritture su di esso. Di seguito vengono introdotti Hive e Parquet, rispettivamente un software di data warehouse per Hadoop e un formato di file colonnare. Entrambi hanno un ruolo chiave per la tecnologia introdotta nel capitolo tre.
Nel terzo capito verrà appunto presentato Impala un MPP (*Massively Parallel Processing*) SQL query engine. Di questo verranno elencati i vantaggi che offre, tra cui la facilità di utilizzo grazie al suo linguaggio SQL-like e la semplicità con cui si è in grado di fare un porting da un sistema relazionale a Impala. Verrà poi descritta la sua architettura e le best practice da considerare per sfruttare al massimo questa tecnologia.
Nel quarto capitolo verrà fatta una breve introduzione alla Social Business Intelligence per poi passare alla presentazione del progetto e caso di studio WebPolEU. Di seguito verranno affrontate la progettazione della relativa base di dati adatta ad analisi OLAP e il suo

porting su Impala.
Nell'ultimo capitolo verranno esposti i risultati del test delle performance di Impala. In particolare si è confrontata la tecnologia oggetto di questa tesi con un tradizionale RDBMS Oracle e una similare tecnolgia, ElasticSearch.

Capitolo 1

Perché Big data

Oggi giorno le aziende si trovano a fare i conti con una mole di dati in costante crescita soprattutto grazie ai social media che mettono in diretto contatto il produttore con il consumatore. Hanno quindi la necessità di analizzare e immagazzinare una grande mole di dati, ma ha causa della mancanza si strumenti adatti per l'analisi e l'elaborazione non sono in grado di soddisfare questo loro bisogno. Sebbene le aziende abbiamo la possibilità di accedere a questa enorme mole di dati, con i tradizionali strumenti come per esempio i classici database relazionali, non sono nelle condizioni di ricavare alcuna informazione utile da questi ultimi, perché spesso sono "grezzi" quindi difficilmente analizzabili. La porzione di dati processabile sta calando molto rapidamente, non cosentendo così il completo accesso ai dati di cui le aziende hanno bisogno per ottimizzare il loro businnes. Nell'ultimo decennio, grazie alla possibilità che ogni individuo ha di produrre contenuti, si sono resi disponibili dati in così grande quantità da generare il fenomeno conosciuto come Big Data.

1.1 Principali aspetti dei Big Data

Come già accennato precedentemente, i big data raffigurano quei dati con un volume abbastanza consistente, che possono presentarsi in formato strutturato o destrutturato e hanno un crescita in termini di volume molto rapida. Possiamo identificare in volume, varietà, velocità e veridicità, quattro aspetti cruciali che caratterizzano i big data **[2]** .

- **Volume**: Quando si parla di un grande volume di dati si fa rifermento a realtà come Twitter o Facebook, che sono in grado di generare quasi una decina di terabyte (TB) di dati ogni giorno. Se si pensa che tutto ciò che riguarda noi essere umani, sta velocemente subendo un processo di informatizzazione, si capisce bene che questo

andamento è tutt'altro che in calo. L'aspetto primario per operare con i big data è la memorizzazione dei dati "grezzi" per un futuro utilizzo. Ogni operazione di pulizia o scarto potrebbe portare all'eliminazione di informazioni utili successivamente. Questa politica contribuisce in modo cospicuo alla velocità con cui il volume di dati immagazzinati cresce. Durante la fase decisionale, su quale sistema adottare per memorizzare i dati una azienda deve soprattutto tener conto di cifre da investire per lo storage e la capacità di calcolo. Questi costi, se si sceglie di utilizzare normali RDBMS, potrebbero risultare non sufficenti a causa dell'elevata rapidità con cui il volume di dati cresce. Si sono trovate quindi, soluzioni e tecnologie alternative che permettono di immagazzinare, gestire e analizzare al meglio l'enorme mole di dati fornendo supporto al business. La tecnologia più diffusa è Apache Hadoop, grazie alla sua ottima capacità di processare grandi quantità di dati con costi relativamente bassi.

- **Varietà**: Con la crescita del fenomeno IoT (Internet of Things), la creazione di contenuti web da parti di chiunque e soprattutto i vari social media, i dati assumono una struttura sempre più eterogenea e non predefinita, rendendo complicata l'analisi e la loro rappresentazione in un database relazionale. Per la memorizzazione di grandi quantità di dati semistrutturati, quasi sempre si opta per database NoSQL, i quali sono adatti per organizzare questo tipo di dati non imponendo loro uno schema predefinito. Questa loro caratteristica di essere schemaless consente loro una maggiore adattabilità verso i dati. Nella situazione attuale in cui ci troviamo, possiamo notare come ben l'80% dei dati prodotti siano destrutturati e se possibile semistrutturati; le imprese devono quindi adattarsi per poter sfruttare a pieno le oppurtunità fornite dai big data.

- **Velocità**: Un'altra caratteristica che caratterizza i big data è la velocità con cui i dati vengono generati. I nuovi sistemi per la gestione di grandi quantità di dati devono essere in grado di tenere il passo con la velocità con la quale questi aumentano di volume. Qui la difficoltà, per le aziende, sta nel riusciure a sfruttare i dati con altrettanta rapidità, pulendo i dati in tempi brevi per poi utilizzarli per il business. Sopprattutto oggi, dove tutto si muove con velocità considerevoli, le decisioni se prese tempestivamente possono fare la differenza; per questo una azienda deve essere in grado di analizzare i dati prodotti in tempi più che brevi.

- **Veridicità**: Tutti questi dati raccolti costituiscono un valore per una azienda. E' dall'analisi dei dati che si colgono le opportunità e si trae supporto per i processi decisionali in modo tale che questi possano avere un grande impatto sull' attività, e più dati si hanno a disposizione più informazioni e valore si riescono ad estrarre.

Tuttavia il solo volume dei dati non garantisce sufficientemente la "qualità" dei dati: La veridicità e la qualità dei dati diventa pertanto un requisito fondamentale affinché i dati possano davvero "alimentare" nuove intuizioni e idee e costituire valore.

1.2 Acquisizione dei dati

Contrariamente ai sistemi sistemi tradizionali le sorgenti da cui provengono i dati sono diverse. Alcune delle fonti più comuni sono, per esempio, i social network, i sensori di controllo o qualsiasi altro dispositivo elettronico che genera informazioni; per ricavare i dati da tutte queste fonti si utilizzano,ovviamente, tecniche diverse **[1]** . L'aquisizione dei big data può essere essenzialmente effettuata tramite in quattro modi, tramite:

- API (Application Programming Interface): sono protolli usati come interfaccia di comunicazione tra componenti software. Alcuni esempi di API sono la Twitter API, la Graph Facebook API e le API offrite da alcuni motori di ricerca come Google, Bing e Yahoo!. Esse permettono ad esempio di ottenere i tweet relativi a determinati argomenti o di esaminare i contenuti pubblicitati che rispondono a determinaticriteri di ricerca nel caso della Graph API di Facebook.

- Strumenti di ETL (Extract, Transform, Load): sono strumenti utilizzati in Data Warehousing per convertire i database da un formato o tipo ad un altro oppure per eseguire una pulizia dei dati.

- Software di Web Scraping: possono prelevare dati semplicemente analizzando il Web, cioè la rete di pagine collegate tra loro tramite hyperlinks. Software come Apache Tika automatizzano il processo di lettura di dati e metadati contenuti in file HTML, XML, PDF, EPUB, file office, audio,immagini, JAR e così via.

- Lettura di stream di dati: esistono ecnologie per la cattura e il traferimento di dati in tempo reale. Le tecnologie più diffuse sono Apache Flume e Microsoft StreamInsight.

1.3 Gestione e memorizzazione dei dati

Con l'avvento di questa sempre più consistente mole di dati destrutturati o semistrutturati, è emerso il bisogno di database sempre più flessibili e scalabili. Per questo motivo sono stati creati nuovi sistemi che permettono di memorizzare tipi di dati non relazionale offrendo scalabilità orizzontale, cioè le prestazioni aumentano in maniera lineare rispetto al

numero di nodi presenti. La tecnologia che spicca, per semplicità e diffusione, in questo nuovo settore è Hadoop, una libreria software (framework) open-source scalabile e affidabile per il calcolo distribuito. Questo tipo di software struttano, come si può evincere dallo scopo per il quale sono stati creati, la capacità computazionale di macchine distribuite suddividendo tra loro il carico di lavoro; la capacità di calcolo complessiva risulta essere composta dalla somma della capacità di ogni singolo nodo del cluster, consendendo così di eleborare molte più informazioni. Daro l'esecuzione di una qualsiasi operazione necessità di dati, è fondamentale che ogni nodo abbia accesso ad essi. Entra così in gioco il file system distribuito di Hadoop, che diversamente dai tradizionali file system, consente la memorizzare di file di grandi dimensioni (Terabytes/Petabytes) su macchine distribuite. HDFS sfrutta la replicazione dei dati per ottenere una forte tolleranza ai guasti e una bassa latenza di accesso ai dati.

1.4 Analisi e processamento dei dati

Le quattro caratteristiche che contraddistinguono in Big Data hanno modificato il modo con cui fare analisi sui dati. Come già detto in precedenza le fonti il formato e la quantità che caratterizzano questo tipo di dati porta con sé la necessità di avere strumenti adeguati per poterli gestire e analizzare in modo efficiente e adeguato. In origine e ancora tutt'oggi, si utilizzavano strumenti come SQL, OLAP e Excel per analizzare e addirittura esplorare i dati. Negli ultimi tempi a causa dell'aumento della competitività e degli aspetti che caratterizzano i Big Data, sono nate nuove tecnologie e tecniche di analisi predittiva e di monitoraggio in tempo reale. Questi strumenti sono in grado di operare con dati strutturati e non ad alta velocità come veri e propri strumenti business analitycs. Sempre più spesso infatti si parla di Big Data Analitycs: processo in grado di esaminare grandi quantità di dati di vario tipo, con l'intento di trovare patterns nascosti, andamenti di mercato, correlazioni, preferenze dei clienti e altre informazioni utili. Secondo la piattaforma adottata per la gestione dei big data e cosa si vuole analizzare è opportuno scegliere lo strumento che meglio si addice a tale scopo. Le principali tecnologie più diffuse per la big data analitycs sono: Drill, ElasticSearch, Hadoop MapReduce, Hive, Impala, Mahout, Pig, Presto e R. Alcuni di questi strumenti come per esempio Spark, Mahout e R mettono a disposizione algoritmi parallelizzabili ed avanzati di machine learning, text-mining, analisi di grafi e statistica avanzata. Imapala mette a disposizione algoritmi di aggregazione scalabili e distribuiti che consentono di interrogare, in modo parallelo, i dati (salvati su HDFS) dislocati su migliaia di macchine.

Capitolo 2

Introduzione al framework Hadoop

Hadoop è uno dei framework open source progettato per operare in modo distribuito su grandi quantità di dati e, al momento, è uno dei più diffusi grazie alla sua alta affidabilità e scalabilità. Prima dell'avvento di Hadoop le elaborazioni su grandi quantità di dati erano realizzate da sistemi apposti ad alte prestazioni (High Performance Computing e Grid Computing). Hadoop, al contrario dei sistemi che lo hanno preceduto, mette a disposizione librerie ad alto livello facili da utilizzare; inoltre è in grado di minimizzare i tempi di latenza di rete, quindi migliorare i tempi di accesso ai file, utilizzando un meccanismo di replicazione dei dati sui nodi. Attualmente Hadoop è alla versione 2 la quale, rispetto alla prima versione, consente ad applicazioni diverse da MapReduce di sfruttare le potenzialità del framework **[3]**; **[4]**; **[5]** .

2.1 Caratteristiche di Hadoop

Il framework Hadoop include i seguenti moduli:

- Hadoop Common: è un package che fornisce accesso al file system supportato da Hadoop. Questo package contiene i file jar e gli script necessari per avviare Hadoop. Il package fornisce inoltre il codice sorgente, la documentazione e una sezione contributi che include i progetti della comunità Hadoop.

- Hadoop Distributed File System (HDFS): è il filesystem distribuito di Hadoop che è stato specificatamente progettato per memorizzare grandi quantità di dati su commodity hardware. Ogni file è diviso in parti consistenti e memorizzato su diversi nodi, chiamati Data Node, come file locali. In aggiunta il meccanismo della replicazione dei dati permette l'accesso parallelo e distribuito ai dati durante situazioni di

alto carico. A differenza di altri filesystem di questo tipo, HDFS rende disponibile tramite API la locazione dei blocchi, permettendo ai client di accedere direttamente ai datanode per la lettura.

- Hadoop YARN: Acronimo di Yet-Another-Resource-Negotiator, YARN è un framework che consente di creare applicazioni o infrastrutture di calcolo distribuito di qualsiasi tipo. YARN si prende carico della gestione delle risorse quali la memoria e la CPU, e monitora l'esecuzione delle applicazioni. Rispetto alla prima versione di MapReduce è ora un sistema molto più generico, infatti la seconda versione di MapReduce è stata riscritta come applicazione YARN, con alcune sostanziali differenze. Le due funzioni svolte dal JobTracker sono state suddivise in altrettante componenti differenti:

  - ResourceManager si occupa dell'assegnazione delle risorse alle applicazioni;
  - ApplicationMaster gestisce la singola applicazione, facendola partire, monitorandola e determinandone il riavvio in caso di errore;

  Inoltre, per ciascun nodo troviamo un NodeManager che ha il compito di gestire i processi. Riassumendo, un'applicazione MapReduce è un job MapReduce eseguito da un ApplicationManager e il framework MapReduce è quindi solo uno dei possibili framework di calcolo sviluppabili attraverso YARN.

- Hadoop MapReduce: MapReduce è un framework per la creazione di applicazioni in grado di elaborare grandi quantità di dati in parallelo basandosi sul concetto di functional programming. A differenza della programmazione multithreading, in cui i thread condividono i dati oggetto delle elaborazioni presentando così una certa complessità proprio nel coordinare l'accesso alle risorse condivise, nel functional programming, invece, la condivisione dei dati è eliminata, e questi sono passati tra le funzioni come parametri o valori di ritorno. MapReduce lavora secondo il principio del divide et impera, suddividendo l'operazione di calcolo 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. È MapReduce stesso che si occupa dell'esecuzione dei vari task di calcolo, del loro monitoraggio e della ripetizione dell'esecuzione in caso si verifichino problemi. Il framework lavora attraverso i compute node cioè dei nodi di calcolo che si trovano assieme ai DataNode di HDFS: infatti, lavorando in congiunzione con HDFS, MapReduce può eseguire i task di calcolo sui nodi dove i dati sono già presenti, aumentando così l'efficienza di calcolo.

2.2 HDFS: Hadoop Distributed Filesystem

Il filesystem distribuito di Hadoop è un filesystem particolare, specializzato, il cui scopo primario è di contenere i dati di input ed output per MapReduce. HDFS è strutturato sul modello di un filesystem POSIX, ma sacrifica l'aderenza allo standard per migliorare le prestazioni se utilizzato in coppia con Hadoop MapReduce. In particolare le caratteristiche salienti sono:

- Supporto di file di grandi dimensioni: un tipico file su HDFS ha dimensione compresa tra le centinaia di MegaByte e qualche TeraByte. Non esiste un limite esplicito alla dimensione dei file
- Accesso Sequenziale: supporta un paradigma write-one, read-many-times. Il troughput è ottimizzato a spese della latenza.
- Utilizzo di commodity hardware : progettato per lavorare con hardware a basso costo. Non prevede l'uso di dispositivi RAID, la tolleranza ai guasti viene fornita tramite replicazione sui nodi.

È opportuno far notare che HDFS risulta inadeguato per applicazioni che richiedano l'accesso ai dati con bassa latenza, o la scrittura di un singolo file da parte di più programmi. Nello stesso modo risulta problematica la gestione di un gran numero di piccoli file. La principale caratteristica che rende invece adatto HDFS a supportare l'esecuzione di MapReduce è data dalla gestione della replicazione, che verrà descritta di seguito.

2.2.1 Architettura

HDFS è composto da un'architettura di tipo master/slave, in cui ogni componente del sistema (slave) agisce esclusivamente sotto ordine di un master, a cui rendiconta le operazioni svolte. Si definisce NameNode l'unico master presente nella rete. Tutti gli altri componenti sono noti con il nome di DataNode. Questi ultimi contengono al loro interno i dati memorizzati dalle applicazioni, ed eseguono le operazioni di creazione, scrittura, lettura, rimozione e replicazione. Affinché possano operare necessitano di un esplicito ordine da parte del NameNode, che stabilisce quali DataNode debbano contenere determinati dati. Gli slave della rete non conoscono la presenza o la locazione degli altri DataNode, se non nel caso di un'esplicita richiesta di trasferimento dati da o per essi coordinata dal NameNode. Un'applicazione interagisce esclusivamente con quest'ultimo, che smista le operazioni ai nodi più opportuni. Il master non riceve o scrive direttamente alcun file, ne essi transitano da esso. Come già accennato, un file non può essere modificato una volta che lo si è memorizzato nell'infrastruttura. Una richiesta di creazione comporta la

divisione del file in blocchi, ognuno dei quali verrà memorizzato in un diverso DataNode, se ne esistono a sufficienza. Tipicamente un blocco ha dimensione pari a 64MB/128MB, anche se un'applicazione può modificarla per singoli file. Essa è uguale per ogni blocco del file, ad eccezione, eventualmente, dell'ultimo che sarà più piccolo degli altri. Il NameNode per ogni blocco memorizza una lista di datanode che ne possiedono una copia, ed i metadati relativi, come il file di appartenenza. I metadati sono mantenuti in RAM, in modo da velocizzarne l'accesso; per migliorare l'affidabilità, namenode mantiene anche una versione iniziale statica dei metadati su disco detta checkpoint , e un journal , con le modifiche differenziali. L'unione tra journal e checkpoint viene effettuato solo al riavvio, aggiornando il checkpoint. Il NameNode decide anche in quali DataNode i blocchi devono essere memorizzati. Il master, unico per costruzione, costituisce il Single Point Of Failure (SPOF). Senza il NameNode, il filesystem non sarebbe usufruibile in quanto i file contenuti in esso non sarebbero più raggiungibili.Per ovviare qualsiasi problematica riguardante il NameNode, ci sono altri due nodi con ruoili differenti. Il **checkpoint node** (detto anche secondary namenode), normalmente eseguito su un nodo master separato (in quanto ha le stesse richieste di memoria del namenode), si occupa di recuperare journal e checkpoint dal namenode e di effettuarne l'unione. La nuova versione viene quindi reinviata al namenode. È utilizzato per cluster che rimangano in funzione per lunghi periodi: questa operazione viene normalmente svolta una volta al giorno. Il secondo nodo è il **backup node** , che mantiene una copia read-only sincronizzata in tempo reale del namenode. Può essere utilizzato per creare i checkpoint al posto del checkpoint node. La versione 2.x di Hadoop ha aggiunto due nuovi concetti (*HDFS Federation* e *HDFS High-Availability*) che migliorano l'architettura di HDFS rispetto alle versioni precedenti.
La comunicazione con l'HDFS da parte di un client avviene tramite protocolli stratificati al di sopra della suite TCP/IP. E' importante sottolineare che il NameNode funge da server, pertanto non inizia mai una comunicazione, ma si limita a rispondere alle richieste dei DataNode. Per leggere i dati un client consulta prima il NameNode and chiede le locazioni dei blocchi del file.Il NameNode ritorna una lista composta da tutti i Data Node aventi un blocco di interesse.Il client esegue la lettura in modo sequenziale.Nel caso in cui il client debba eseguire una scrittura deve per prima cosa consultare il NameNode il quale gli risponde con una lista di Data Nodes.Il client una volta ottenuti i permessi può iniziare la scrittura a blocchi del file; successivamente i vari Data Nodes replicheranno i blocchi scritti sugli altri nodi.

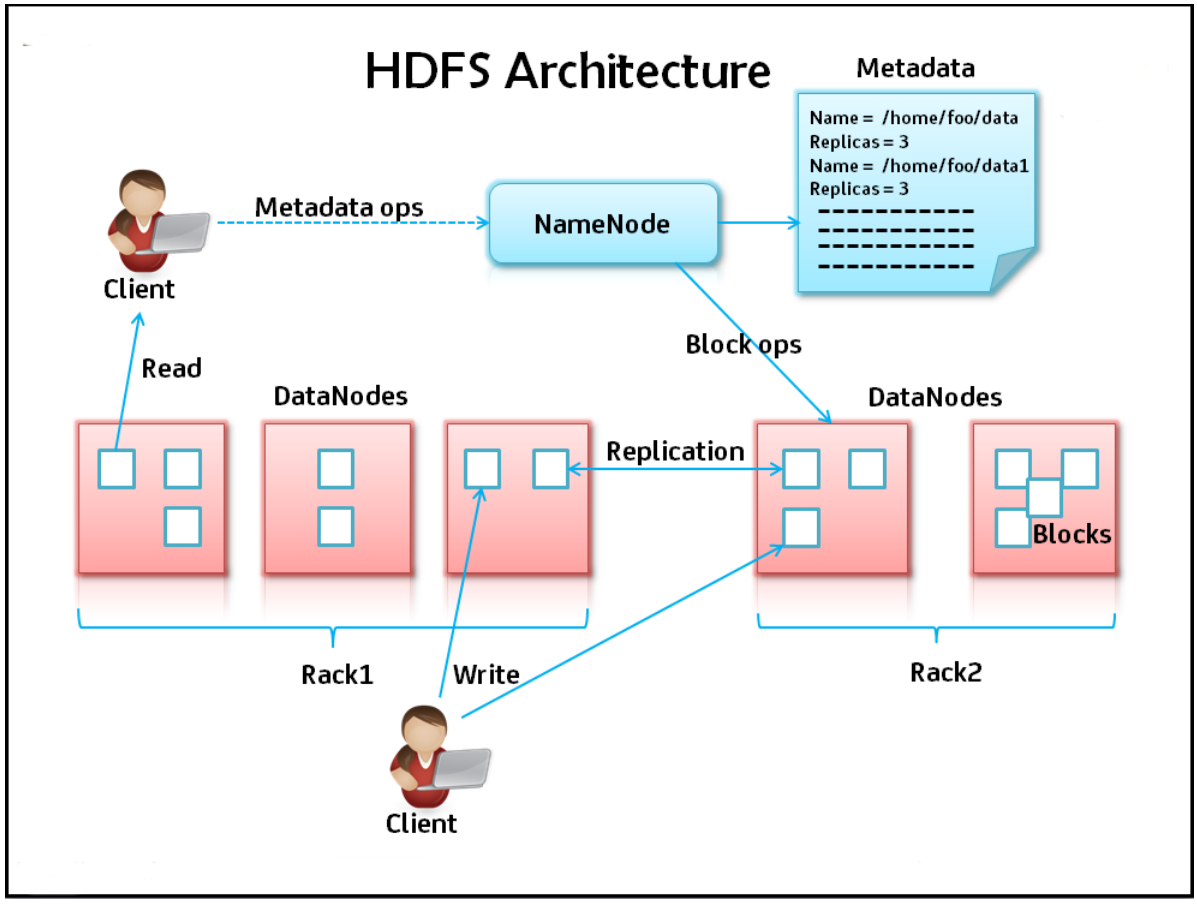

FIGURA 2.1 – Architettura HDFS - comprese operazioni di lettura, scrittura e resplicazione **[6]** .

2.2.2 Replicazione dei dati

La piattaforma è progettata per operare con un numero di DataNode anche nell'ordine delle centinaia. Con tale estensione, e ricordando che nel cloud è solitamente impiegato hardware comune, senza attenzioni speciali sulla sua durabilità, il fallimento di un nodo della rete è da prendere in seria considerazione. HDFS mette in atto un sistema di replicazione per prevenire il più possibile i danni causati da un'eventuale caduta di un nodo. Ogni file è caratterizzato da un fattore di replicazione, che rappresenta il numero di repliche che ogni suo blocco deve avere all'interno del file system. Alla creazione di un file, il NameNode non si limita a dividerlo in blocchi e a decidere la posizione di ognuno di essi, ma ordina anche la loro replicazione su più DataNode del sistema. Il NameNode

controlla costantemente che nessun blocco scenda al di sotto del fattore di replicazione del proprio file, neanche in seguito alla loro memorizzazione, ad esempio per guasti ai DataNode. In tal caso dispone nuove repliche in nuovi nodi. Ogni DataNode invia al NameNode un segnale periodico, denominato Heartbeat, per notificare la sua operatività. Questo è accompagnato dall'elenco dei file in lui memorizzati, in modo che il NameNode possa tenere traccia della presenza e dello stato di ogni blocco. La mancata ricezione dell'Heartbeat concretizza la caduta del nodo. Similmente alla grandezza di un blocco, un'applicazione può scegliere anche un diverso fattore di replicazione per ogni file. Di default questo è pari a tre, pertanto ogni blocco sarà memorizzato in almeno quattro DataNode diversi. La politica con cui questi sono scelti prevede di distribuire le repliche sia tra DataNode dello stesso rack sia tra nodi di rack diversi. Con il fattore di replicazione di default, una replica sarà memorizzata nello stesso rack del nodo contenente il blocco originale, mentre due repliche saranno memorizzate in un rack diverso, ognuna in un differente DataNode. In questo modo, Hadoop tenta di limitare il traffico inter-rack, utilizzando solo due di essi, salvaguardando allo stesso tempo il sistema dalla caduta di un intero rack, ad esempio per guasti di una certa entità all'interno di un data center. Le repliche sono totalmente identiche al blocco originale, pertanto una richiesta di lettura di un blocco può essere portata a termine indifferente- mente su esso o su una delle sue repliche. In questo modo il NameNode può ottimizzare la vicinanza di dati e applicazioni, minimizzando il tempo necessario a compiere le operazioni. L'impossibilità di modificare un file in seguito alla sua memorizzazione semplifica inoltre la gestione delle repliche, che sono sempre coerenti con l'originale.

2.3 Hive

Apache Hive è un tool per il data warehousing basato sull'infrastruttura Hadoop. Esso permette di operare con grandi data sets attraverso query in un linguaggio SQL-like. I dati utilizzati da Hive sono memorizzati nel File System HDFS di Hadoop, o in file systems compatibili con esso. E' quindi scalabile, tollerante ai fallimenti e garantisce un certo grado di parallelismo computazionale poichè le richieste, una volta arrivate all'infrastruttura Hadoop, vengono distribuite sui vari nodi che compongono il cluster, e vengono tradotte in funzioni MapReduce. Offre, in aggiunta alle funzionalità di Hadoop, i bridges JDBC e ODBC, interfacce grafiche per l'utilizzo di Hive-QL e svariati drivers per l'ottimizzazione delle query.

Hive offre le piene funzionalità di Hadoop, e quindi può avvalersi di un file system distribuito come HDFS e di una parallelizzazione dei lavori con MapReduce. Interfacciarsi con Hadoop, prima, significava dover fornirgli le funzioni di Map e Reduce, e quin-

di implementarle in Java. Hive risolve queste problematiche offrendosi come traduttore (Hive)SQL-MapReduce. In questo modo si sfrutta l'ecosistema Hadoop per risolvere i problemi legati all'incapacità di gestione dei Big data da parte dei sistemi RDBMS e si utilizza la comodità di un linguaggio simile all'SQL, tipico di sistemi relazionali.

2.3.1 Architettura Hive

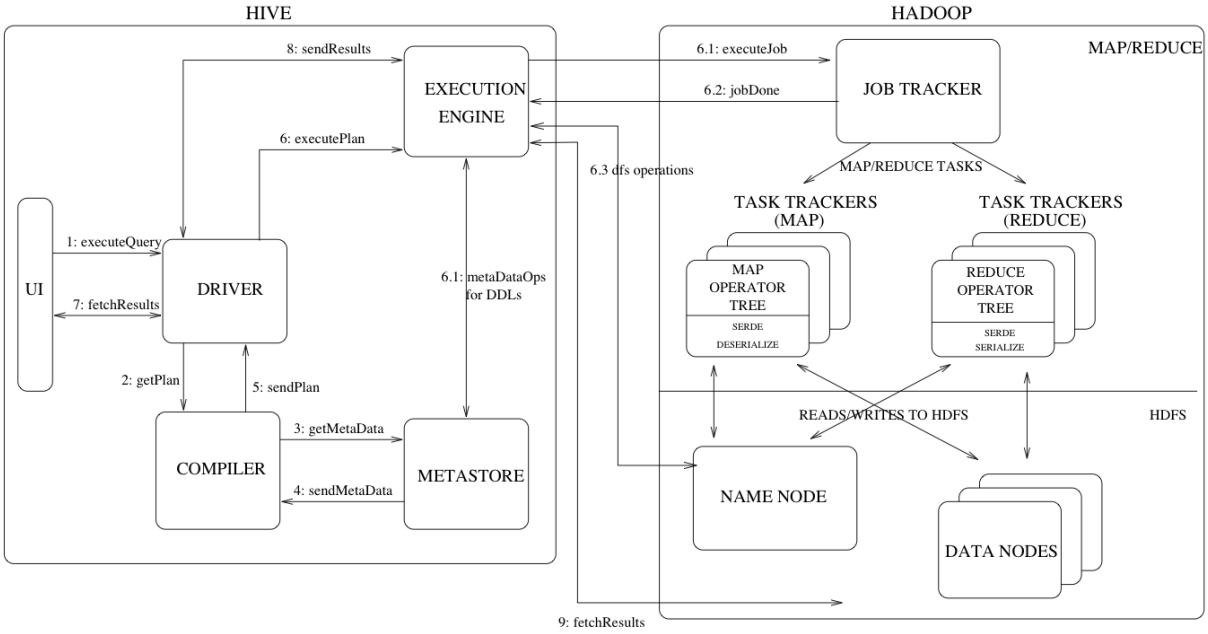

FIGURA 2.2 – Principali componenti di hive e la loro interazione con Hadoop

La figura 2.2 mostra i componenti più importanti di Hive e le interazioni con Hadoop. Come si può leggere dalla figura i componenti principali sono:

- UI: l'interfaccia grafica utilizzata dall'utente per effettuare interrogazioni e altre operazioni sul sistema. E' disponibile sia una interfaccia web ed una da linea di comando.
- Driver: è il componente che riceve le query.
- Compiler: è il componente che effettua il parsing e l'analisi semantica della query ed eventualmente genera un piano di esecuzione basandosi anche sui metadati presenti nel metastore.

- Metastore: si occupa della memorizzazione di tutte le informazioni strutturali delle tabelle e partizioni includendo anche le colonne, il loro tipo, i serializzatori/deserializzatori per scrivere e leggere i dati ed i corrispondenti file HDFS dove i dati sono memorizzati.

- Execution Engine: componente che esegue il piano di esecuzione creato precedentemente dal compiler. s

La figura 2.2 **[7]** mostra anche un tipico flusso di esecuzione di una query attraverso il sistema. L'interfaccia grafica chiama l'interfaccia di esecuzione tramite il Driver (step 1). Il Driver crea un handle della sessione per la query invia quest'ultima al compiler per generare un piano di esecuzione (step 2). Il compiler ottiene dal metastore i metadati necessari (step 3 e 4). Questi metadati sono usati per una fase di controllo del l'albero che include per esempio, l'esclusione di partizioni basandosi sui predicati di selezione. Il piano generato dal compiler (step 5) è un grafo aciclico diretto di stadi dove ognuno di questi può essere un map/reduce job, un'operazione sui metadati o un operazione sull'HDFS. Per quanto riguarda gli stadi map/reduce, il piano contiene i map operator tree (operazioni eseguite sui mappers) ed i reduce operator tree(operazioni che richiedono reducers). L' execution engine invia questi stadi ai componenti appropriati (step 6, 6.1, 6.2, 6.3). In ogni task (mapper/reducer) il serializzatore associato ad una tabella è usato per leggere le righe dai file sull'HDFS e queste sono approvate tramite l'operator tree associato. Una volta che l'output è stato generato viene salvato temporaneamente sul un file HDFS. I file temporanei usati per fornire dati ai successivi stadi map/reduce del piano. Per le operazioni DML il file finale temporaneo è spostato nella stessa locazione della tabella. Per le query, il contenuto dei file temporanei è letto dal execution engine direttamente dall'HDFS (step 7, 8 and 9).

2.4 Parquet

La memorizzazione a colonne (columnar storage) è una tecnica che permette di ottimizzare i ca- richi di lavoro analitici in RDBMS paralleli. Le performance e i benefici della compressione atta a immagazzinare e processare grandi quantità di dati sono già ampiamente documentate in letteratura.L'obiettivo è quello di mantenere l'I/O al minimo leggendo dal disco solamente gli specifici dati necessari alla query. Utilizzando Parquet in Twitter, è stata sperimentato una notevole diminuzione (pari ad un terzo) nei loro dataset. Lo scan time ha a sua volta subito una forte riduzione ad una frazione dell'originale nei casi più comuni in cui si richiede solamente un sottoinsieme delle colonne. Il principio è piuttosto semplice: invece che avere un adottare un layout di riga tradizionale, i dati sono scritti

una colonna alla volta. Dato uno schema flat la trasformazione di righe in colonne è una operazione lineare che non presenta alcuna difficoltà, non è così invece quando si ha a che fare con strutture dati innestate. Parquet è un formato di file open source per Hadoop che fornisce la memorizzazione a colonne. Inizialmente nato dallo sforzo congiunto di Twitter e Cloudera, ha oggi molti altri un contributors incluse alcune compagnie (Criteo). Parquet memorizza le strutture dati annidate in un formato flat colonnare (flat columnar format) utilizzando una tecnica illustrata nel paper di Google relativo a Dremel. Il modello implementato è basato sul precedente articolo ma la descrizione qui trattata è più accessibile e semplice. In primo luogo sarà descritto il modello utilizzato per la rappresentazione delle strutture dati annidate. Dopodiché sarà illustrato come questo modello può essere rappresentato come una lista di colonne flat per terminare con il perché dell'efficacia di questa.

La seguente figura mostra una rappresentazione di un columnar storage.

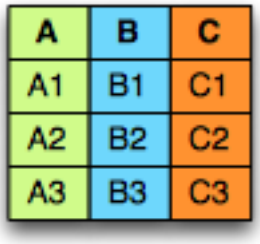

FIGURA 2.3 – Esempio grafico di un columnar storage **[8]** .

In uno storage row-oriented, i dati sono disposti in una riga nel seguente modo.

FIGURA 2.4 – Dati salvati in formato row-oriented **[8]** .

Mentre in uno storage column-oriented, essi sono disposti una colonna per volta.

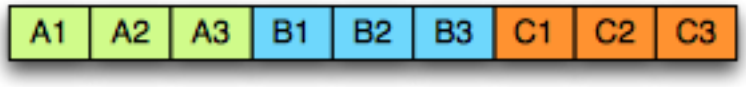

FIGURA 2.5 – Dati salvati in formato column-oriented **[8]** .

Ci sono diversi vantaggi nell'utilizzare il formato colonnare:

- è possibile una miglior compressione siccome i dati sono omologhi. Lo spazio salvato èrilevante quando si ha a che fare con le dimensione di un cluster Hadoop;
- l'I/O è ridotto in quanto è possibile analizzare efficientemente solo un sottoinsieme delle colonne durante la lettura dei dati. La miglior compressione riduce inoltre anche la larghezza di banda richiesta per leggere l'input;
- siccome vengono salvati dati dello stesso tipo per ogni colonna, si può utilizzare una codifica più adatta alle pipeline dei processori moderni in modo da rendere i diversi flussi di istruzioni meglio predicibili.

2.4.1 Il modello

Per memorizzare in un formato colonnare è prima di tutto necessario descrivere le strutture datu utilizzando uno schema. Questo è fatto utilizzando un modello simile ai Protocol buffers (un metodo per la serializzazione delle strutture dati). Questo modello è semplificato (minimalistic) in quanto rappresenta l'annidamento utilizzando gruppi di campi e la ripetizione utilizzando campi ripetuti. Non c'è alcun bisogno di strutture dati complesse come Mappe, Liste o Insieme (Maps, Lists, Sets) in quanto esse possono tutte essere mappate in una combinazione di campi ripetuti e gruppi.La radice di uno schema è un gruppo di campi chiamato messaggio. Ogni campo ha tre attributi: una ripetizione, un tipo e un nome. Il tipo di un campo è un gruppo oppure un tipo primitivo (int, float, boolean, string) e l'attributo ripetizione può assumere uno dei seguenti valori:

- required: esattamente una occorrenza
- optional: 0 o 1 occorrenza
- repeated: 0 o più occorrenze

Per esempio, qui sotto è presentato una schema che rappresenta un elenco telefonico.

LISTATO 2.1 – caption

```
1 message AddressBook {
2   required string owner;
3   repeated string ownerPhoneNumbers;
4   repeated group contacts {
5     required string name;
6     optional string phoneNumber;
```

```
7     }
8 }
```

Liste o insiemi possono essere rappresentate attraverso un campo multiplo.

Schema: List of Strings	**Data**: ["a", "b", "c", ...]

message ExampleList {
repeated string list;
}
	{
	list: "a",
	list: "b",
	list: "c",
	…
	}

FIGURA 2.6 – Esempio rappresentazione liste **[8]** .

Una Mappa è equivalente alla ripetizione di un campo contenente un gruppo di di coppie chiave-valore dove la chiave è *required*.

Schema: Map of strings to strings	**Data**: {"AL" => "Alabama", ...}

message ExampleMap {
repeated group map {
required string key;
optional string value;
}
}
	{
	map: {
	key: "AL",
	value: "Alabama"
	},
	map: {
	key: "AK",
	value: "Alaska"
	},
	…
	}

FIGURA 2.7 – Esempio rappresentazione mappa **[8]** .

2.4.2 Il formato a colonne (Columnar format)

Questo formato fornisce una codifica e decodifica più efficiente grazie alla memorizzazione di valori dello stesso tipo primitivo insieme. Per memorizzare strutture dati annidate nel formato colonnare, è necessario mappare lo schema in una lista di colonne in modo da poter scrivere i record in colonne flat e leggerli indietro come strutture annidate originali. In Parquet, viene creata una colonna per ogni campo di tipo primitivo presente nello schema. Se si rappresenta lo schema come un albero, i tipi primitivi sono le foglie dell'albero.

L'elenco telefonico precedente è rappresentato come albero nel seguente modo.

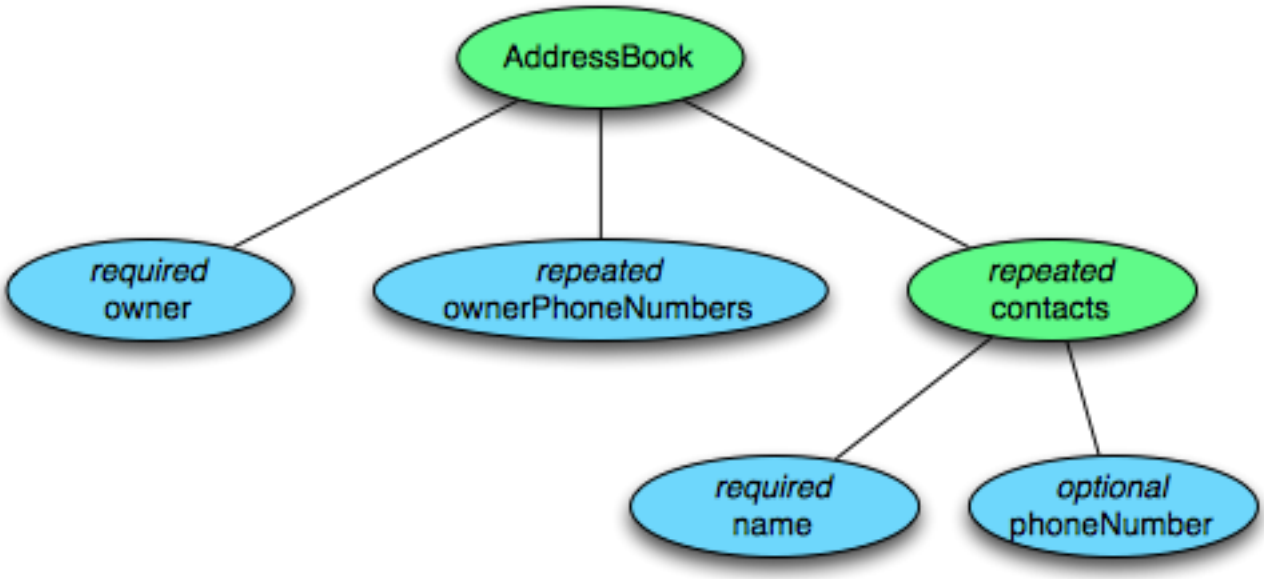

FIGURA 2.8 – Elenco telefonico rappresentato come albero **[8]** .

Per rappresentare i dati nel formato colonnare si crea una colonna per ogni cella di tipo primitivo mostrata in azzurro.

Column	Type
owner	string
ownerPhoneNumbers	string
contacts.name	string
contacts.phoneNumber	string

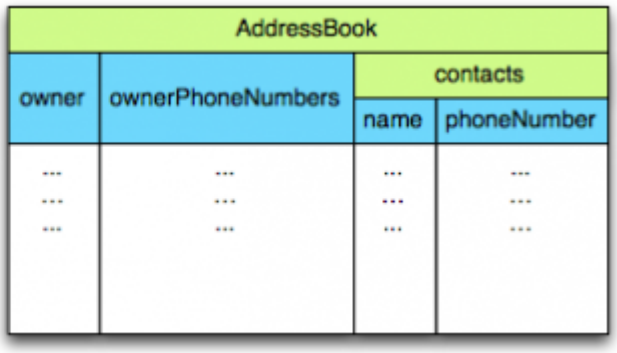

FIGURA 2.9 – Metodo di rappresentazione dei dati in formato colonnare **[8]** .

La struttura dei record è catturata per ogni valore da due interi chiamati livello della ripetizione (*repetition level*) e definizione del livello (*definition level*). Utilizzando questi due interi, è possibile ricostruire le strutture annidata.

Capitolo 3

Impala: introduzione

Impala è un MPP (*Massively Parallel Processing*) SQL query engine progettato appositamente per sfruttare la flessibilità e la scalabilità di Hadoop. L'obiettivo di Impala è quello di combinare il familiare supporto SQL e multi-utente che caratterizzano un database analitico tradizionale con la scalabilità e la flessibilità di Apache Hadoop. Impala è una tecnologia relativamente giovane (prima beta rilasciata nel 2012) e diversamente da molti suoi competitors, Impala è stato scritto da zero in C++ e Java con un approccio bottom-up; mantiene la flessibilità che contraddistingue Hadoop, utilizzando i componenti standard (HDFS, HBase, Metastore, YARN, Sentry) e supportando la maggior parte dei formati file utilizzati in questo ambito (Parquet, Avro, RCFile). La maggior parte dei competitors utilizza meccanismi come MapReduce o letture remote di dati, che portano ad un consistente aumento dei tempi di latenza. Al contrario Impala implementa una architettura distribuita basata su processi demone, i quali sono incaricati di coprire tutti gli aspetti che caratterizzano l'esecuzione di una query. Il risultato, in termini di prestazioni, è alla pari o superiore dei più noti DBMS MPP commerciali. Impala essendo un query engine perfettamente integrato nell'ambiente Hadoop che utilizza molti componenti standard di quest'ultimo con lo scopo di offrire un'esperienza il più possibile simile a quella che un utente ha quando utilizza i comuni RDBMS. Impala è stato progettato specificatamente per essere integrato coi i pre-esistenti business intelligence environments, per questo supporta molti degli standard utilizzati in questa industria, come per esempio: la possibilità ci connettersi via ODBC o JDBC, la presenza di un sistema di autenticazione basato su ruoli e privilegi. Per poter esplorare i dati presenti sull' HDFS, l'utente può creare tabelle tramite un DDL arricchito che permette di definire lo schema logico dei dati e il loro layout fisico (formati file, struttura directory HDFS). Queste tabelle saranno facilmente interrogabili tramite, il familiare linguaggio di interrogazioni su database relazionali, SQL **[10]**; **[11]** .

3.1 Vantaggi di Impala

- Linguaggio SQL familiare a quello che gli sviluppatori e analisti di dati già conoscono.

- Possibilità di eseguire query in modo interattivo grandi quantità di dati (big data) in Apache Hadoop

- Le Query distribuite in un ambiente cluster permettono di scalare in modo conveniente e di utilizzare hardware commodity

- Possibilità di condividere i file di dati tra le diverse componenti senza dover effettuare nessuna operazione di copia o import / export dei dati; per esempio, è possibile "scrivere" con Pig e "leggere" con Impala, o "scrivere" con Impala e leggere con Hive

3.2 Progettazione fisica dello schema

Durante la creazione di una tabella è possibile specificare, tramite il DDL arricchito sopra citato, come questa deve essere partizionata. Per quanto riguarda le tabelle non partizionate i dati sono memorizzati direttamente della cartella "root" rappresentante quella specifica tabella. Nelle tabelle partizionate invece, i file sono memorizzati nelle apposite sottocartelle create per ciascuna partizione. Per esempio supponiamo di avere una generica tabella T; se questa non è partizionata verrà rappresentata nel HDFS in questo modo $\langle root/table\_name\rangle/datafile.file\_format$, mentre se partzionata $\langle root/table\_name\rangle/column1/datafile.file\_format$. E' importante notare che, nel caso una tabella sia partizionata, i file di una singola partizione sono comunque distribuiti in modo casuale su tutti i nodi (HDFS data nodes). Impala, inoltre fornisce una grande flessibilità nel scegliere il formato con il quale i file vengono memorizzati. Attualmente supporta formati compressi e non, file sequenziali, RCFile, Avro e infine Parquet che risulta essere l'opzione di archiviazione con più alto rendimento. Il DDL mette anche a disposizione delle clausole per convertire dati (intere tabelle) da un formato ad un altro; per il benchmark realizzato di seguito si fatto utilizzo di questa caratteristica per trasformare dump file personalizzati (non in formato tradizionale come per esempio CSV) in file parquet.

3.3 Impala Query Language

Il linguaggio utilizzato da Impala per effettuare interrogazioni è basato su SQL standard, e allo stesso tempo fornisce un alto grado di compatibilità con HiveQL. Le istruzioni del linguaggio utilizzato da Hive si basano anch'esso su SQL standard, e molte delle istruzioni combaciano esattamente con quelle usate da Impala. Tuttavia alcune istruzioni sono differenti. Di sotto verranno elencate le principali proprietà del linguaggio SQL di Impala.

- Possiede un Data Definition Language (DDL) arricchito.
- Per memorizzare le strutture delle tabelle e le loro proprietà, Hive usa Metastore; Impala utilizza lo stesso Metastore per salvare le sue informazioni.
- Offre un Data Manipulation Language (DML), anche se non incompleto a causa del sistema di memorizzazione.
- i tipi di dato supportarti sono gli stessi di Hive (int , tinyint , smallint , bigint , float , double , boolean ,string , and timestamp)
- Supporta molte istruzioni e clausole offerte da HiveQL, come SELECT, FROM, JOIN, LIMIT, DISTINCT, INSERT TO, INSERT OVERWRITE e molte altre.

3.4 Architettura

Impala è un query engine ad elevato parallelismo progettato per lavorare su un preesistente cluster Hadoop composto da centinaia di macchine. Esso non è legato al motore di archiviazione sottostante, a differenza dei sistemi tradizionali di gestione di database relazionali dove l'elaborazione delle query e il motore di storage sottostante sono componenti di un unico sistema. Una distribuzione di Impala è composta da tre servizi **[9]** .

Il servizio Impala (**impalad)** è incariato di accettare le query dai vari processi client e gestire la loro esecuzione nel cluster. Questo demone è anche responsabile dell'esecuzione di singoli frammenti di query per conto di altri demoni Impala. Tutti i demoni Impala sono di pari livello, ciascuno di essi posso assumere differenti ruoli (non contemporaneamente) . Quando un demone assume il ruolo di gestore di una determinata query, questo viene nominato coordinatore per quella particolare query. Questa struttura aiuta a contrastare eventuali problematiche di fault-tolerance e load-balancing. Un demone Impala (impalad) è eseguito su ogni macchina del cluster che sta anche eseguendo un processo datanode, quindi timpicamente è in esecuzione un impalad su ogni macchina del cluster.

Questo consente ad Impala di trarre vantaggio dai dati in locale leggendo direttamente i dati dal filesystem senza utilizzare la rete (molto più lenta).

Il demone **Statestore** è un servizio di tipo publish-subscribe che diffonde i metadati a tutti i processi impala del cluster.

Infine il demone **Catalog**, serve da repository e da gateway di accesso metadati. Attraverso questo demone Impala può eseguire comandi DDL che sono riflessi su catalog stores esterni come quelli di Hive Metastore. I cambiamenti del system catalog sono trasmessi utilizzando il Statestore.

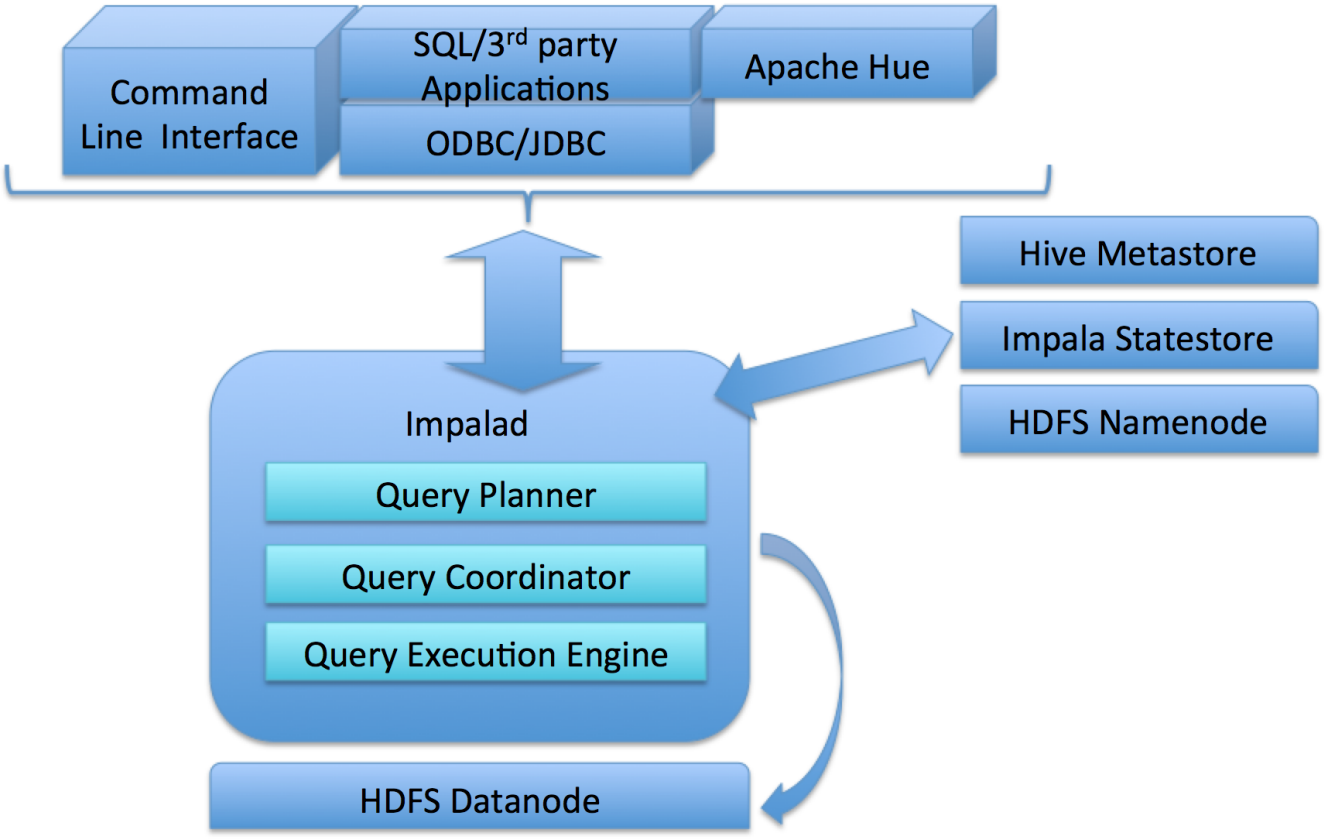

FIGURA 3.1 – Singola istanza di un demone impalad con le sue principali interazione con gli altri denomi/processi **[12]**

3.4.1 Distribuzione dello stato

La struttura nodo-simmetrica di Impala fa si che tutti i nodi debbano essere in grado di accettare e eseguire una query. Quindi tutti i nodi devono avere, per esempio, una versione aggiornata del system catalog e una visione degli altri membri del cluster così da consentire la corretta pianificazione delle query. Impala adotta un "servizio" di tipo push degli aggiornameti di interesse per le altre parti. Questo compito lo svolge un servizio chiamato statestore che distribuisce le modifiche dei metadati a tutti i subscribers. Lo statestore mantiene un insieme di topics, compisti da triplette (chiave, valore, versione) chiamate entries dove 'chiave' e 'valore' sono array di byte mentre 'versione' è un integer di 64 bit. Un topic è difinito da una applicazione, quindi lo statestore non ha alcuna conoscenza del contenuto di nessun topic. I topic sono persistenti soltanto durante il tempo di vita dello statestore, quindi un riavvio del servizio ne provoca la cancellazione. I processi che desiderano ricevere aggiornamenti di qualsiasi topic sono chiamati subscribers. Per ricevere gli aggiornamenti i processi devono registrarsi con lo statestore all'avvio e fornendo una lista di topics. Lo statestore invia una risposta alle singole richeste dei subscribers contenente tutte le voci attualmente contenute nel topic richiesto. Dopo la registrazione, lo statestore manda periodicamente due tipi di messaggi ad ogni subscribers. Il primo tipo di messaggio è un aggiornamento riguardante il topic, e consiste in tutti i cambiamenti (inserimenti, modifiche e cancellazioni) di un topic dal momento in cui l'ultimo aggiornmento è stato spedito con successo al subscriber. Ogni subscriber mantiene un identificatore per la versione più recente di ogni topic; ciò consente allo statestore di inviare solo le differenze riscontrate tra la l'ultima versione e la penultima. In risposta all'aggiornamento ricevuto, ogni subscriber invia una lista di cambiamenti che desidera apportare ai suoi argomenti sottoscritti. L'applicazione di questi cambiamenti è garantita al momento in cui verrà ricevuto il successivo aggiornamento. Il secondo tipo di messaggio è di tipo keepalive. Lo statestore usa questo tipo di messaggio per matenere, con ogni subscriber, la connessione attiva. Se lo statestore nota che la connessione con un subscriber non più attiva non invierà più aggiornamenti verso questultimo. Alcuni topic possono essere contrassegnati come "transient", il che significa che se la connessione del loro "possessore" dovesse cadere questi verranno cancellati. Lo statestore fornisce una semantica molto debole; i vari subscribers possono ottenere gli stessi aggiornamenti in tempi diversi, il che può portare a visioni differenti del contenuto dei topic. Detto ciò, Impala usa i topic metadata solo per prendere decisioni in locale, senza alcun tipo di coordinazione a livello di cluster. Prendiamo per esempio un piano di esecuzione di una query (basato sui metadati del catalog), che è eseguito su un singolo nodo. Quando l'itero piano è stato calcolato, tutte le informazione necessare all'esecuzione del piano sono distribuite agli altri nodi (exchange nodes), non è necessario che i metadati dei nodi esecutori siano della stessa versione del nodo

coordinatore. Il fatto di avere una singola istanza dello statestore presente nel cluster, non comporta problemi nemmeno in caso di cluster di medie dimensioni e con opportune configurazioni può servire anche cluster di grandi dimensioni. Lo statestore non ha nessuna necessità di salvare i metadati in modo persistente in quando questi ultimi sono ricevuti dai subscribers. Nel caso lo statestore necessiti di un riavvio il suo stato può essere recuperato durante la fase iniziale di registrazione dei subdcribers. Or ifthe machine that the statestore is running on fails, a new statestore process can be started elsewhere, and subscribers may fail over to it. There is no built-in failover mechanism in Impala, instead deployments commonly use a retargetable DNS entry to force subscribers to automatically move to the new process instance.

3.4.2 Catalog Service

Il servizio catalog fornisce metadati ai demoni Impala attraverso il meccanismo broadcast dello statestore, e esegue operazioni DDL per conto dei demoni Impala. Questo servizio prendi informazioni da metadata di terze parti (per esempio, HDFS Namenode), e aggrega queste informazioni in una struttura catalog compatibile con Impala. Questa architettura permette ad Impala di essere indipendente dal tipo di metadata store, il che permette di aggiungere un supporto a nuovi tipi di metadata store in un tempo ristretto. Qualsiasi cambiamento al system catalog sono diffusi tramite lo statestore. Dato che i catalogs spesso sono di dimensioni molto grandi, e gli accesi alle tabelle sono raramente uniformi, il catalog service carica solo informazioni riassuntive per ogni tabella presente all'avvio. Tabelle più dettagliate potranno essere caricate in background dagli store di terze parti. Se una tabella è richiesta prima di essere completamente caricata, un demone Impala noterà questo problema e farà una richiesta di priorità al catalog service.

3.4.3 Frontend

La parte frontend è responsabile della trasformazione del codice SQL in piano di esecuzione eseguibile dalla backend di Impala. Il frontend è scritto Java e consiste in un SQL parser e un ottimizzatore di query. Il processo di compilazione di una query segue una tradizionale divisione del lavoro: parsing testuale della query, analisi semantica, pianificazione e ottimizzazione della query. Al query planner vengoni forniti in input il parse tree e delle informazioni globali della query (table/columnidentifiers, equivalence classes, etc.). Un piano di esecuzione è suddiviso in due fasi:

1. Pianificazione (effettuata da un singolo nodo)
2. Parallelizzazione del piano e frammentazione (esecuzione divisa fra i nodi)

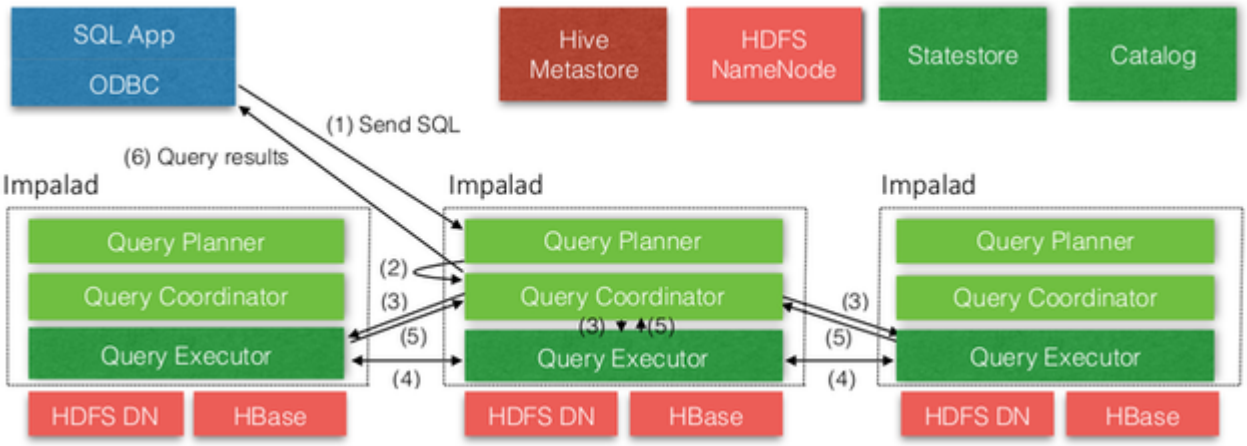

FIGURA 3.2 – Esempio di flusso di esecuzione distribuito

Nella prima fase, il parse tree è tradotto in un piano di esecuzione non distribuito, costituito dai seguenti nodi: HDFS/HBase scan, hash join, cross join, union, hash aggregation, sort, top-n, and analytic evaluation. In questo step vengono: eseguite selezioni nei nodi più bassi, scartate le partizioni non necessarie, settatti evenutali limiti, eseguite le proizioni sulle colonne e operazioni di ottimizzazione come il riordino delle operazioni di join reordering. In particolare Impala usa semplici euristiche per evitare di valutare tutte le possibili combinazioni di join. La stima dei costi si basa sulla cardinalità delle tabelle o partizioni e il numero di valori distinti per ogni tabella; Nello specifico, per il calcolo dei valori distinti, viene utilizzato un algoritmo denominato HyperLogLog è in grado di minimizzare notevolmente l'utilizzo della RAM. Nelle prossime versioni, Impala, ha annunciato che saranno disponibili anche gli istogrammi.

La seconda fase utilizza il piano di esecuzione, calcolato dal nodo coordinatore, per produrre un piano di esecuzione distribuito. L'obiettivo complessivo è minimizzare lo spostamento di dati e massimizzare le letture in locale (le letture remote in HDFS sono considerevolmente più lente delle letture in locale). Il piano è reso distribuito aggiungendo nodi di scambio tra i nodi dell'albero, e aggiungendo nodi extra (non-exchange) per minimizzare lo spostamento di dati attraverso la rete (local aggregation nodes). Durante questa fase si decide la strategia (broadcast / partitioned) di join per ogni nodo (join node), viene determino così l'ordine di join che rimarra fisso da questo punto in poi. Nel caso sia stato scelta la strategia *broadcast* la tabella con grandezza minore viene spedita a tutti gli altri nodi, mentre nel caso *partitioned* entrambe le tabelle vengono parizionate e le partizioni spedite ai vari nodi. Impala sceglie qual'è la strategia migliore per ridurre al minimo la quantità di dati scambiati attraverso la rete. Tutte le aggregazioni vengono processate

eseguendo una serie di pre-aggregazioni locali seguite da un unica operazione di aggregazione. Per le aggregazioni di raggruppamento, l'output di pre-aggregazione è partizionato sulle espressioni di raggruppamento e l'aggregazione di tipo merge viene fatto in parallelo su tutti i nodi partecipanti.Infine il piano viene distribuito fra tutti i nodi.Ogni porzione del piano è posta all'interno di un *fragment*, unità di esecuzione a livello backend di Impala. Un fragment incapsula una porzione di piano che opera in una singola macchina e su una partizione di dati presenti sulla stessa.
La figura 3.3 mostra in un esempio le due fasi della pianificazione di una query. La parte a sinistra mostra il piano di esecuzione, calcolato dal singolo nodo, di una query nella quale si effettua: un join tra le tabelle HDFS t1,t2 e una tabella Hbase t3; un aggregazione; un limit (top-n). La parte destra mostra come il piano viene frammentato e suddiviso. I rettangoli smussati indicano i singoli nodi, le freccie lo scambio di dati. Il join fra le tabelle t1 e t2 è di tipologia partitioned. Le scansioni sono effettuare da i singoli nodi che provvederanno poi ad inviare il risultato della lettura (partizionato in base a una funzione di hash) hai nodi responsabili del join. Il successivo join, di tipo broadcast, con t3 è posto nello stesso frammento del join precedente perché un join di tipo broadcast la partizione di data esistente (il risultati dei join fra t1, t2, e t3 sono partizionati tramite un algoritmo di hash che si basa sulle chiavi di join di t1 e t2). Dopo i joins è eseguita un'aggregazione distribuita divisa in due fasi, dove una pre-aggregazione viene calcolata nello stesso frammento dell'ultimo join; i risultati di quest'ultima sono scambiati tramite un algoritmo di hashing basato sulle chiavi di raggruppamento e poi aggregati un ulteriore volta per calcolare l'aggregazione finale. Lo stesso approccio a due fasi è applicato alla clausola LIMIT (top-n); il top-n finale è eseguito dal nodo coordinatore che ritorna il risultato all'utente.

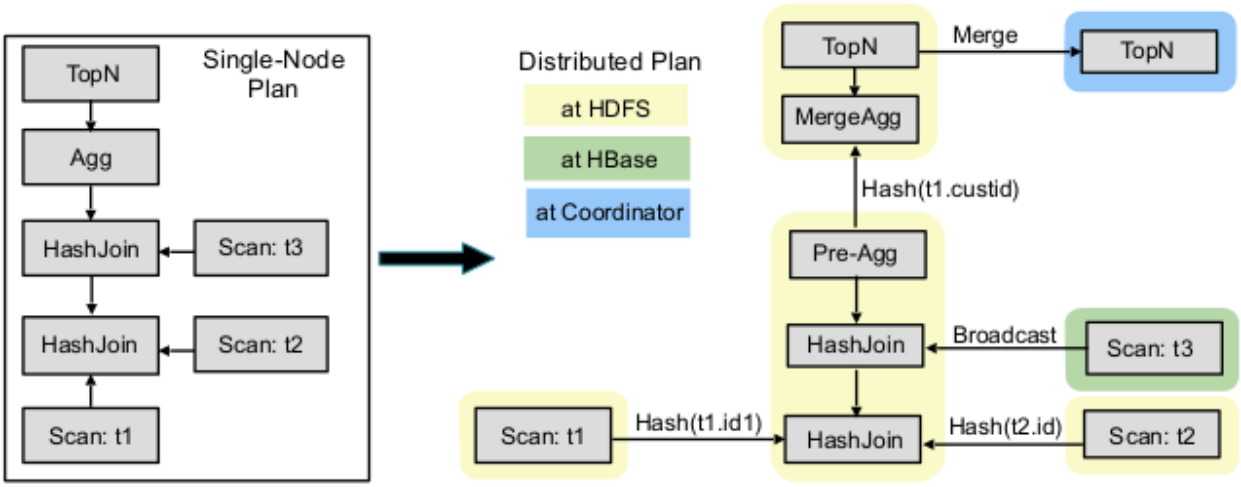

FIGURA 3.3 – Esempio di costruzione di un piano di esecuzione in due fasi **[13]** .

3.5 File formats

Impala supporta diversi formati di file utilizzati anche in Apache Hadoop. Impala può caricare ed interrogare file prodotti da altri componenti di Hadoop come Pig o MapReduce, e i file prodotti da Impala possono essere usati, a loro volta, dagli altri componenti di Hadoop. Il formato di file utilizzato per una tabella ha un impatto significativo sulle performance delle interrogazioni. Alcuni formati includono un supporto alla compressione che diminuisce lo spazio spazio occupato su disco occupato e di conseguenza le letture di I/O e utilizzo della CPU richiesti per deserializzare i dati. La quantità di risorse I/O e CPU richieste posso essere un fattore limitante in termini di performance dell'esecuzione di una query. Per ridurre questo possibile effetto i dati sono vengono spesso compressi; applicando un algoritmo di compressione a dati, una minore quantità di questi verrà trasferita dal disco alla memoria. Questo riduce il tempo speso per trasferire i dati, ma bisogna sempre tener conto delle risorse utilizzate dalla CPU per decomprimere i dati. Impala può interrogare file codificati nei formati più popolari e compressi con codecs usati in Hadoop. Impala può creare ed inserire dati nelle tabelle che utilizzano alcuni formati, alcuni non sono supportati; per ovviare a questo problema si possono effettuare tali operazioni con Hive per poi, tramite la shell di Impala, lanciare il comando INVALIDATE METADATA e interrogare normalmente tramite Impala. I formati più utilizzati sono:

- RCFile: Row Columnar File (RCFile) fù sviluppato originariamente da Facebook per lo storage di backend del suo sistema di data warehouse Hive, che fu il primo

sistema mainstream SQL su Hadoop disponibile come open source. RCFile punta a offrire:

  - caricamento veloce dei dati
  - elaborazione rapida delle query
  - uso efficiente dello storage
  - adattabilità ai carichi di lavoro dinamici

- ORC: Optimized Row Columnar (ORC) ha lo scopo di combinare le prestazioni di RCFile con la flessibilità di Avro. È studiato soprattutto per lavorare con Apache Hive ed è stato sviluppato inizialmente da Hortonworks per superare i limiti percepiti degli altri formati di file disponibili.

- Apache Avro: è un contenitore di file e un formato di serializzazione di dati binari orientato allo schema. È la nostra scelta in tutto il libro per quanto riguarda il formato binario. Può essere suddiviso e compresso, il che lo rende efficace per l'elaborazione dei dati con framework come MapReduce. Sono molti i progetti che includono un supporto specifico e un'integrazione con Avro, che si può quindi applicare diffusamente. Quando i dati vengono salvati in un file Avro, con esso viene salvato anche il suo schema, definito come un oggetto JSON. Un file può essere elaborato successivamente da una terza parte senza una conoscenza preliminare di come i dati sono codificati. Questo permette l'autodescrizione e ne agevola l'uso con i linguaggi dinamici e di scripting. Il modello di schema in lettura favorisce anche l'efficienza dello storage dei record, poiché non occorre che i singoli campi vengano taggati. Nei prossimi capitoli vedremo come queste proprietà possono facilitare la gestione del ciclo di vita dei dati e consentire operazioni complesse come la migrazione dei dati.

- Parquet: è il frutto dello sforzo congiunto di Cloudera, Twitter e Criteo, e ora è stato donato all'Apache Software Foundation. I suoi obiettivi sono quelli di fornire un formato di file colonnare moderno e a prestazioni elevate da usare con Cloudera Impala. Come Impala, è stato ispirato dal documento su Dremel. Permette di lavorare con strutture di dati complesse e annidate e consente una codifica efficiente a livello di colonne. Attualmente è consigliato dalla stessa Cloudera come formato più performante. Per maggiori dettagli vedere sezione x.

3.6 Best practice per l'esecuzione di query

Per distribuire e parallelizzare il lavoro in modo efficiente è necessario che Impala sia a conoscenza di alcune informazioni (statistiche) riguardanti le tabelle coinvolte nella query, come per esempio il volume di dati e come i dati sono distribuiti. Le statistiche devono essere aggiornate manualmente tramite uno specifico comando COMPUTE STATS ogni qualvolta si crei una tabella o la si aggiorni.

Per eseguire una query e far si che le risorse vengano usate in modo efficiente ci si può avvalere di alcuni accorgimenti:

- usare le table e column statistic
- file format appropriato per i dati
  - file block size appropriato
- partizionare tabelle di grandi dimensioni
- considerare varianti di join
- HDFS caching
- verificare il piano di esecuzione tramite EXPLAIN
  - cambiare livello di dettaglio SET EXPLAIN_LEVEL = (1,2,3)

3.6.1 Formato file Parquet

Parquet è un formato di file binario e column-oriented molto efficiente per query di grandi dimensioni. Parquet è particolarmente efficace per query di scansione su poche colonne in una tabella con molte di queste, oppure per operazioni di aggregazione come SUM () e AVG () che devono processare tutti i dati per una determinata colonna. Un file parquet usualmente è scomposto in più data file. Ogni data file contiene i valori per una serie di righe (row group). All'interno di un data file, i valori di ogni colonna sono organizzati in modo tale da essere tutti adiacenti, consentendo così una buona compressione, che tenendo conto del fattore di replicazione dei dati, consente di risparmiare 3x spazio risparmiato. Le Query lanciate su una tabella Parquet possono recuperare e analizzare i valori delle colonne in modo rapido e con un i / O minimo.

Sebbene Parquet è un formato column-oriented, non ci si aspetti di trovare un data file per ogni colonna. Parquet mantiene tutti i dati relativi ad una riga nello stesso data file, per garantire che tutte le colonne di una riga siano disponibili per essere processate sullo

stesso nodo. Quello che Parquet fa è settare la PARQUET_FILE_SIZE ad un valore pari a quello del HDFS block size, per assicurarsi che le operazioni di I/O e trasfermento di dati in rete siano applicate effettuate solo per grandi quantità di dati.

All'interno di ogni data file, i dati appartenenti ad un insieme di righe sono ridisposti in modo che tutti i valori della prima colonna sono organizzati in un blocco contiguo, lo stesso vale per i valori della seconda colonna, e così via. Organizzare i valori della stessa colonna in questo modo consente ad Impala di utilizzare tecniche di compressione efficaci.

i vantaggi di prestazioni di questo approccio sono amplificati quando si utilizzano tabelle Parquet in combinazione con il partizionamento, in quanto Impala è in grado di scartare interi data file per alcune partizioni.

3.6.2 Partitioning

I dati relativi ad una tabella vengono salvati, di default, in un unica cartella. Il partizionamento è una tecnica per dividere fisicamente (sottocartelle) i dati durante l'operazione di inserimento,in base a valori appartenenti una o più colonne. In questo modo si ha un ottimizzazione delle performance riducendo le operazioni di I/O su disco e massimizzando la scalabilità delle query, velocizzando così l'esecuzione delle query che considerano le colonne usate per il partizionamento in quanto si esamina una porzione di dati molto limitata (singola directory della partizione) rispetto all'intera tabella.

Questa tecnica viene utilizzata quando:

- La tabella E' così grande che le lettura dell'intero data set richiederebbe un impraticabile quantità di tempo.
- La tabella E' soggetta, sempre o quasi, a query con condizioni sulle colonne di partizione.
- Le colonne di partizione hanno una ragionevole cardinalità
  - Se una colonna può assumere pochi valori, per esempio Maschio o Femmina, non guadagnerà molto in termini di efficienza in quanto si "elimina" solo il 50% dei dati letti dalla query
  - Se la colonna di partizione ha una granularità molto fine, tali da creare altrettante directories con una quantità minima di dati all'interno di esse, può causare problemi di efficienza in quanto il meccanismo di Hadoop ottimizato per l'esecuzione parallela e per trasmettere i dati in blocchi consistenti (multi-megabyte blocks) sarebbe inefficace.

- Il tipo di dato della colonna di partizione non ha alcun effetto significativo sul salvataggio dei dati perché i valori della colonna non sono sono salvati nei data files ma bensì sono rappresentati come stringhe (nome sotto cartella contenente i data files).

3.6.3 Considerazioni sulle prestazioni (JOIN)

Le queries che fanno utilizzo dell'operatore di JOIN richiedono una maggiore attenzione verso l'ottimizzazione delle performance rispetto a quelle che coinvolgono solo una tabella. La grandezza massima del risultato finale di una join query è il prodotto delle righe di tutte le tabelle coinvolte nel join. Nel caso di join fra tabelle con milioni o addirittura miliardi di righe, ogni mancata oppurtunità di filtrare l'insieme di dati, o altre inefficenze nella query, potrebbero portare ad una situazione di stallo o cancellazione della query stessa. In questo caso una delle tecniche più semplici che si possono adottare per migliorare le prestazioni è calcolare le statistiche per ogni tabella coinvolta nel join utilizzando l'istruzione "COMPUTE STATS". Impala ottimizzerà automaticamente la query basandosi sulla dimensione delle tabelle, il numero di valori distinti di ogni colonna, etc.. .

Nel caso in cui le statistiche non dovessero essere presenti, o se Impala ha scelto un ordine di join che non è il più efficiente, è possibile sovvrascrivere l'ordine scelto in modo automatico usando la keyword STRAIGHT_JOIN immediatamente la keyword SELECT. In questo modo Impala usa l'ordine con cuoi le tabelle appaiono nella query.

Il processo di scelta dell'ordine dei join Impala adotta queste strategie:

- sono presi in considerazione solo gli alberi di join left-deep
  - il primo join eseguito è sempre quello riguardante la tabella di grandezza maggiore; il query planner mette sempre come prima operazione la lettura della tabella di dimensioni maggiori.
- trova l'ordine di join più conveniente in termini di costo
- si basa fortemente sulle table e column statistics

Per quanto riguarda il tipo di join, Impala sceglie quale fra le due tecniche di join utilizzare basandosi sulle dimensioni assolute e relative delle tabelle coinvolte. Le uniche due tecniche a disposizione sono:

- **Broadcast**: Manda l'intero contenuto della tabella di destra (del join left-deep tree) a tutti i nodi coinvolti nel processare la query. Questa tecnica è scelta di default quando le statistiche sulla tabella non sono disponibili, quindi è necessario utilizzare lo hint

[BROADCAST] solo se Impala ha erroneamente scelto di usare un partitioned join. Tipicamente i join di tipo broadcast sono più efficienti nei casi in cui una tabella è notevolmente più piccola rispetto all'altra.

- **Partitioned**: Esegue l'operazione di join usando un'operazione di partizionamento che ,usando un algoritmo di hashing, suddivide le righe provenienti da entrambe le tabelle per poi inviare sottoinsiemi di righe ad altri nodi per la fase di processo in parallelo. Tipicamente i join di tipo partitioned sono più efficienti nel caso in cui entrambe le tabelle sono di grandi dimensione e aventi allo stesso tempo una simile dimensione.

E' possibile suggerire ad Impala quale tecnica utilizzare specificando un hint: [BROADCAST] o [SHUFFLE] (partitioned), immediatamente prima dell'operatore di JOIN.

3.6.4 HDFS Caching

Questa tecnica porta ad un miglioramento delle performance e maggior scalabilità in un ambiente di produzione dove le query di Impala e gli Hadoop jobs operano su quantità di dati molto maggiore della dimensione fisica della RAM presente nei vari nodi, il che rende impraticabile fare affidamento sulla cache sistema operativo Linux perché essa mantiene solo i data più recentemente utilizzati in memoria. Impala può usare la funzione di HDFS caching per utilizzare in modo efficace della RAM, in modo che le queries più frequenti possano trarre vantaggio dai dati presenti nella cache indipendentemente da quanti dati vengono elaborati in generale. La funzione di HDFS consente di definire un sottoinsieme di dati (letti più frequentemente) che verrà salvato permanentemente in memoria, garantendo così un accesso molto più veloce. Questa tecnica è adatta per tabelle o partizione lette frequentemente e con dimensione abbastanza piccola da poter essere salvata interamente all'interno della memoria HDFS cache.

3.6.5 Explain

L'istruzione EXPLAIN fornisce una descrizione dei passaggi logici che la query eseguirà, come per esempio, come il lavoro verrà distribuito tra i nodi e come i risultati intermedi saranno combinati per poi produrre il set di dati finale. E' possibile quindi determinare a priori se una query è stata ottimizzata per sfruttare le risorse a sua dispozione come memoria e tempo.

```
+-----------------------------------------------------------+
| Explain String                                            |
+-----------------------------------------------------------+
| Estimated Per-Host Requirements: Memory=884,64MB VCores=3 |
|                                                           |
| 18:MERGING-EXCHANGE [UNPARTITIONED]                       |
| |  order by: count(*) DESC                                |
| |                                                         |
| 10:SORT                                                   |
| |  order by: count(*) DESC                                |
| |                                                         |
| 17:AGGREGATE [FINALIZE]                                   |
| |  output: count:merge(*)                                 |
| |  group by: dte2.name                                    |
| |                                                         |
| 16:EXCHANGE [HASH(dte2.name)]                             |
| |                                                         |
| 09:AGGREGATE                                              |
| |  output: count(*)                                       |
| |  group by: dte2.name                                    |
| |                                                         |
| 08:HASH JOIN [INNER JOIN, BROADCAST]                      |
| |  hash predicates: ft2.entity_id = dte2.id               |
| |                                                         |
| |--15:EXCHANGE [BROADCAST]                                |
| |  |                                                      |
| |  04:SCAN HDFS [firbdmeng.dt_entity_parq dte2]           |
| |     partitions=1/1 files=2 size=97,03MB                 |
| |                                                         |
| 07:HASH JOIN [INNER JOIN, PARTITIONED]                    |
| |  hash predicates: ft1.clip_id = ft2.clip_id             |
| |  other predicates: ft1.entity_id != ft2.entity_id       |
| |                                                         |
| |--14:EXCHANGE [HASH(ft2.clip_id)]                        |
| |  |                                                      |
| |  03:SCAN HDFS [firbdmeng.ft_contain_parq ft2]           |
| |     partitions=1/1 files=14 size=2,97GB                 |
| |                                                         |
| 13:EXCHANGE [HASH(ft1.clip_id)]                           |
| |                                                         |
| 06:HASH JOIN [INNER JOIN, BROADCAST]                      |
| |  hash predicates: dte.topic_id = dtt.id                 |
| |                                                         |
| |--12:EXCHANGE [BROADCAST]                                |
| |  |                                                      |
| |  00:SCAN HDFS [firbdmeng.dt_topic_parq dtt]             |
| |     partitions=1/1 files=1 size=25,68KB                 |
| |     predicates: dtt.name = 'ukip'                       |
| |                                                         |
| 05:HASH JOIN [INNER JOIN, BROADCAST]                      |
| |  hash predicates: ft1.entity_id = dte.id                |
| |                                                         |
| |--11:EXCHANGE [BROADCAST]                                |
| |  |                                                      |
| |  01:SCAN HDFS [firbdmeng.dt_entity_parq dte]            |
| |     partitions=1/1 files=2 size=97,03MB                 |
| |                                                         |
| 02:SCAN HDFS [firbdmeng.ft_contain_parq ft1]              |
|    partitions=1/1 files=14 size=2,97GB                    |
+-----------------------------------------------------------+
```


FIGURA 3.4 – Piano di esecuzione della query COOCC con explain_level = 1

Nello specifico il piano di esecuzione (esempio nella figura 3.4) di una query mostra le seguenti informazioni:

- Lettura tabelle: per esempio nel punto 02 viene letta la tabella ft_contain_parq.
  - Grandezza tabella letta: sempre nel punto 2, la grandezza è di 2.97GB
  - Quanti nodi participano: (visibile con explain_level = 2)
  - Stima memoria utilizzata da ogni nodo: (visibile con explain_level = 2)
- strategia utilizizzata per scambiare i dati (EXCHANGE): nel punto 11 viene mandata in BROADCAST l'intera tabella *dt_entity_parq*, mentre nel punto 07 si può notare l'utilizzo di un join ***partitioned***.
- Tipi e strategie di join utilizzate: come si può facilemente notare nel punto 5 viene specificato il tipo di join (*INNER JOIN*) e la tecnica utilizzata (*BROADCAST*)
- Memoria totale e Vcores utilizzati da ogni nodo: visibile con explain_level = 2)

Capitolo 4

Caso di studio sulla politica europea

In questo capitolo verrà fatta una breve introduzione alla Social Business Intelligence e verrà presentato l'interessante caso di studio WebPolEU, il quale mira a mettere in luce come i diversi attori politici comunicano attraverso la rete e come i cittadini si relazionano questi ultimi.

4.1 Social Business Intelligence

Grazie alla diffusione degli smartphone ed alla popolarità dei social network, un enorme quantità di contenuti prodotta dagli utenti (user-generated content, UGC) è disponibile per essere analizzata. Questa enorme ricchezza di informazioni suscita grande interesse da parte dei decision makers perché può dare loro una percezione puntuale dello stato d'animo del mercato e aiutarli a spiegare fenomeni di business e della società. Social Business Intelligence (SBI) è una disciplina che mira a coniugare dati aziendali con UGC permettendo ai decisori di analizzare e migliorare il loro business basandosi sulle tendenze e umori ricavati da questi dati. Come la BI tradizionale, l'obiettivo della SBI è quello di consentire analisi flessibili ed efficaci ai decision makers con una scarsa esperienza in database. Nel contesto della SBI, la fonte primaria di contenuti proviene da clip testuali; un esempio comune di clip sono i contenuti pubblicati sui social media o articoli di giornali online. Un aspetto fondamentale è l'individuazione dei topic, ossia quei termini che individuano i concetti di interesse per l'analista. A seconda del dominio, un topic può essere il nome di un prodotto, di una persona, oppure un termine di uso comune che ha un significato particolare per l'utente. Attraverso la l'individuazine dei topic e la definizione di una gerarchia di aggregazione degli stessi si aggiunge un'importante dimensione di analisi degli UGC focalizzata sul dominio dell'utente. L'Estrazione di informazioni, utili per i decision makers, da UGC testuali richiede una fase di scansione della fonte durante la quale si cercano

clip riguardanti un argomento di interesse. Le clip una saranno poi arricchite cercando di ricavare quante più informazioni possibili dal testo grezzo. Un esempio di attività di arricchimento è l'identificazione di informazioni strutturali come: autore, sentiment e argomenti correlati. Un processo di SBI nel suo complesso è composto da tutte le fasi che vanno dalla scansione del web alla analisi dei risultati.

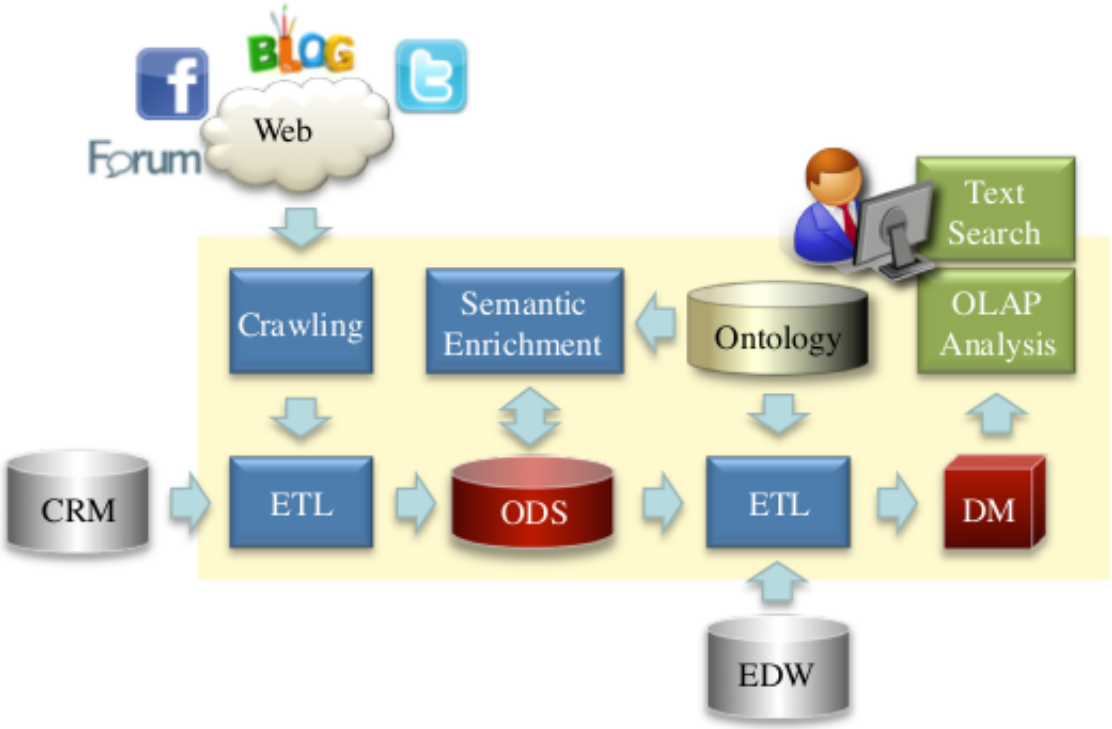

FIGURA 4.1 – Architettura di un generico processo di Social Business Intelligence

In particolare, questa tesi effettua un test delle prestazioni su un progetto (WebPolEU) volta ad indagare il legame tra politica e mezzi di comunicazione sociale. Il progetto utilizza UGC in tre lingue diverse (italiano, inglese e tedesco) ed è focalizzato sul periodo delle elezioni europee del 2014. Nel suo complesso si compone di circa 10 milioni di clip provenienti da vari fonti web come testate giornalistiche, social network e blog.

Nella figura 4.3 sono mostrate le cardinalità delle principali tabelle di interesse; comprendono solamente i dati relativi alla lingua inglese **[14]** .

	Cardinalità
Topic	432
Clip	3.275.193
Entity	2.902.942
Occorrenze Topic	23.398.601
Occorrenze Topic	519.446.526
relazioni semantiche	23.837.474

FIGURA 4.2 – Principali dati presi in esame con rispettive cardinalità

4.2 Architettura processo di SBI

Di seguito sono descritte brevemente le parti principali che compongono un processo di SBI, il cui obiettivo è mettere a disposizione le informazioni ai decision makers tramite cubi multidimensionali con contenuti di tipo testuale.

The Operational Data Store (ODS) stores all the relevant data about clips, their authors, and their source channels; the ODS also represents all the topics within the subject area and their relationships.

- Operational Data Store (ODS): memorizza tutti i dati rilevanti le clip, gli autori e le sorgenti; the ODS also represents all the topics within the subject area and their relationships.
- Il Data Mart (DM) memorizza i dati in una serie di cubi multidimensionali che supportano il processo decisionale.
- Crawling svolge una serie di scansioni basate su parole chiave volte a recuperare clip inerenti ad un determinato argomento. Questa attività può essere lanciata su tutto il web (definendo solo il punto di partenza) oppure un inseme di fonti scelte (forums, siti web, social netoworks).
- Semantic Enrichment estrae dall'ODS le informazioni semantiche nascoste nei testi delle clip. A seconda della tecnologia adottata tali informazioni possono includere le singole frasi contenute in una clip, il/i suo/i argomenti , le relazioni sintattiche e

semantiche tra parole, o il sentimento relativa a una intera frase o per ogni argomento singolo che contiene.

- Il processo di ETL trasforma periodicamente l'output semi-strutturato del crawler in una forma strutturata e lo carica sulle ODS, possibilmente in combinazione con i dati estratti dal CRM aziendale. Poi estrae i dati riguardanti le clip ed i topic dall'ODS, li integra con i dati aziendali estratti dal EDW, e li carica nel DM.

- L'Analisi OLAP consente ai decision makers di esplorare la UGC da diverse prospettive ed controllare il sentimento sociale generale.

4.3 WebPolEU

Il progetto si propone di studiare la relazione fra politica e social media in prospettiva comparata dal punto di vista sia dei cittadini, sia degli attori politici. Attraverso lo studio dei processi di alfabetizzazione digitale, della partecipazione politica online e delle discussioni politiche sui social media, la ricerca valuterà l'inclusività, la rappresentatività e la qualità della discussione e della partecipazione politica su internet in Germania, Gran Bretagna e Italia. In particolare, mediante lo studio della comunicazione online degli attori politici e di come i cittadini si relazionano con i leader di partito e amministratori locali sui social media, il progetto chiarirà se l'offerta di comunicazione politica in rete incontri o meno la domanda dei cittadini e se i media digitali offrano opportunità efficaci di garantire il controllo dei governati sui governanti. Lo studio comparato di Germania, Gran Bretagna e Italia consentirà di controllare ipotesi relative agli effetti di fattori sistemici e istituzionali e di comprendere lo sviluppo della politica online in sistemi politici rilevanti nella definizione del ruolo globale dell'Europa.

SBI è utilizzato nel progetto come tecnologia abilitante per l'analisi gli UGC generati in Germania, Italia e UK durante un periodo temporale che spazia da Marzo 2014 a Maggio 2014 (le elezioni del Parlamento Europeo si sono tenute tra il 22 e il 25 Maggio 2014). Nell'architettura che è stata adottata, gli argomenti (topics) e le tassonomie correlate sono stati definiti attraverso Protégé; è stato usato Brandwatch per il keyword-based crawling (scansione basata su parole chiave), Talend per l'ETL, SyN Semantic Center fornito da SyNTHEMA per l'arricchimento semantico (in particolare, per l'etichettamento di ciascuna clip con il relativo sentiment), Oracle per la memorizzazione dei dati dell'ODS e il DM (data mart), e MongoDB per il salvataggio del database dei documenti adibito alla ricerca testuale (full text search). Lo strumento CASE Indyco è stato utilizzato per la progettazione del Data Mart ed è stata sviluppata ad-hoc un'interfaccia OLAP e una

dashboard tramite JavaScript. Alcuni esempi di indagini raccolte durante la fase di macro-analisi sono: “Quali sono i principali temi trattati dai diversi politici, partiti e famiglie di partiti nel contesto delle Elezioni Europee?” “Come cambia l’opinione dei cittadini riguardo l’Eurozona e l’integrazione europea nelle diverse nazioni?” “Come si trasformano nel tempo i temi più discussi?” “C’è un qualche processo di comunicazione tra le nazionalità che va avanti, ad esempio, riguardo temi e argomenti specificatamente correlati a nazioni diventate soggetto di discussione anche in altre nazioni?”.
Tra i temi emersi durante questa fase, si menzionano le istituzioni EU, la società civile, la cultura e la democrazia. Campioni di argomenti trovati in queste tematiche sono la Corte Europea della Giustizia, il sindacato (trade union), la scuola e l’Euroscettico. Per quanti riguarda le sorgenti di selezione, Brandwatch possiede già alcune proprie fonti di base per il crawling. Alla prima iterazione di progettazione, i nostri esperti hanno indicato un insieme di circa 100 siti web rilevanti per le tre nazioni coinvolte in modo da poter controllare che essi fossero già inclusi nelle fonti di base e quindi analizzati. Nelle seguenti iterazioni, un elevato numero di fonti che generata troppe clip off-topic e quindi non rilevanti è stato progressivamente eliminato dalle scansioni. Dall’ontolgia mostrata in figura 4.3 sono stati estrapolati tutti i topic di interesse come per esempio i nomi dei politici candidati alle elezioni e gli argomenti più discussi. È stata poi costruita una gerarchia di topic per raffinare l’aggregazione sugli UGC raccolti (ad esempio, è possibile effettuare un’aggregare sulle clip in cui compare un partito considerando anche le clip in cui compaiono i politici di tale partito)**[15]** .

	Cardinalità
Topic	432
Clip	3.275.193
Entity	2.902.942
Occorrenze Topic	23.398.601
Occorrenze Topic	519.446.526
relazioni semantiche	23.837.474

FIGURA 4.3 – Una rappresentazione UML dell’ontologia Topic per il progetto WebPolEU

4.4 Progettazione della base di dati

La tecnologia da utilizzare per supportare i processi descritta nell'immagine 4.1 potrebbe richiedere cambiamenti a seconda di un specifico progetto.
Come si può notare dallo schema, sempre presentato in figura 4.1, il processo di acquisizione dei dati provenienti dai social media incomincia con un'attività di crawling. Questa attività produce come output i primi dati grezzi; in particolare questi dati sono composti da un testo (post di facebook, status di twetter, articolo blog) che viene denominato clip (testo di interesse) e da i sui metadati (autore, data, canale di comunicazione). I dati vengono poi dati in input ad un altra attività denominata ETL (Extract, Transform, Load) la quale effettua una trasformazione dei dati. L'output della fase di ETL viene salvato nel ODS.Nello specifico nella figura 4.4 possiamo notare come un Autore è associato ad una o più clip, mentre una clip è associata ad un singolo autore.Nel caso di più autori di una stessa clip viene utilizzato il formato JSON per creare un gruppo di autori, avendo così sempre un autore per clip.

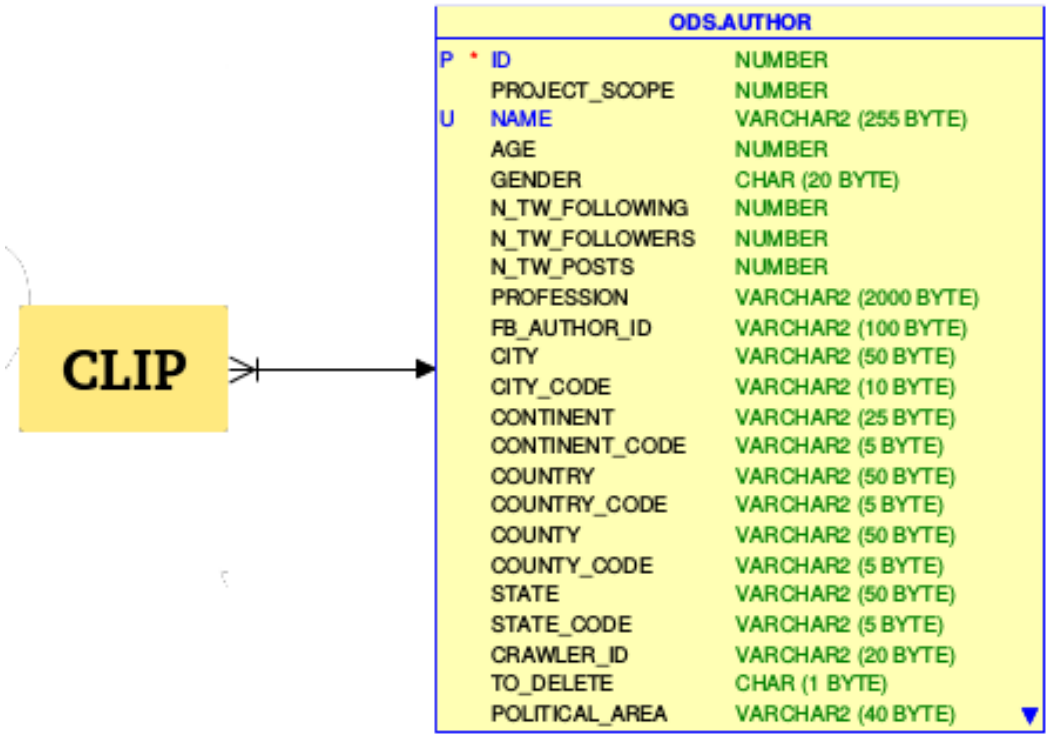

FIGURA 4.4 – porzione semplificata dello schema logico dell'ODS

A questo punto viene effettuato un arricchimento dei dati tramite l'attività di semantic enrichment tramite un processo di NLP (natural language processing). Questo processo in particolare estrapola dal testo le frasi e vengono individuate le parole (entity) che occorrono ed eventuali relazioni presenti fra due entity eventualmente accompagnate da un qualifier. Successivamente tramite un procedimento manuale si creano due tabelle topic (argomenti di interesse) e alias (sinonimi e declinazioni di un topic). Sempre manualmente, tramite un processo di matching, verranno collegate le entity agli alias (sinonimi e declinazioni). Infine si esegue un ultimo processo di ETL durante il quale, per motivi tecnici legati all'analisi, i dati vengono denormalizzati e salvati sul Data mart, su il quale si svolgeranno le analisi OLAP.

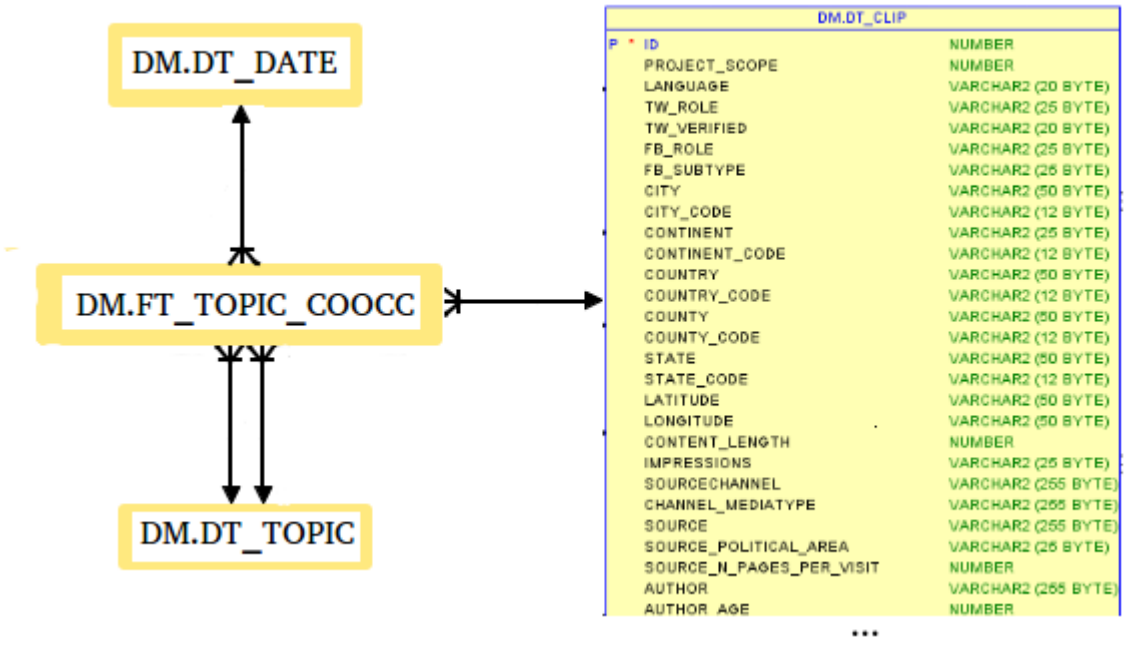

FIGURA 4.5 – orzione semplificata dello schema logico del Data mart

Come è possibile osservare in figura 4.5 lo shema è stato sottoposto ad un processo di de-normalizzazione. Infatti i dati di alcune tabelle come per esempio *Author* sono stati riportati dentro le corrispettive clip.Nella stessa figura è rappresentato anche uno dei tre fatti (*FT_TOPIC_COOCC*) che ci permette di analizzare le coocorrenze tra due topic.

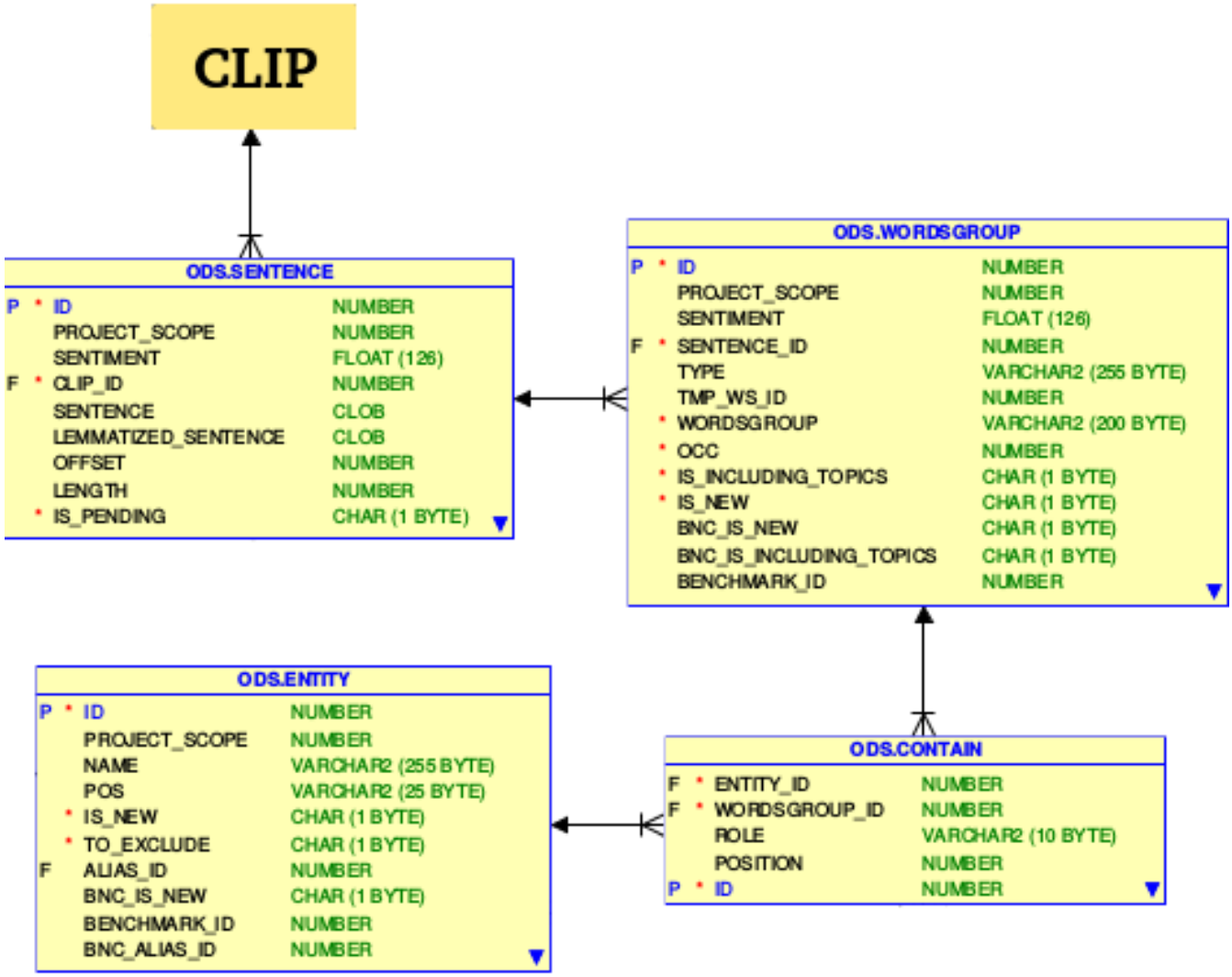

FIGURA 4.6 – Schema logico dell'ODS relativo alla memorizzazione delle relazioni fra le entity all'interno delle clip

Lo schema logico mette in evidenza la gerarchia delle relazioni presenti in una clip. Ciascuna clip viene suddivisa in sentences, le quali a loro volta contengono gruppi di parole aventi un aventi una particolare relazione.

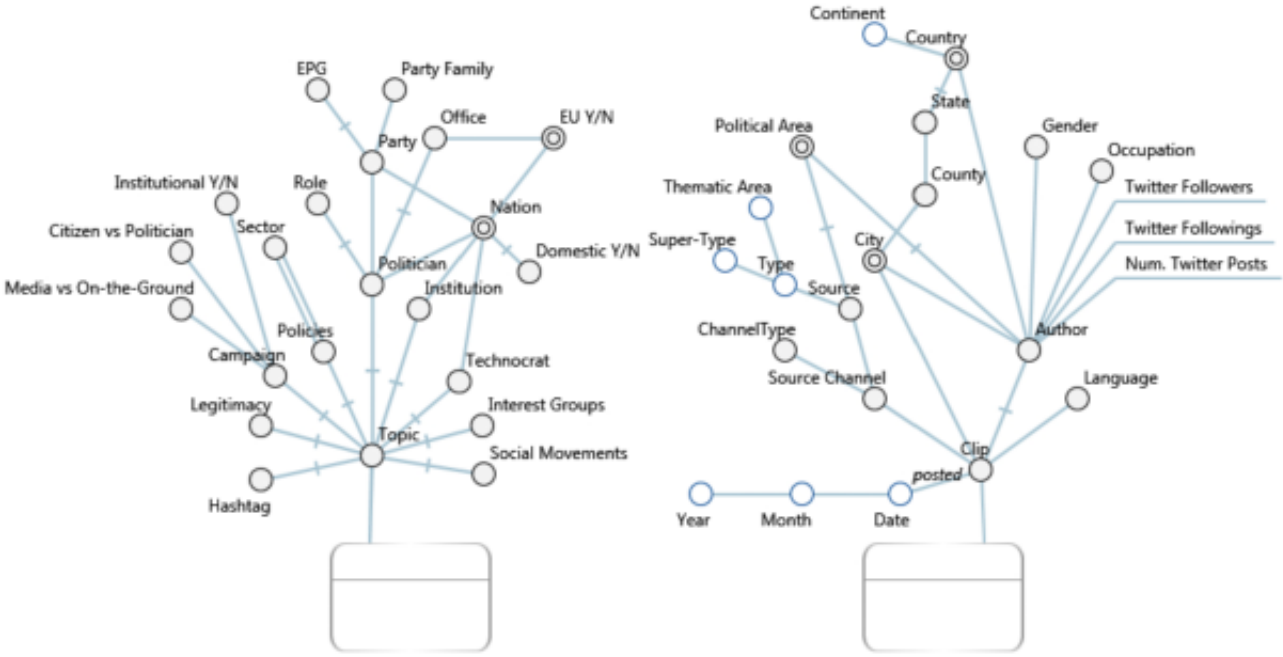

FIGURA 4.7 – Schemi DFM dove sono rappresentate le gerarchie di Topic e Clip

Per abilitare l'aggregazione delle clip in base agli argomenti nel front-end OLAP, le classi nell'ontologia di dominio sono state riorganizzate in una gerarchia di topic.

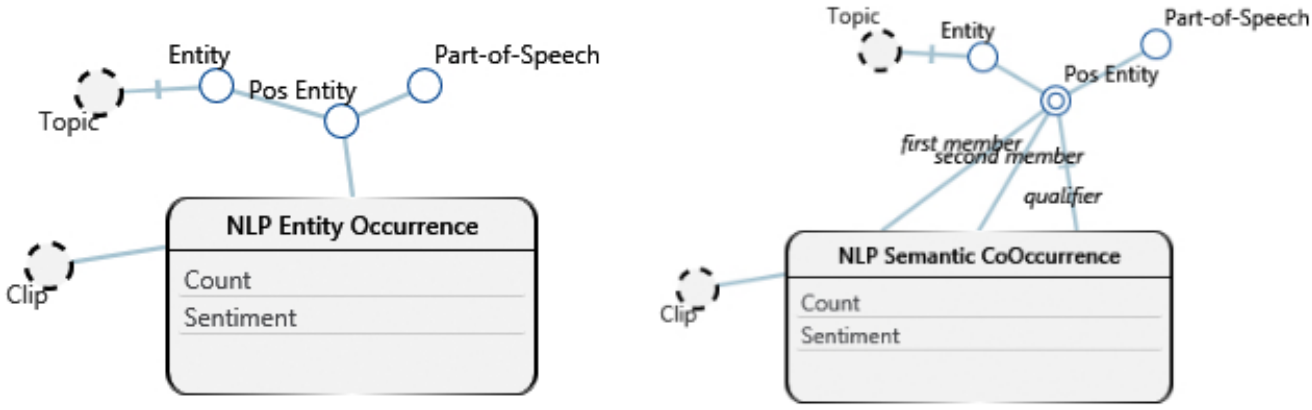

FIGURA 4.8 – la figura mostra i due principali fatti di interesse del sistema

Nel data warehouse ci sono due fatti principali: le occorrenze delle entity all'interno delle clip; le occorrenze di relazioni tra due o più entity all'interno di una clip.Entrambi hanno le misure *count* e *sentiment*.

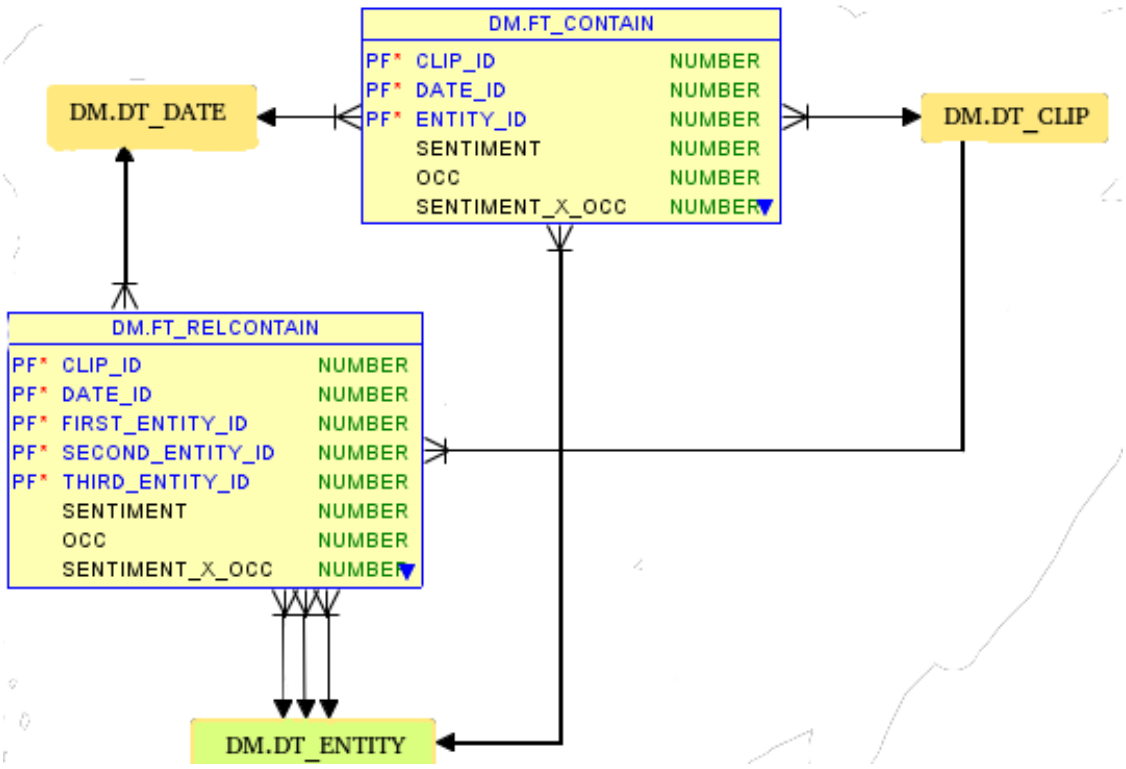

FIGURA 4.9 – Implementazione dei due fatti principali sul data mart

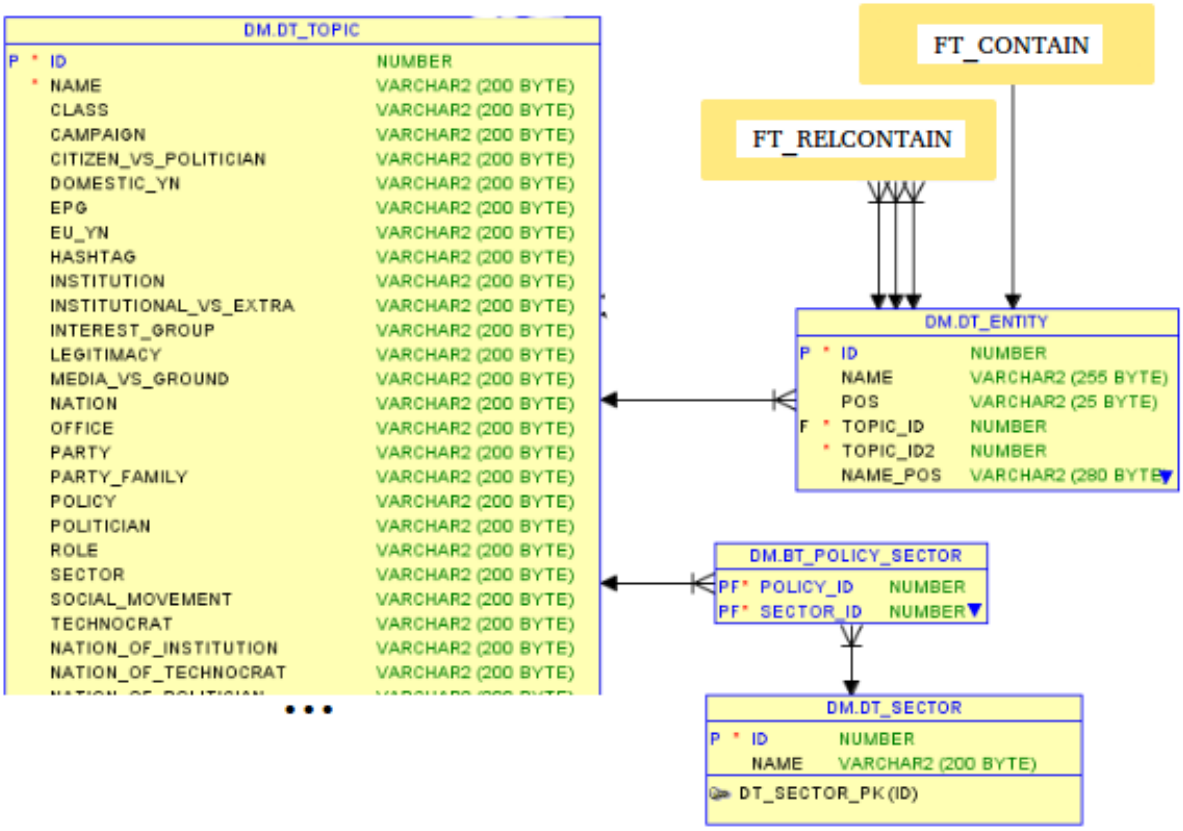

FIGURA 4.10 – Implementazione dei Topic sul data mart

Nel data mart i due fatti principali sopra citati sono implementati come mostrato in figura 4.9 e 4.10. In particolare possiamo notare la mancanza degli alias (non è di interesse una aggregazione per alias; sono stati utilizzati solo per creare un collegamento fra topic e entity) e una relazione many-to-many fra topic e sector valida solo per le policy.

4.5 Porting su Impala

Un porting da un qualsiasi database relazionale verso Impala, o per essere più precisi, verso HDFS è molto semplice da realizzare. Infatti è sufficiente eseguire un operazione di export di tutte le tabelle che compongono la base di dati in formato csv, o comunque

utilizzando un separatore a piacere, e trasferire i file dump risultanti sull'HDFS tramite un semplice comando lanciato dalla shell di Impala:

```
!hdfs dfs -put  /<path_locale_qualsiasi_macchina_cluster>/file.csv
          "/user/hive/warehouse/<database>/<nome_tabella>";
```

Ricordiamo, come già detto nel capitolo 2.2 Progettazione fisica dello schema, che una tabella è identificata da cartelle e file ($\langle database/table\_name \rangle$ / datafile.file_format) presenti nel HDFS. A questo punto è necessario fare una precisazione. Una delle principali qualità di Hadoop è "schema on read" , ovvero non è necessario fare una ampia pianificazione come i dati sono disposti, ed è possibile successivamente modificare e perfezionare tali schemi senza alcun tipo di problematica. Questo modello si scontra con i tradizionali modelli SQL dove il comando CREATE TABLE definisce un preciso layout per una specifica tabella, e i dati, quando caricati, sono riorganizzati in modo dipendente dal layout. Impala utilizza le seguenti politiche:

- Impala permette di definire lo schema per files che sono già presenti nel sistema e , senza alcun cambiamento necessario a livello di contenuto dei file, permette di interrogarli immediatamente.
- Impala non richiede vincoli di lunghezza per le stringhe.
- Utilizzando il più semplice dei formati, il formato di txt, gli attributi possono essere interpretati come stringa, numero, timestamps o qualsiasi altro tipo ammesso.
- Impala in caso di mancata corrispondenza fra numero di attributi dello schema e numero di attributi effettivi nel file (x colonne schema, y colonne file), Impala ignora i campi extra o ritorna NULL in caso il campo non sia presente.
- E' possibile cambiare lo schema di una tabella aggiungendo nuove colonne, rimuovendole oppure cambiare il tipo di dato degli attributi in ogni momento. I file interessati non saranno in alcun modo modificati.

I benefici di questo approccio sono: più flessibilità, meno tempo e costo speso per convertire i dati in altri formati, e soprattutto la possibilità di progettare schemi senza particolari nozioni databasistiche. Di seguito verrà descritto come importare un intero database da Oracle. Come specificato nel paragrafo precedente è necessario fornire ad Impala una interpretazione (schema) del file/tabella. Per fare questo si usa la dichiarazione SQL CREATE TABLE, grazie alla quale è possibile specificare un separatore che spazia i vari campi; in questo caso, essendo il formato scelto CSV, lo spaziatore è ','.

Impala sarà ora già in grado di effettuare delle query su questa tabella importata ma, come abbiamo visto in uno dei capitoli precedenti, una delle best practices è scegliere il formato di file più adatto e performante. Il cambio di formato può essere fatto anch'esso tramite la dichiarazione CREATE TABLE nel seguente modo:

LISTATO 4.1 – Esempio di trasformazione di formato

```
1  create table [tabella_parquet] stored as parquet
2                      as select * [tabella_csv]
```

Così facendo verrà creata una nuova tabella nel formato desiderato mantenendo lo schema della tabella selezionata. A questo punto, si può procedere applicando questi passaggi a tutte le tabelle facenti parte del nostro database di interesse, e una volta ultimato questo processo il porting del database è completato ed è possibile procedere con le interrogazioni.È doveroso sottolineare come in vari libri e nel tutorial della stessa Cloudera sia proposta l'utilizzo della tecnologia Sqoop per importare dei dati.Durante la fasi di porting dei dati di WebPolEu è stato fatto un tentativo di utilizzo di questa tecnologia ma senza successo.

Capitolo 5

Test delle performance di Impala

In questo capitolo verrà effettuato un'analisi delle prestazioni sullo schema del Data Mart del progetto WebPolEU. In particolare la tecnologia utilizzata è Impala. Le prestazioni di quest'ultima vengono comparate in un primo luogo con le prestazioni del RDBMS di Oracle il quale è in esecuzione su una sola macchina, mentre in un secondo tempo vengono confrontate le prestazioni con una tecnologia, simile ad Impala, per interrogazioni su basi di dati non relazionali distribuite. Nello specifico verranno messe a confronto le prestazioni di Impala con quelle di ElastiSearch, un search engine open source con capacità full text.

5.1 Architettura Hardware

In questa sezione viene fatta una panoramica sulle due tipologie di architettura utilizzate:

- Singola macchina
- Cluster distribuito

L'enviroment di Oracle è, come detto in precedenza, in esecuzione su di una singola macchina con i seguenti componenti hardware:

- CPU: Intel i7-4790, 4 core, 8 threads, 3.6 Ghz
- RAM: 32 GB
- HARD-DRIVE: 2 x 2 TB HDD, 7200 rpm

Le altre due tecnologie invece, sono in esecuzione su di un cluster composto da 7 nodi, ciascuno di essei avente le seguenti caratteristiche hardware:

- CPU: Intel i7-4790, 4 core, 8 threads, 3.6 Ghz
- RAM: 32 GB
- HARD-DRIVE: 2 x 2 TB HDD

5.2 Query OLAP

Per fornire un confronto accurato fra le varie tecnologie, è stato creato un insieme di query divise per complessità e scopo. In particolare:

- Gruppo 1 – variazione selettività: le query di questo gruppo sommano le varie le occorrenze di una parola e viene effettuato un raggruppamento per classe e sorgente. Questo gruppo è composto da 4 query aventi selettività via via decrescenti ed impostate sulla dimensione geografica:

occ	class	channel_mediatype
5674410	policy	facebook
2618597	policy	news
2016376	campaign	facebook
1240707	institution	facebook
1220733	office	facebook
769204	campaign	news
643354	policy	forum
542452	institution	news

FIGURA 5.1 – Output (SELECT_1)

- Gruppo 2 – variazione numero predicati: questo gruppo (PRED_1, .. , PRED_6) è a sua volta diviso in due parti in ciascuna delle quali vengono aggiunti predicati di selezione (da 1 fino a 3 predicati). Nella prima parte vengono contate le occorrenze di ogni topic, mentre nella seconda parte si contano le coocorrenze raggruppate per topics.

occ	topic_name
773402	election campaign
759663	government
455172	minister
447326	vote
362019	common agricultural policy
351916	europe

FIGURA 5.2 – Output (parte 1, query PRED_1)

cooc	topic_name_1	topic_name_2
210511	election campaign	vote
199576	ukip	vote
185958	vote	election campaign
167722	vote	ukip
162004	election campaign	government
161449	ukip	election campaign
157212	government	election campaign

FIGURA 5.3 – Output (parte 2, query PRED_4)

- Gruppo 3 – variazione numero risultati: in questo gruppo (AGGREG1, .. , AGGREG5) vengono contate le occorrenze per raggruppate per schieramento politico e attributi della clip (diversi per ogni query).

occ	party_family	continent
11898329	Unclassified	North America
8941025	Unclassified	Europe
688217	radical right	Europe
469935	christian-democrat & conservative	Europe
390980	christian-democrat & conservative	North America
138387	radical right	North America

FIGURA 5.4 – Output (AGGREG1)

- Gruppo 4 – variazione numero join: in questo gruppo (N_JOIN1, .. , N_JOIN5) vengono contate le occorrenze delle entity che compaiono con "UKIP" il 26/05/14 variando i predicati di join.

occ	second_entity_id	second_entity_name
6341	134793	vote
3494	137895	be
1712	141382	win
897	135321	scotland
883	135507	get
879	130837	vote

FIGURA 5.5 – Output (N_JOIN5)

- Gruppo 5 – variazione numero colonne: in questo gruppo (N_COL1, .. , N_COL3) vengono contate le occorrenze raggruppate per politico e aggiungendo via via predicati di raggruppamento sulle misure della clip/topic.

occ	politician	city
12540596	Unclassified	United States
113917	david william donald cameron	United States
28662	nigel paul farage	United States
15435	nicholas william peter clegg	United States
13772	angela dorothea merkel	United States
11232	alex salmond	United States
11202	edward samuel miliband	United States

FIGURA 5.6 – Output (N_COL1)

- Gruppo 6 – ordinamento e coocorrenze: in 2 (ORDER_1, ORDER_2) delle 3 query facenti parte di questo gruppo viene effettuato un ordinamento sulla somma delle occorrenze con e senza filtro.

count(*)	name
193644	route
132200	not
124812	vote
66088	party
60638	people
60521	say
57293	election
53415	get
51603	just
48505	more
48482	that
46990	nigel paul farage

FIGURA 5.7 – Output (COOCC)

5.3 Confronto con Oracle

Il primo confronto è stato eseguito considerando tutti i gruppi di query descritti in precedenza. In particolare si è scelto di considerare due configurazioni di Impala e una di oracle:

- 7 demoni: configurazione dove tutti i sette demoni (impalad) di Impala sono in stato di run; ovvero viene sfruttata al massimo la capacità di calcolo del cluster.
- 1 demoni: configurazione che comprende solamente un demone (impalad) in esecuziones; ovvero è come se venisse usata soltanto una delle sette macchine del cluster.
- Oracle: normale istanza di oracle in esecuzione sulla macchina descritta in precedenza.

Lanciate le query si sono ottenuti i seguenti risultati mostrati in tablella e nel grafico:

	tempo esecuzione in secondi		
Nome Query	Impala (7 demoni)	Impala (1 demone)	Oracle
SELETT1	34	140	39
SELETT2	21	76	31
SELETT3	6	17	15
SELETT4	4	10	35
N_PRED1	19	79	26
N_PRED2	8	22	19
N_PRED3	6	11	16
N_PRED4	7	27	125
N_PRED5	2	4	77
N_PRED6	2	4	287

FIGURA 5.8 – Porzione tabella con i tempi relativi ai primi due gruppi di query

	tempo esecuzione in secondi		
Nome Query	Impala (7 demoni)	Impala (1 demone)	Oracle
AGGREG1	29	140	38
AGGREG2	30	138	41
AGGREG3	30	139	43
AGGREG4	30	139	43
AGGREG5	32	153	45
N_JOIN1	1	2	92
N_JOIN2	1	2	88
N_JOIN3	1	2	89
N_JOIN4	3	2	88
N_JOIN5	7	3	89
N_COL1	22	99	32
N_COL2	22	100	32
N_COL3	24	109	40
ORDER_1	35	101	246
ORDER_2	3	12	84
COOCC	155	83	654

FIGURA 5.9 – Porzione tabella con i tempi relativi agli ultimi 4 gruppi di query

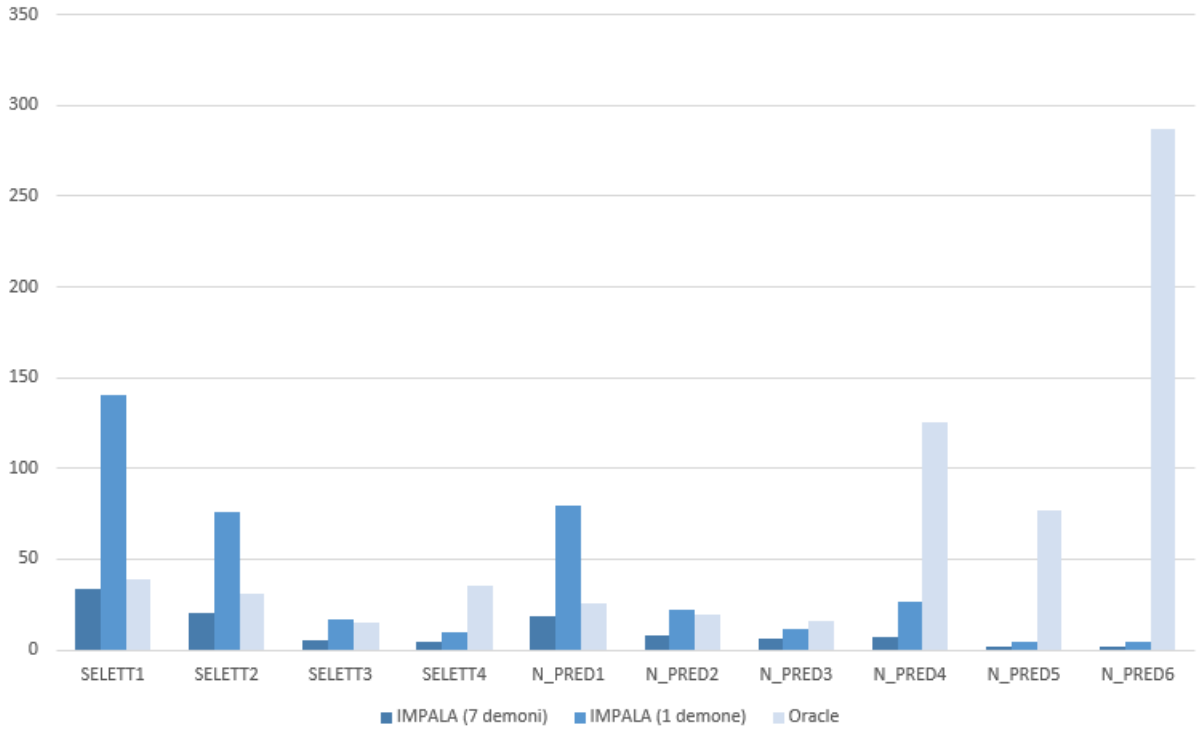

FIGURA 5.10 – Grafico con i tempi dei primi due gruppi di query della comparazione fra 7 demoni - 1 demone - Oracle

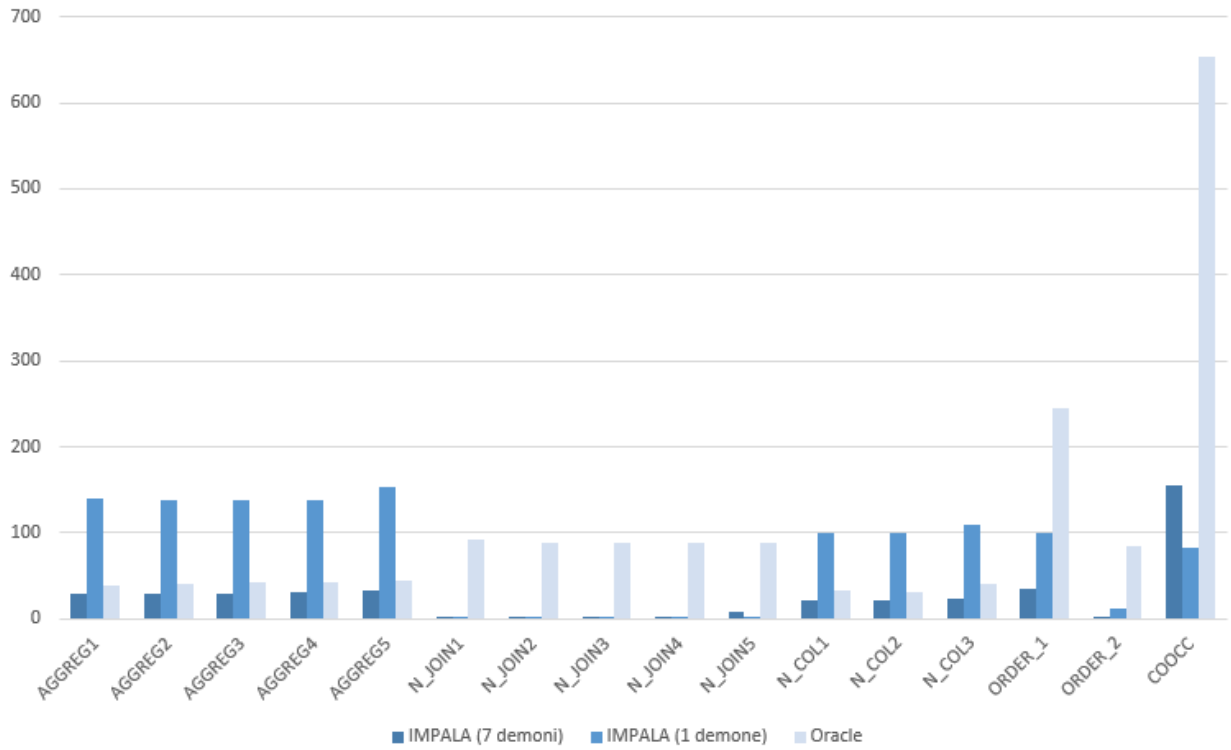

FIGURA 5.11 – Grafico con i tempi gli ultimi 4 gruppi di query della comparazione fra 7 demoni - 1 demone - Oracle

Analizzando i risultati si possono fare le seguenti considerazioni:

- Nel gruppo 1 la configurazione con sette demoni scala molto bene al diminuire della selettività ed ottiene sempre tempi migliori delle altre due configurazioni
- Nel gruppo 2, in particolare nella seconda parte, notiamo che la variazione dei tempi di Oracle è discordante rispetto al grado di selettività; ciò è dovuto alle limitazione negli istogrammi che impediscono la generazione del piano di esecuzione ideale.
- Nel gruppo 3, la configurazione con sette demoni di Impala ottiene sempre i tempi migliori; notiamo anche che in tutte e tre le configurazioni non ci grosse variazioni di tempo.
- Nel gruppo 4 entrambe le configurazioni di Impala ottengono tempi pressoché vicini, mentre allo stesso tempo Oracle segna tempi più grandi di un ordine di grandezza.

- Nel gruppo 5 si può notare che gli ordinamenti incidono molto nella configurazione con un demone di Impala, mentre le restanti due ottengono tempi simili.
- Nell'ultimo gruppo si può notare che Oracle ottiene tempi di gran lunga superiori rispetto alle altre due configurazioni; Inoltre è opportuno notare che il tempo segnato dalla configurazione Impala (1 demone) è per la prima volta minore rispetto a quella con 7 demoni. Ciò è dovuto ad un errore di Impala nella scelta della strategia di join utilizzata. In particolare in quest'ultima query viene eseguito un join fra due FT_CONTAIN; Impala in questi casi deve decidere se usare un broadcast join o un partitioned join. Si è visto, eseguendo il comando EXPLAIN, come Impala sbagli a scegliere la strategia di join scegliendo broadcast. Così facendo viene mandata l'intera tabella (oltre 3Gb) a tutti i nodi facenti parte del cluster e ovviamente il trasferimento di quest'ultima influisce di molto sul tempo di esecuzione della query.

Approfondendo questo ultimo punto si è tentato di migliorare i tempi di esecuzione di alcune query.

	BROADCAST	SHUFFLE
SELETT1	34	29
JOIN5	7	2
COOCC	155	25

FIGURA 5.12 – Comparazione di alcune query cambiando la strategia di join - solo con configurazione Impala (7 demoni)

Come è possibile vedere in tabella, la prima colonna rappresenta le query eseguite con strategia broadcast (strategia scelta di defalut dal query planner), mentre la seconda le query eseguite forzando, tramite hint, la strategia partitioned. La prima query comprende un join fra ft_contain e dt_clip; quest'ultima avendo una grandezza moderata (125 MB) fa si che il miglioramento sia esiguo.Nel secondo caso (JOIN 5) si verifica un caso analogo in quanto dt_entity ha una grandezza di solamente 97 MB. Nell'ultimo caso invece il miglioramento è nettamente visibile perché si verifica il caso in cui si dovrebbe applicare la strategia partitioned, ovvero: un join fra due tabelle di di grandi dimensioni (in questo caso 2.97 GB) e di simile dimensione (in questo caso uguale). Infatti applicato lo hint [SHUFFLE] il tempo è diminuito di un fattore 5x.

Come è possibile vedere in tabella, nelle prime due query i tempi sono diminuiti ottenendo risultati di poco migliori. Consigliando al query planner, tramite l'utilizzo di un hint ([SHUFFLE]), la strategia da adottare per quel join problematico si è ottenuto un tempo 5 volte migliore rispetto a quello precedente.

Contemporaneamente a questa tesi ne è stata scritta una similare prendendo come soggetto una tecnologia simile denominata ElasticSearch. Quest'ultima sfrutta anch'essa il cluster, ma con l'unica differenza di non sfruttare il framework Hadoop. Esiste una versione di ElasticSearch per sistemi Hadoop ma la versione installata sul cluster è standalone, cioè si gestisce in modo del tutto autonomo.
Si è deciso quindi di effettuare un confronto fra le due tecnologie. Prima di commentare i tempi della tabella sottostante è opportuno specificare cosa sia la configurazione "Impala Filtro (Unclassified)". Mentre su Impala e Oracle le entity che non hanno un topic sono comunque associate ad un topic fittizio chiamato Unclassified, nella modellazione scelta per ElasticSearch si è deciso di non associare tali entity ad alcun topic. Questo significa che un raggruppamento su tutti i topic va a considerare un insieme di fatti minore rispetto alla modellazione originale. Per far fronte a questa differenza alcune query sono state modificate inserendo un predicato di selezione che - ove opportuno - escluda le entity che fanno riferimento al topic Unclassified. In particolare abbiamo considerato le seguenti query mostrate in tabella e nel grafico.

	tempo esecuzione in secondi		
Nome Query	Impala (7 demoni)	Impala (filtro Unclassified)	ElasticSearch
SELETT1	34	9	16
SELETT2	21	3	16
SELETT3	6	5	16
SELETT4	4	5	16
N_PRED1	19	3	20
N_PRED2	8	3	20
N_PRED3	6	2	19
AGGREG1	29	9	17
AGGREG2	30	4	15
AGGREG3	30	4	22
AGGREG4	30	4	19
AGGREG5	32	5	20
N_COL1	22	4	19
N_COL2	22	4	21
N_COL3	24	4	26
ORDER_1	35	-	38
ORDER_2	3	-	-
COOCC	155	-	-

FIGURA 5.13 – Tabella con i tempi (secondi) della comparazione fra Impala (7 demoni) - Impala Filtro (Unclassified) - ElasticSearch

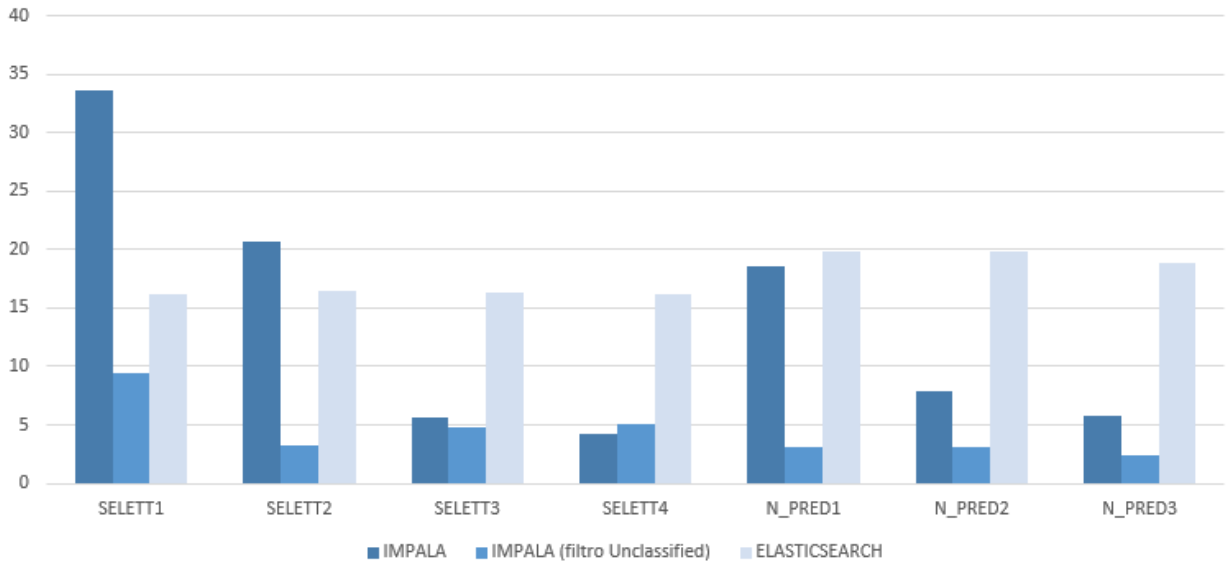

FIGURA 5.14 – Risultati del primi due gruppi di query della tabella 5.13

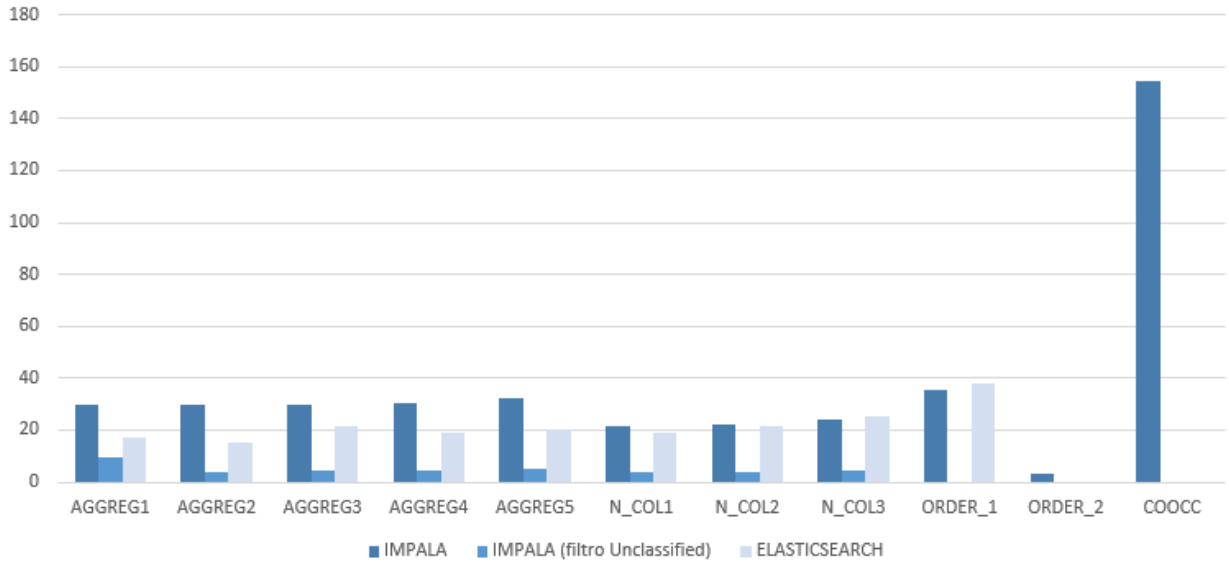

FIGURA 5.15 – Risultati degli ultimi tre gruppi di query della tabella 5.13

Notiamo inanzi tutto che la configurazione di Impala con l'aggiunta del predicato di "esclusione", ottenie sempre tempi minori rispetto alle altre due configurazioni. Scendendo nel dettaglio è possibile notare che ElasticSearch è l'unica a non scalare nel primo e

nel secondo gruppo di query, e nel complesso ottiene sempre tempi più o meno costanti. Questo fenomeno è probabilmente dovuto al fatto che ElasticSearch è in grado di eseguire ricerche sui dati in modo molto efficente mentre trova molto oneroso l'operazione di raggruppamento sui dati selezionati. Dalla tabella notiamo anche come alcuni tempi siano mancanti. Nel caso della query COOC mancano alcuni tempi: nel caso di ElasticSearch è dovuto alla impossibilità del linguaggio di effettuare questo tipo di query. Per quanto riguarda la configurazione di Impala con filtro, il tempo è assente in quanto quella query non necessita di tale filtro. Questa considerazione vale anche per le query ORDER_1 e ORDER_2. Eseguendo la query ORDER_2 con ElasticSearch il sistema non è riuscito a completarne l'esecuzione a causa di una mancanza di memoria disponibile.

5.4 Conclusioni

Con l'avvento del Web 2.0 chiunque abbia accesso ad Internet può creare contenuti condivisibili attraverso la rete stessa. Questa mole di dati ha poi attirato l'attenzione delle aziende che in pochi anni hanno dovuto adottare nuove soluzioni per la memorizzazione e l'analisi dei dati. Hadoop è una delle soluzioni per la gestione dei Big Data più popolare in questo periodo, grazie anche al fatto di essere una tecnologia open source. In particolare questo framework offre un supporto a molteplici software per l'analisi dei dati, tra i quali ci sono ElasticSearch ed Impala. Per ora queste soluzioni sono state adottate da aziende con uno stretto bisogno di queste tecnologie, cioè non possono più affidarsi ai sistemi relazionali tradizionali. Per la maggior parte delle aziende queste tecnologie vanno ad affiancarsi ai sistemi fino ad ora utilizzati consentendo di dare un supporto alle decisioni in tempi più brevi, ed in mondo che si muove sempre più velocemente il fattore tempo gioca sicuramente un ruolo cruciale. Per esempio questo tipo di strumenti può essere di grande aiuto nell'settore della Social Businness Intelligence dove i contenuti sono generati dagli utenti ma sono spesso caratterizzati dal fatto di contenere opinioni di qualsiasi genere. Un azienda in grado di sfruttare queste tecnologie per memorizzare e analizzare questi dati in tempo reale sicuramente otterrà grandi benefici per il proprio business. In un ambito simile rientra il progetto WebPolEU per il quale è stata raccolta e analizzata una discreta quantità di dati relativi alla politica.Per effettuare un confronto prestazionale con un classico RDBMS Oracle si è scelta la tecnologia di Cloudera, in particolare il query engine Impala. Questa tecnologia relativamente giovane è ancora in sviluppo e ogni anno migliora le sue prestazioni. Nonostante la sua giovane età alcune aziende come Amazon e Oracle offrono questa tecnologia come soluzione SQL-on-Hadoop. Le principali caratteristiche: la facilità di utilizzo, per quanto riguarda il porting e l'interrogazioni dei dati, e l'alta scalabilità sono essenziali per realizzare sistemi di analisi real time. Ritornando

nell'ambito del progetto di questa questa, la soluzione Impala si è dimostrata fino a otto volte più veloce rispetto alla soluzione su cubo relazionale. Al crescere dei dati processati si è notato come la tradizionale soluzione relazionale risulti essere nettamente inefficente rispetto alle altre due non relazionali che hanno sensibilmente ridotto i tempi di risposta. Tenuto conto dei risultati ottenuti e guardando al processo di SBI nel suo complessoi, si può concludere che per quanto riguarda le analisi OLAP le tecnologie relazionali fino ad ora utilizzate sono destinate ad essere soppiantate da quelle non relazionali. I sistemi tradizionali relazionale dovranno comunque essere affiancati a quelli non relazionali con la funzione di supporto alla fase che precede l'analisi.

Bibliografia

[1] Progettezione e sviluppo di una soluzione Hadoop per il calcolo di big data analytics, Tesi di Laurea Magistrale di Francesca Marchi, Anno accademico 2013-2014

[2] https en.wikipedia.orgwikiBig_data

[3] http://www.html.it/pag/50728/architettura-di-hadoop/

[4] http://www.html.it/pag/50111/introduzione-ad-hadoop/

[5] https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html

[6] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[7] https://cwiki.apache.org/confluence/display/Hive/Design

[8] http://hadooptalk.com/awesome-introduction-of-parquet/

[9] http://www.cidrdb.org/cidr2015/Papers/CIDR15_Paper28.pdf

[10] http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala.html

[11] http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/PDF/cloudera-impala.pdf

[12] Learning Cloudera Impala by Avkash Chauhan (Packt, 2014)

[13] http://www.slideshare.net/cloudera/query-compilation-in-impala

[14] Gallinucci E., Golfarelli M., Rizzi S., Advanced topic modeling for social business intelligence. Information Systems (2015).

[15] http://webpoleu.altervista.org/it/