ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

PROGETTAZIONE E SVILUPPO DI UNA SOLUZIONE HADOOP PER IL CALCOLO DI BIG DATA ANALYTICS

Tesi in
Sistemi Informativi E Business Intelligence

Relatore:
Prof. MATTEO GOLFARELLI

Presentata da:
FRANCESCA MARCHI

Co-relatori:
PAOLO RODEGHIERO
ALESSANDRO VENNI

ANNO ACCADEMICO 2013-2014
SESSIONE III

Ai miei genitori che mi hanno
sempre sostenuto in tutte le mie scelte
A Francesco per il suo amore
A tutti coloro che hanno sempre creduto in me

Indice

Introduzione

Il presente elaborato ha come oggetto la progettazione e lo sviluppo di una soluzione Hadoop per il Calcolo di Big Data Analytics. Nell'ambito del progetto di monitoraggio dei bottle cooler, le necessità emerse dall'elaborazione di dati in continua crescita, ha richiesto lo sviluppo di una soluzione in grado di sostituire le tradizionali tecniche di ETL, non più sufficienti per l'elaborazione di Big Data. L'obiettivo del presente elaborato consiste nel valutare e confrontare le perfomance di elaborazione ottenute, da un lato, dal flusso di ETL tradizionale, e dall'altro dalla soluzione Hadoop implementata sulla base del framework MapReduce.

Nel primo capitolo viene svolta una panoramica generale sulle origini e l'evoluzione del fenomeno dei BigData e come questi vengono definiti da chi li osserva. Viene inoltre fornita una descrizione dettagliata sulle 3 caratteristiche principali (le cosidette 3V) che caratterizzano i Big Data e i contesti in cui si sviluppano. Una volta individuati e compresa la potenzialità che contraddistingue questa tipologia di dati, occorre comprendere come questi devono essere gestiti e come possono essere sfruttati. A tal fine, viene illustrato il ciclo di vita che caratterizza il contesto dei Big Data, soffermandosi e mettendo il luce, per ognuna delle fasi del ciclo, le tecnologie principali che attualmente vengono utilizzate in materia. Infine, è stata svolta una panoramica sullo stato dell'arte in materia di Big Data, facendo riferimento sia al contesto Italiano che a quello Estero, in particolare negli Stati Uniti. Tale panoramica è stata volta per mostrare come tale fenomeno, in alcuni casi non si è realmente manifestato e invece, in altri casi si è sviluppato e affermato.

Il secondo capitolo si concentra principalmente sulla descrizione e le caratteristiche del framework di calcolo Hadoop. Capire come avvengono i meccanismi e come è organizzata l'architettura di Hadoop, è importante al fine di estrapolarne i concetti utili per lo sviluppo di una soluzione Ha-

doop. Nel dettaglio sono state descritte le architetture dei due componenti principali, che stanno alla base del framework, ovvero HDFS e MapReduce. Relativamente ad HDFS sono stati mostrati i meccanismi di scrittura e lettura, soffermandosi sulle strategie di replicazione e di ottimizzazione adottate per aumentare le perfomance di computazione. Invece, relativamente a MapReduce, l'attenzione si è focalizzata sull'architettura di YARN e su come MapReduce gestisce il flusso di dati durante l'esecuzione di un Job MapReduce. A conclusione del capitolo, è stata svolta prima, una panoramica sull'insieme delle componenti principali, che fanno parte dell'ecosistema di Hadoop, poi, si è passati ad analizzare le distribuzioni Hadoop più note, soffermandosi sui vantaggi e gli svantaggi legati all'utilizzo di un cluster in cloud piuttosto che un cluster in-house.

Il terzo e ultimo capitolo si focalizza interamente sul caso di studio trattato. Inizialmente è stata svolta una panoramica generale sul contesto progettuale nel quale ci si trova a sviluppare la soluzione Hadoop. Dopo aver descritto il dominio del progetto e gli obiettivi che si intende raggiungere, si è proseguito il capitolo suddividendolo, come il progetto stesso, in sue macro aree: Fase 1 e Fase 2. La prima fase riguarda l'analisi e lo sviluppo di una soluzione di Business Intelligence, tramite la quale è stato possibile svolgere lo studio di fattibilità, che ha permesso di stabilire se gli obiettivi prefissati potessero in qualche modo essere soddisfatti o meno. Confermata la fattibilità del progetto, si è passati alla seconda fase del progetto, dove è stata svolta la progettazione e lo sviluppo della soluzione Big Data, mediante tecnologia Hadoop. La soluzione è stata progettata in modo tale che, all'aumentare della quantità di dati, fosse in grado di svolgere le stesse elaborazioni svolte dalla prima soluzione, mantenendo costanti i tempi e le perfomance di computazione. La soluzione Hadoop prodotta doveva dimostrarsi in grado di superare le limitazioni che si sono presentate nei tradizionali sistemi di ETL, all'aumento della quantità di dati. A conclusione del capitolo e della tesi sono stati svolti alcuni test. Questi sono stati fatti applicando le due distinte soluzioni ai diversi data set, contenenti quantità di dati via via crescenti. Questi test hanno permesso, da un lato, di svolgere un confronto tra le due soluzioni, dall'altro ha permesso di osservare quali sono stati gli impatti di computazione sulle singole soluzioni all'aumentare dei dati.

Capitolo 1

Big Data

1.1 L'evoluzione dei dati e delle tecniche di analisi

Negli anni i dati, la loro gestione, ma soprattutto i processi di analisi volti a trasformare i dati in informazioni hanno subito un'evoluzione. Negli anni Sessanta, le uniche tecnologie disponibili consentivano di raccogliere su supporti magnetici i dati relativi ai processi aziendali. Le uniche analisi che potevano essere svolte erano statiche e si limitavano alla sola estrazione dei dati raccolti. Con l'avvento dei database relazionali e del linguaggio SQL, negli anni Ottanta, l'analisi dei dati assume una certa dinamicità. Infatti l'SQL consente di estrarre in maniera semplice i dati, sia in modo aggregato, sia a livello di massimo dettaglio. Le attività di analisi avvengono su basi di dati operazionali, ovvero sistemi di tipo OLTP (*On Line Transaction Processing*) caratterizzati e ottimizzati prevalentemente per attività di tipo transazionale (inserimento, cancellazione e modifiche dei dati), piuttosto che per attività di lettura e di analisi di grandi quantità di record. La maggior parte dei sistemi OLTP offrono una limitata, se non mancante, storicizzazione dei dati, e molto spesso, anche in presenza di dati storici, risulta complesso ricostruire la situazione dei dati nel passsato. Inoltre vi sono sempre più contesti in cui sono presenti numerose applicazioni che non condividono la stessa sorgente, ma i dati sono replicati e manipolati da ciascun software, non garantendo così l'uniformità e la coerenza dei dati. La difficoltà nell'effettuare l'analisi dei dati direttamente sulle fonti operazionali ha

portato, a partire dagli anni Novanta, alla creazione di database progettati appositamente per l'integrazione dei dati e l'analisi. Nascono così i *Data Warehouse*, database che contengono dati integrati, consistenti e certificati relativi ai processi di business delle aziende. Questi costituiscono il punto di partenza per le attività analitiche dei sistemi di *Business Intelligence* (BI). La *Business Intelligence* è un insieme di modelli, metodi, processi, persone e strumenti che rendono possibile la raccolta, e la riorganizzazione dei dati generati da un'azienda. Attraverso elaborazioni, analisi o aggregazioni, ne permette la trasformazione in informazioni, la conservazione, la reperibilità e la presentazione in una forma semplice, flessibile ed efficiente, tale da costituire un supporto per i processi decisionali. Tuttavia si tratta sempre di una visione storica, che consente soltanto una valutazione a consuntvo di ciò che è accaduto nel passato, oppure di ciò che sta accadendo ora. Più di recente, a partire dai primi anni Duemila, è emersa la necessità di effettuare analisi previsionali, per anticipare gli eventi e ottenere un vantaggio di business. Emergono così le tecniche di *Data Mining* che consentono di "scavare" nei dati ed estrarre informazioni, pattern e relazioni non immediatamente identificabili e non note a priori. A partire dal 2010 si evidenziate ulteriori diverse evoluzioni nell'ambito dell'analisi dei dati e della BI:

- *Business Analytics*: tecnologie, metodi e applicazioni che utilizzano modelli matematici e statistici per l'analisi dei dati e per il data mining. Dotati spesso di funzionalità di analisi visuale dei dati.

- *Mobile BI e Reporting*: progettazione e produzione di applicazioni per la navigazione dei dati e per la visualizzazione dei report su supporti "mobili" come smartphone e tablet.

- *Self-Service BI*: software di semplice utilizzo che mettono in grado l'utente finale di costruire report, analisi e addirittura nuovi modelli dati.

- *Cloud Computing*: insieme di tecnologie che permettono di offrire risorse hardware e software come servizi su Internet. Queste risorse possono essere utilizzate nell'analisi dei dati e nella BI.

- *Big Data*: insieme di tecnolgie e fattori evolutivi volti all'analisi complessa di grandi moli di dati eterogenei e/o destrutturati.

Nei paragrafi successivi verrà trattato più nel dettaglio il contesto dei Big Data, descrivendone le caratteristiche, le tecnologie e gli strumenti che li caratterizzano.

1.2 Cosa è Big Data

Accade sempre più spesso che le aziende abbiano la necessitá di analizzare i dati, da sempre prodotti in grandi quantità, ma non immagazzinati a causa della mancanza di strumenti di analisi in grado di elaborare tale mole di dati.

Nonostante le aziende abbiano la possibilità di accedere a questa mole di dati, con i tradizionali strumenti messi a disposizione, non sono in gradi di "estrapolare" valore da questi, perchè molto spesso si presentano nella loro versione più grezza oppure in formati semistrutturati o addirittura non strutturati. La percentuale di dati che il business può processare sta calando molto velocemente. Da indagini [2] è emerso che oltre la metà delle organizzazioni non hanno pieno accesso alle informazioni di cui hanno bisogno per svolgere il proprio lavoro.

Nel corso degli anni si sono resi disponibili dati che, per tipologie e per numerosità, hanno contribuito a far nascere il fenomeno dei Big Data. Il termine Big Data viene applicato a dati e informazioni che non possono essere processati o analizzati utilizzando processi e strumenti tradizionali. I big data rappresentano uno dei fattori evolutivi nel mondo dell'analisi dei dati e della Business Intelligence.

1.2.1 Dai Dati Operazionali ai Big Data

Le basi di dati operazionali variano a seconda della tipologia di azienda che si sta considerando. Alcuni esempi di fonti operazionali potrebbero essere applicativi per la:

- gestione della produzione;
- gestione degli acquisti;
- gestione degli ordini e delle consegne;
- contabilità;

- gestione del personale;
- gestione dei clienti.

Tradizionalmente le basi di dati operazionali risiedono su database relazionali RDBMS (*Relational Database Management System*). I database relazionali sono progettati utilizzando tecniche di normalizzazione che facilitano le attività transazionali di inserimento, modifica e cancellazione dei dati, ottimizzandone le prestazioni. I database normalizzati però non sono adatti alle analisi, per questo motivo si sfruttano appositi database, detti *Data Warehouse* che permettono di ottimizzare le performance di interrogazione. I dati provenienti dalle fonti operazionali vengono aggiunti in maniera incrementale all'interno dei *Data Warehouse*, garantendo così la storicizzazione dei dati. Questi tipi di sistemi però, in presenza di fonti che generano grandi moli di dati, sono caratterizzati da una storicizzazione molto onerosa che nel tempo potrebbe risultare ingestibile dal punto di vista delle risorse e dei costi.

I dati operazionali, a seconda del business, possono assumere volumi rilevanti. Si prenda come esempio l'ambito bancario, considerando solamente una parte del patrimonio dei dati della banca, dove per ogni cliente, vengono registrati i saldi giornalieri dei conti e le movimentazioni. Inoltre la velocità con cui i dati vengono prodotti è un aspetto critico che deve essere considerato parallelamente alla mole di dati mantenuti in memoria. Accanto alle fonti strettamente legate al business vi possono essere sistemi, più vicini alla produzione, che generano enormi quantità di dati. Solitamente ci si riferisce a sistemi *DCS* (*Distributed Control System*), sistemi computerizzati utilizzati per il controllo di impianti industriali. I componenti distribuiti sull'impianto generano dati mediante sensori legati al componente stesso; le rilevazioni dei dati possono avvenire a intervalli temporali molto piccoli e ciò, assieme alla possibile presenza di migliaia di sensori, porta a produrre una mole molto elevata di valori grezzi da gestire. I dati provenienti da sensori non sono gli unici che possono assumere dimensioni ragguardevoli, anche le apparecchiature scientifiche di misurazione e analisi e le apparecchiature mediche e diagnostiche sono potenzialmente in grado di generare una quantità molto elevata di dati.

Le tecniche legate ai database relazionali, molto spesso, non riescono a tenere testa alla quantità di dati e la velocità con cui essi sono prodotti dai sistemi di misurazione o dal business. Le limitazioni degli RDBMS

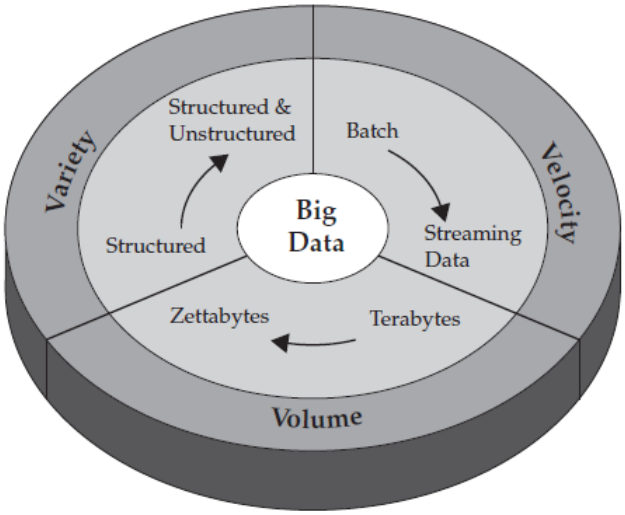


Figura 1.1: IBM: Caratteristiche dei Big Data [2]

possono essere superate se si considerano i cosidetti database *historian*. Questi sistemi, da un lato comportano un notevole risparmio in termini di spazio e consentono un recupero efficiente dei dati, dall'altro risultano essere poco efficaci per svolgere analisi complesse sulle serie storiche, come analisi statistiche avanzate o ricerca di pattern attraverso tecniche di data mining.

Le problematiche citate richiedono tecnologie diverse dagli RDBMS e dagli historian, tecnologie che consentano, senza investimenti proibitivi, di ottenere potenza di calcolo e scalabilità.

1.3 Le Caratteristiche dei Big Data

I big data rappresentano tutti quei dati che possono essere disponibili in enormi volumi, possono presentarsi con formati semistrutturati o addirittura destrutturati e possono essere prodotti con estrema velocità. Volume, varietà e velocità (*Volume*, *variety*, *velocity*) sono i fattori che caratterizzano i big data.

1.3.1 Volume

Uno degli aspetti che caratterizzano i big data è la loro quantità. Dati generati dall'utente attraverso gli strumenti del Web 2.0, sistemi gestionali, oppure dati generati automaticamente da macchine (sensori, strumenti scientifici) possono assumere volumi rilevanti, non più gestibili con strumenti di database tradizionali.

Una valanga di dati viene generata ogni giorno, solo Twitter e Facebook generano più di 7 TeraByte (TB) di dati ogni giorno. Il volume di dati che ad oggi si sta memorizzando sta esplodendo. Se ci si ferma a pensare non c'è da meravigliarsi se progressivamente si sta annegando in questo mare di dati; oggigiorno se si ha la possibilità di tenere traccia o registrare qualcosa lo si fà, basti pensare ad un semplice smartphone e all'insieme di informazioni e dati che ogni giorno viene prodotto da ogni suo singolo sensore. Uno dei principi chiave per operare con i big data è la memorizzazione di tutti i dati grezzi/originali, indipendetemente dal loro immediato utilizzo. Ogni operazione di pulizia o scarto potrebbe portare all'eliminazione di informazioni utili in futuro. È evidente come, così facendo, l'ammontare di dati da mantenere nei sistemi diventi estremamente elevato.

In certi casi, si potrebbe pensare di utilizzare dei normali RDBMS per memorizzare i dati, ma questo presuppone di investire cifre elevatissime sia per lo storage, si per la capacità di calcolo necessaria a elaborare tale mole di dati. Tali investimenti potrebbero rivelarsi non giustificabili alla luce dei risultati ottenuti in termini di performance. Al crescere del volume i dati non possono più essere immagazzinati utilizzando i sistemi tradizionali. Esistono soluzioni basate su architetture hardware *MPP* (*Massive Parallel Processing*) utilizzate in ambito data warehousing, che però non sono adatte a far fronte a un'altra caratteristica dei big data: l'eterogeneità dei formati e la presenza di dati destrutturati. Esistono pertanto soluzioni e tecnologie alternative che permettono di gestire e analizzare al meglio l'intera mole di dati, con l'obiettivo di ottenere informazioni a supporto del business che si sta considerando. Tra le tecnologie open source, la più diffusa e utilizzata è Apache Hadoop, grazie alla sua capacità di processare grandi quantità di dati a costi contenuti.

1.3.2 Varietà

Con l'esplosione dei sensori, degli smartphone, degli strumenti del Web 2.0 e dei social network i dati si sono "complicati", ovvero non presentano più una struttura predefinita e quindi non sono più riconducibili ad uno schema tabellare, ma possono presentare un formato semistrutturato o destrutturato, non più rapplesentabile in modo efficiente in un database relazionale. La diversità di formati e, spesso, l'assenza di una struttura sono la seconda possibile caratteristica dei big data. La varietà perciò, ha portato un drastico cambiamento all'interno dei processi analitici; si è passati dai tradizionali dati strutturati a dati semistruttirato e/o destrutturati che non possono essere gestiti e processati dai tradizionali strumenti analitici. Per il salvataggio di dati semistrutturati, molto spesso la scelta ricade su cosidetti database NoSql, che forniscono i maccanismi adatti a organizzare i dati ma, allo stesso tempo, non impongono uno schema predefinito, come invece avviene per i database relazionali; infatti vengono anche detti *schemaless database*. La mancanza di schema, che negli RDBMS deve essere progettata prima dello sviluppo, consente di adattarsi alla variabilità dei dati.

Se ci si ferma ad osservare ciò che ci circonda è possibile notare che solo il 20% dei dati presentano un formato strutturato adatto agli schemi dei database relazioni; ben l'80% dei dati del mondo sono destrutturati o, nella migliore delle ipotesi, semistrutturati. Per cui le imprese, per poter sfruttare l'opportunità dei big data, devono essere in grado di gestire e analizzare tutti i tipi di dati che si presentano, sia relazionali che non relazionali.

1.3.3 Velocità

Non solo la varietà e il volume dei dati che vengono memorizzati sta cambiando, anche la velocità con cui i dati vengono generati sta cambiando e deve essere gestita. La velocità con cui i nuovi dati si rendono disponibili è il terzo fattore con cui è possibile identificare i big data. Oltre al volume, anche la velocità con cui le fonti generano nuovi elementi rende necessario l'utilizzo di strumenti in grado di tenerne il passo. La sfida per le aziende consiste nella capacità di sfruttare i dati provenienti ad alte velocità con altrettanta rapidità, estrapolando le informazioni utili per il business, minimizzando i tempi di elaborazione. A volte, essere in vantaggio rispetto alla concorrenza, significa identificare un problema, una tendenza o un'opportunità in pochi secondi, prima di chiunque altro; quindi, per poter trovare informazioni utili,

le aziende devono possedere gli strumenti ed essere in grado di analizzare tali dati "quasi" in tempo reale. Le tecnologie di riferimento per la gestione di questo aspetto dei big data sono chiamate *streaming data* o *complex event processing* (CEP), descritte nei paragrafi successivi.

1.4 Il ciclo di vita dei Big Data

Una volta riconosciuti i big data, è necessario pensare a come gestirli e dove memorizzarli, inoltre la molteplicità e la varietà delle fonti da cui possono provenire i dati hanno portato alla nascita di tecnologie apposite in grado di supportare il volume, la varietà e la velocità che caratterizzano i big data. Le tradizionali tecnologie sono risultate poco adatte e poco performanti al fine di gestire e analizzare la moltitudine di dati a cui ci si trova di fronte. Riunire in modo funzionale dati strutturati e non strutturati, proveniente da fonti più disparate, può aiutare le organizzazione a ridurre i costi, migliorare le relazioni con i clienti, sviluppare nuovi prodotti o soluzioni per la collettività, accelerare e sincronizzare le consegne, formulare e rispondere a richieste più approfondite, migliorare e semplificare il processo decisionale, ecc. Come per i tradizionali sistemi di gestione dei dati, anche in questo contesto è possibile parlare di **Ciclo di Vita dei Big Data**. La Figura1.2 mostra le tre principali fasi che caratterizzano il ciclo di vita dei Big Data, e per ognuna di essa viene mostrato l'insieme degli strumenti associabili alle varie fasi del ciclo. La figura non mostra tutti i possibili software ad oggi in commercio, ma raccoglie i più conosciuti e utilizzati dalle aziende.

1.4.1 Acquisizione dei Big Data

Diversamente dai sistemi tradizionali, la tipologia e la quantità di fonti diverse da cui possono provenire i dati sono molteplici. Prima del fenomeno dei big data si aveva a che fare con fonti operazionali costituite prevalentemente da database relazioni, e quindi caratterizzate da dati strutturati. Ora, la presenza dei social network, dei sensori di controllo, del web e di qualsiasi altro dispositivo elettronico che genera masse d'informazioni, spesso semistrutturate e destrutturate, portano ad avere una moltitudine di fonti dati diverse da acquisire, che utilizzano tecniche diverse. L'acquisizione dei big data può avvenire, a seconda del tipo di fonte, attraverso differenti mezzi, che è possibile suddividuere in quattro categorie:

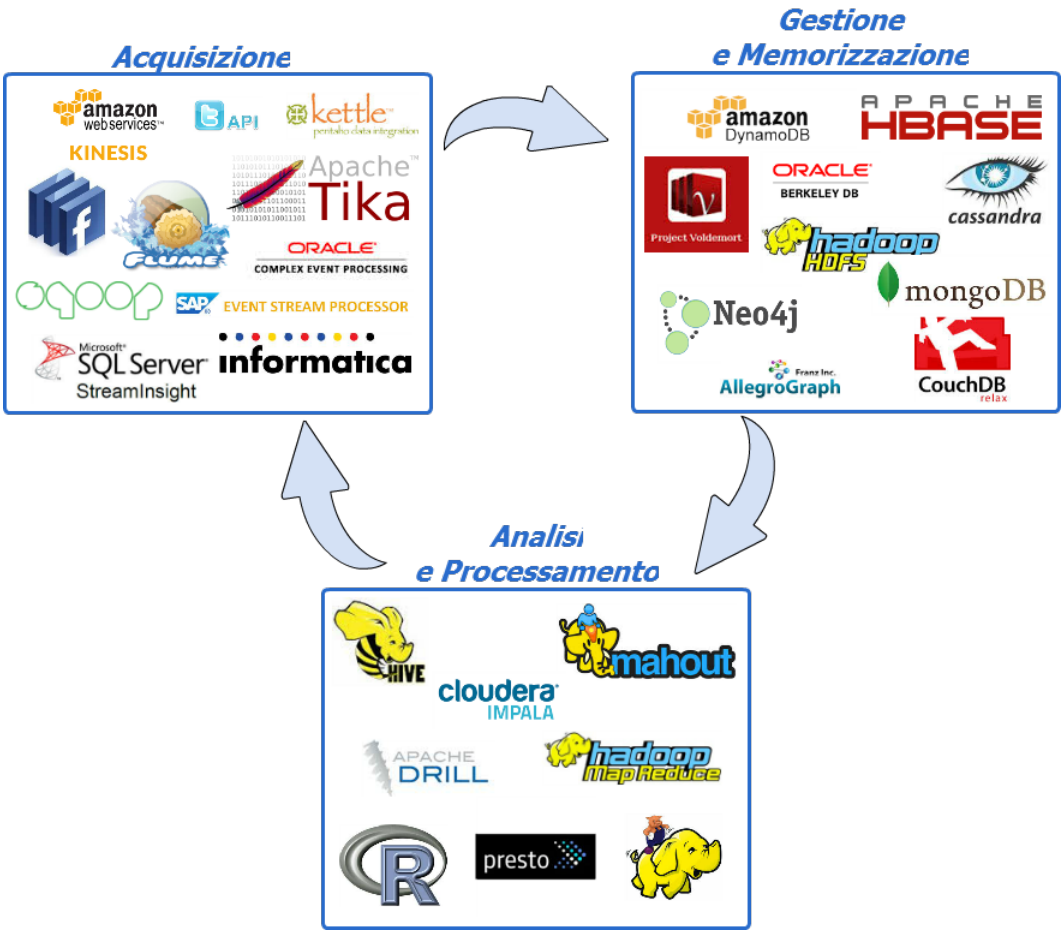


Figura 1.2: Big Data Life Cycle

- API (Application Programming Interface)
- Strumenti di ETL
- Software di Web Scraping
- Lettura di stream di dati

Di seguito, per ognuna delle suddette categorie, viene fornita una breve descrizione, facendo riferimento alle principali tecnologie esistenti nel settore.

API (Application Programming Interface)

Le Application Programming Interface sono protocolli utilizzati come interfaccia di comunicazione tra componenti software. In questa categoria rientrano sia i dati provenienti dalle fonti operazionali, sia i dati provenienti dal Web, in particolare dai social network.

Due importanti esempi sono le **Twitter API** e le **Graph API di Facebook**. Esse permettono di interfacciarsi con le piattarforme social, esaminando nel primo caso tutti i tweet legati a particolari argomenti d'interesse e, nel secondo caso, tutti i contenuti pubblicitari che corrispondono ai criteri di ricerca. Anche i motori di ricerca, come **Google**, **Yahoo!** e **Microsoft Bing** hanno messo a disposizione API che consentono l'interfacciamento con alcuni dei propri servizi come Google Maps, Windows Azure Market Place. Una funzionalità interessante messa a disposizione da Yahoo! è YQL (Yahoo Query Language), un linguaggio SQL-like per l'interrogazione di numerose sorgenti dati, in grado di restituire URL utilizzabili in un qualsiasi browser o applicazione.

Strumenti ETL

Gli strumenti di ETL, utilizzati nei contesti di Business Intelligence e Data Warehousing, permettono di svolgere i processi di estrazione, trasformazione e caricamento dei dati, provenienti da fonti operazionali e destinati ai sistemi di Data Warehouse. Molti strumenti di ETL, ad oggi, sono già attrezzati per importare i dati, dai formati più disparati, nel sistema di gestione dei big data. Un esempio è **Penthao Kettle**, strumento di ETL utilizzato principalmente negli scenari tradizionali di Data Warehouse, ma con

l'evoluzione dei big data, ha integrato componenti in grado di interfacciarsi con le principali tipologie di fonti big data (esempio Hive, S3, etc...).

Recentemente Apache ha rilasciato **Sqoop**, uno strumento open source progettato per estrarre e trasferire in modo efficiente dati strutturati da database relazionali (RDBMS) a HDFS (oppure Hive e HBase). Dualmente, una volta che i dati caricati su Hadoop sono stati elaborati, Sqoop è in grado di estrarre i dati da HDFS ed esportarli su database strutturati esterni. Ad oggi, Sqoop è uno dei progetti di punta di Apache; dispone di un'interfaccia a linea di comando attraverso la quale è possibile eseguire le istruzioni per la movimentazione dei dati; supporta in modo nativo i database HSQLSB, MySQL, Oracle, PostgreSQL, Netezza e Teradata.

Software di Web Scraping

Il web scraping è il processo attraverso il quale è possibile raccogliere automaticamente dati dal Web. Esistono diversi tipi di livelli di automazione; per esempio esistono software, come *Apache Tika*, oppure software per il parser di pagine HTML, e così via.

Apache Tika è uno strumento scritto in Java per l'identificazione e l'estrazione di metadati e testo da numerosi documenti dai formati più diversi. È un software molto utile per il recupero di dati sia da fonti esterne (Web), sia da fonti interne, come per esempio la documentazione interna di un'azienda. I formati da cui Tika può estrarre dati e metadati sono vari: HTML, XML, PDF, Microsoft Office, Open Document, EPUB, RTF, file compressi, file audio, immagini e persino classi Java e archivi Jar. Tika non esegue solamente l'estrazione di metadati e testo, ma permette anche il riconoscimento della lingua in cui il documento è scritto.

Lettura di stream di dati

La valocità di produzione che caratterizza alcune tipologie di dati ha reso necessarie tecnologie per la cattura in tempo reale e il trasferimento continuo dei dati. Un esempio open source è Apache Flume, servizio distribuito per la raccolta, l'aggregazione e lo spostamento di grandi moli di dati. Un'altra piattaforma per la gestione di stream di dati è Microsoft StreamInsight.

Apache Flume è un sistema distribuito per la movimentazione di grandi quantità di dati da fonti di diverso tipo a diversi tipi di filesystem distribuiti, o altre destinazioni (HBASE, Logger, etc...). Flume è caratterizzato

da una architettura semplice e flessibile basata sullo streaming di flussi di dati. L'architettura si basa sul concetto di *agent*, cioè una componente software che al suo interno gestisce autonomamente la raccolta dei dati provenienti dall'esterno, il passaggio dei dati attraverso il canale, ed infine, la lettura dei dati dal canale e l'instradamento verso la sorgente di destinazione. Un utilizzo piuttosto frequente consiste nel recuperare i dati di log da più web server e salvarli su filesystem distribuiti, dopo averli ricomposti.

I sistemi di *Complex Event Processing* (CEP) consentono di catturare eventi, anche ad alta frequenza, come per esempio tweet oppure dati inviati da sensori, in modo efficiente, combinandoli ed eventualmente analizzandoli in tempo reale, ovvero mentre sono recepiti dalle fonti, e fornire un risultato, salvandoli su una base di dati. **Microsoft StreamInsight** è una piattaforma per lo sviluppo di applicazioni di *Complex Event Processing* (CEP), un insieme di tecnologie per tracciare e analizzare stream di dati, anche provenienti da più fonti. StreamInsight consente di gestire un elevato numero di eventi in maniera efficiente e piuttosto semplice. Attraverso StreamInsight è possibile sia analizzare i dati e determinare trend e pattern in tempo reale, sia salvare i dati su una o più destinazioni. Microsoft StreamInsight non è l'unico strumento di *Complex Event Processing*, vi sono altri fornitore che permettono di processare stream di flussi di dati, come **Informatica RulePoint**, **Oracle Event Processing**, **SAP Event Stream Processor**, **Amazon Kinesis**, e molti altri.

1.4.2 Gestione e Memorizzazione dei Big Data

Negli ultimi anni con l'avvento dei big data è emersa la necessità di lavorare con database sempre più flessibili, ma sopprattutto scalabili. Le tecnologie tradizionali, utilizzate nel contesto dei big data, pongono due problemi che non possono essere trascurati:

- gestione di una grandissima mole di dati
- presenza di dati non strutturati o semistrutturati

Questi due aspetti hanno portato allo sviluppo di nuovi modelli di gestione dei dati, che da un lato vanno a coprire queste specifiche esigenze ma dall'altro si allontanano dal modello relazionale.

Fra le varie tecnologie con le quali è possibile far fronte a queste problematiche, la più diffusa e conosciuta è la piattaforma Hadoop: software

open source, affidabile e scalabile per il calcolo distribuito. I software di calcolo distribuito sfruttano la capacità computazionale di macchine distribuite, suddividendo tra loro l'esecuzione delle operazioni; in questo modo la capacitá di ciascun elaboratore si somma a quella degli altri, consentendo di gestire grandi moli di dati. Affinchè ciascuna macchina esegua le operazioni di calcolo è fondamentale l'accesso ai dati, e quindi ad **HDFS**, il file system distribuito utilizzato da Hadoop. HDFS, diversamente dai tradizionali file system, consente la memorizzare di file di grandi dimensioni (nell'ordine dei Terabytes e Petabytes di dati) su macchine distribuite; sfrutta la replicazione dei dati per ridurre la latenza di accesso ai dati e per la tolleranza ai guasti; inoltre fa uso di commodity hardware per favorire la scalabilità all'aumentare della mole dei dati.

Database NoSql

I file system distribuiti rappresentano una possibile soluzione alla gestione e memorizzazione dei big data, ma non è l'unica; negli ultimi anni, l'esigenza di gestire i big data ha portato alla nascita di un nuovo modello, che prende il nome di **NoSQL**. L'espressione NoSql, che sta per "**Not Only SQL**" o "**Not Relational**", non è contraria all'utilizzo del modello relazionale, ma fa riferimento a tutti quei database che si discostano dalle regole che caratterizzano i database ralazionali (RDBMS), strutturati intorno al concetto matematico di relazione o tabella.

Al crescere della quantità dei dati, i problemi di scalabilità e i costi legati ai database relazionali sono soltanto una parte degli svantaggi; molto spesso, quando ci si trova di fronte alla gestione di big data, anche la variabilità, ovvero la mancanza di una struttura fissa, rappresenta una problematica da non sottovalutare. I database NoSql, a differenza di quelli costruiti basandosi sul modello relazionale, non presuppongono una struttura rigida o uno schema, dove vengono descritte le proprietà che i dati dovranno avere e le relazioni tra loro. I database NoSql puntano sulla flessibilità e sulla capacità di gestire i dati con strutture difficilmente rappresentabili in formati tabellari.

La definizione di database NoSql, riportata sul sito ufficiale [7], mette in luce una serie di caratteristiche che contraddistinguono i database NoSql:

- *distributed*
- *open-source*

- *horizontally scalable*
- *schema-free*
- *easy replication support*
- *simple API*
- *eventually consistent / BASE model*
- *not ACID property*
- *huge amount of data*

Non è strettamente necessario che i vari database NoSql rispecchino contemporaneamente tutte le suddette caratteristiche, anche solo alcune di queste possono essere rispettate.

La natura distribuita dei database NoSql fa si che le proprietà ACID (Atomicity, Consistency, Isolation, e Durability), che caratterizzano i database tradizionali, non possano essere applicate a tale contesto; questa è una diretta conseguenza del **teorema CAP** (Consistency, Availability, Partition Tolerance), il quale afferma l'impossibilità per un sistema distribuito di fornire simultaneamente consistenza, disponibilità e tolleranza di partizione, ma è in grado di soddisfare al massimo due di questi aspetti allo stesso tempo, non tutte e tre. Le tre proprietà appena citate vengono definite nel seguente modo:

- *consistenza*: a seguito di una modifica sui dati, ciascun nodo del sistema dovrà visualizzare la stessa versione dei dati;
- *disponibilità*: ogni nodo di un sistema distribuito deve sempre rispondere alla richiesta di dati a meno che questo non sia indisponibile;
- *tolleranza di partizione*: capacità di un sistema di essere tollerante all'aggiunta o alla rimozione di un nodo del sistema.

I database NoSQL pertanto non offrono garanzie ACID, tuttavia sfruttano proprietà più flessibili e adatte al contesto NoSQL, nello specifico quelle del **modello BASE** (*Basically available, Soft state, Eventual consistency*), secondo cui il sistema deve essere sempre disponibile, e la consistenza, che

non viene garantita ad ogni istante, al termine delle operazioni/esecuzioni deve essere verificata. Il modello BASE potrebbe non risultare adatto per ogni situazione, ma risulta essere un'alternativa flessibile al modello ACID per applicazioni che non richiedono espressivamente di rispettare le proprietà ACID. La caratteristica dei database NoSql di poter scalare orizzontalmente consente di fare a meno di hardware performante ad alto costo, sostituendolo invece con commodity hardware. Infatti, le dimensioni di un cluster su cui è installato un database NoSql possono essere aumentate o diminuite, aggiungendo o rimuovendo nodi a piacere, senza particolari problematiche di gestione, realizzando così una piena scalabilità orizzontale a costi moderati.

La "semplicità" legata ai database NoSql, che consente al sistema di scalare orizzontalmente, così da aggiungere nodi in maniera trasparente all'utente, è legata all'architettura hardware utilizzata. I sistemi comunemente utilizzati per ospitare database relazioni rientrano nella tipologia degli SMP (*Symmetric MultiProcessing*. Questi sono costituiti da più processori che condividono lo stesso sistema operativo, la stessa memoria RAM e lo stesso bus di Input/Output. I sistemi SMP sono molto efficienti nelle applicazioni OLTP, ma presentano limiti quando li si utilizza per elaborare i Big Data. Il limite è dato dal sovraccarico del bus di sistema che costituisce un inevitabile collo di bottiglia. I sistemi MPP (*Massive Parallel Processing*) si differenziano dagli SMP per il fatto che ogni processore utilizza risorse a esso dedicate, sia per quanto riguarda la RAM sia per quanto riguarda il bus di I/O. I processori comunicano tra di loro attraverso un'interfaccia di *messaging*. Le limitazioni dovute alle condivisioni del bus vengono meno, rendendo così le architetture MPP adatte alla gestione di grandi quantità di dati.

Come tutti i modelli, anche quello NoSql, oltre ai vantaggi, presenta svantaggi che devono essere tenuti in considerazione nel momento in cui si sceglie il modello da utilizzare in un determinato contesto. Da un lato i tempi di risposta, all'aumentare della mole dei dati, risultano essere più performanti rispetti a quelli riscontrati con i database relazionali, grazie all'assenza delle costose operazioni di join sui dati che caratterizzano gli ambienti SQL. Le prestazioni ottenute in lettura però, vanno a discapito della **replicazione delle informazioni**, anche se in realtà, i costi sempre meno proibitivi dei sistemi di storage rendono questo svantaggio poco importante.

Dall'altro lato, la mancanza si uno **standard universale**, come per esempio SQL, che caratterizza i database relazionali. Ogni database appartenente al mondo NoSql, invece, ha a disposizione un insieme di API, metodi di storing e accesso ai dati che differiscono a seconda dell'implementazione che si considera.

Non esiste un'unica tipologia di implementazione, infatti i database NoSql vengono classificati sulla base di come i dati sono memorizzati. A seconda di come vengono memorizzati i dati è possibile individuare diverse implementazione del modello NoSql. Le principali sono:

- Column-oriented database
- Key/value store
- Document-oriented database
- Graph database

Di seguito sarà possibile trovare una breve descrizione delle 4 categorie NoSql appena citate, soffermandosi sulle principali implementazioni.

Column-oriented database

I column-oriented database, diversamente dai tradizionali RDBMS che memorizzano i dati per riga, sfruttano la memorizzazione dei dati per colonna. Nonostante i database relazionali sfruttano un modello secondo cui i dati possono essere visualizzati mendiante tabelle bi-dimensionale, caratterizzate da righe e colonne, le fasi di *storage* e *process* dei dati considerano sempre un riga per volta, invece, i database column-oriented processano e memorizzano i dati per colonna, rendendo così tali operazioni più efficienti. Per esempio, se si volessero memorizzare i seguenti dati:

Nome	Cognome	Città	Età	Professione
Mario	Bianchi	Bologna	27	Pasticcere
Luca	Rossi	Ravenna	32	Consulente
Giorgio	Blu	Cesena	30	Operaio

Nei database relazionali, i dati internamente verrebbero memorizzati come:

```
Mario,Bianchi,Bologna,27,Pasticcere
Luca,Rossi,Ravenna,32,Consulente
Giorgio,Blu,Cesena,30,Operaio
```

Invece, nel contesto dei database column-oriented, i dati verrebbero memorizzati come:

```
Mario,Luca,Giorgio
Bianchi,Rossi,Blu
Bologna,Ravenna,Cesena
27,32,30
Pasticcere,Consulente,Operaio
```

Ogni unità di dato può essere pensato come un insieme di coppie chiave/-valore, dove il dato viene identificato mendiante la chiave primaria, detta anche *row-key* in molte implementazioni. Queste unità vengono memorizzate in maniera ordinata sulla base della chiave di riga che le identifica.
L'organizzazione dei dati per colonna, invece che per riga, non è un modello presente unicamente nel movimento NoSql, ma è utilizzano anche in ambito business intelligence, grazie all'efficienza con cui i dati vengono memorizzati; inoltre la memorizzazione dei dati per colonna permette di evitare il fenomeno della Sparsità dei Dati, ovvero sprechi di spazio nel momento in cui un determinato valore non esiste per una determinata colonna.

HBase è uno dei più popolari database column-oriented, open source e distribuito basato sul modello di BigTable di Google e scritto in Java. Come riportato nel documento originale [6], **BigTable**, sviluppato dal colosso americano Google, è un sistema di storage distribuito, basato su commodity hardware, per la gestione dei dati strutturati e progettato per essere scalabile all'aumentare della mole di dati, su larga scala; molti progetti di Google gestiscono i propri dati attraverso BigTable, inclusa anche l'indicizzazione del Web. Nonostante il modello di HBase si basi su BigTable, è un progetto Apache e fa parte dell'ecosistema di Hadoop. L'integrazione con Hadoop è molto elevata, tanto che il database fa uso e si basa su Hadoop HDFS per la persistenza dei dati. I dati memorizzati su HBase possono essere manipolati tramite l'infrastruttura Hadoop MapReduce e, viceversa, MapReduce può utilizzare HBase come fonte sorgente e di destinazione dei dati.

Un'altro componente facente parte dell'ecosistema di Hadoop è Apache **Cassandra**, uno dei database column-orinted più diffusi e utilizzati, anche da aziende come eBay e GitHub [8]. Cassandra è un sistema distribuito

che si concentra principalmente sulla scalabilità e sulla disponibilità (*high availability*) senza compromettere le perfomance; infatti la particolarità di Cassandra è la *scalabilità lineare*: le perfomance di lettura e di scrittura aumentano linearmente con il numero di nuovi nodi che si aggiungo al cluster. Inoltre Cassandra ha un'architettura "masterless" che permette di gestire in completa autonomia la replicazione dei dati sul cluster di database, in maniera del tutto trasparente all'utente.

Key/value store

I Key/Value store rappresentano una tipologia di database NoSql che si basa sul concetto di *associative array*, implementati attraverso *HashMap*: i dati vengono rappresentati come una collezione di coppie chiave/valore. La chiave è un valore univoco con il quale è possibile identificare e ricercare i valori nel database, accedendovi direttamente. La tipologia di memorizzazione adottata dai key/value stores garantisce tempi di esecuzione costanti per tutte le operazioni applicabili sui dati: *add*, *remove*, *modify* e *find*. Fra le principali, la tipologia NoSql Key/Value Store è la più semplice e, in molte implementazioni NoSql, viene utilizzata come base di implementazione di altre tipologie, come ad esempio i document-oriented database.

Un semplice, ma potente, database key/value store è **Barkeley DB**, attualmente gestito e aggiornato da Oracle. Barkeley DB presenta tre diverse implementazioni (C, Java e C++) e consente il salvataggio delle coppie key/value in quattro diverse struttura dati: B-tree, Hash. Queue e Recno. Le caratteristiche principali di Barkeley DB comprendono la possibilità di utilizzare dati complessi sia come chiave sia come valore e la replicazione del database su più nodi in modo da consentire un accesso più veloce ai dati. Nell'ambito dei big data, un aspetto limitate è dato dall'impossibilità di scalare in base al volume dei dati, poiché la replica dei dati consiste in una copia dell'intero database su un altro server. Tuttavia, Barkeley DB è uno strumento molto importante, perchè utilizzando il motore che sta alla base di questo database è stato possibile costruire altri database NoSql, basati sul concetto di key/value store: **Project Voldemort** e **DynamoDB** di Amazon.

Document-oriented database

I database document-oriented, che non sono da confondere con i sistemi si gestione documentale (*content management system*), gestiscono in maniera molto efficiente dati semistrutturati. I *content management system* consentono la pubblicazione, la gestione e la condivisione di contenuti e documenti, invece i database document-oriented rappresentano una specializzazione dei key/value store: i *document* vengono rappresentanti come un insieme strutturato di coppie chiave/valore, spesso organizzti in formato JSON o XML. La struttura delle coppie chiave/valore non pone vincoli allo schema dei documenti garantendo così una grande flessibilità in situazione in cui, per loro natura, i dati hanno una struttura variabile. I document-oriented database infatti gestiscono i documenti nel loro insieme, evitando di suddividere tali documenti in base alla struttura delle coppie chiave/-valore. Questa tipologia di database NoSql permette di gestire in maniera molto efficiente l'aspetto della variabilità che caratterizza i big data, dimostrandosi particolarmente adatta alla memorizzazione di tipologie di dati complessi ed eterogenei. Tra i più famosi e utilizzati database open-source document-oriented è possibile trova **MongoDB** e **CouchDB**.

MongoDB è uno dei principali database document-oriented, scritto in C++ e progettato in base ai criteri tipici del modello NoSql. Ogni record in MongoDB è un documento, ovvero una struttura dati composta da coppie campo/valore o key/value (vedi Figura1.3). I valori dei campi possono includere altri documenti, array, oppure altri array di documenti. Ogni documento ha un campo predefinito, il campo "_id", che può essere assegnato in fase di inserimento, oppure, in mancanza di un valore, il sistema ne assegna uno in modo univoco. MongoDB, per rappresentare i documenti, utilizza il formato BSON, ovvero una rappresentazione binaria molto simile a JSON, con in più alcuni tipi aggiuntivi. Come molti degli strumenti NoSql, MongoDB gestisce la replicazione dei dati per mantenere un'alta disponibilità dei dati e permette la scalabilità orizzontale distribuendo i dati e le repliche sui nodi del cluster. MongoDB contiene un motore di aggregazione dei dati che permette di processare i dati e restituirne un risultato. L'insieme delle operazioni di aggregazione prendono in input una collezione di documenti e restituiscono in output il risultato sotto forma di uno o più documenti. Per operazioni di calcolo semplici, il motore di aggregazione è

Figura 1.3: MongoDB Documents [9]

più che sufficiente, ma quando la complessità e la mole di dati aumentano (come nel caso dei big data), è necessario utilizzare strumenti più potenti, come MapReduce.

MongoDB può essere utilizzato in combinazione con MapReduce secondo due differenti modalità: attraverso la funzione interna di *mapreduce*, oppure utilizzando l'interfaccia Hadoop. La funzione integrata consente di eseguire job in stile MapReduce, non interagendo direttamente con l'ambiente Hadoop, ma rappresenta un'implementazione interna a MongoDB. Invece, a differenza della funzione mapreduce integrata, l'interazione con l'ambiente Hadoop permette a MongoDB di essere utilizzato sia come fonte, sia come destinazione dei dati per i job MapReduce di Hadoop. La combinazione Hadoop-MongoDB può essere utilizzata in modi diversi a seconda dello soluzione che si intente implementare:

- I dati che risiedono su MongoDB vengono estratti ed elaborati attraverso uno o più job MapReduce di Hadoop; in questo scenario il risultato della computazione confluisce nuovamente su MongoDB, a disposizione delle applicazioni che poggiano sul database (vedi Figura1.4).

- In molti scenari aziendali, i dati applicativi risiedono su diversi archivi, ognuno con le proprie funzionalità e il proprio linguaggio di interrogazione. Per ridurre la complessità di analisi, la piattaforma Hadoop

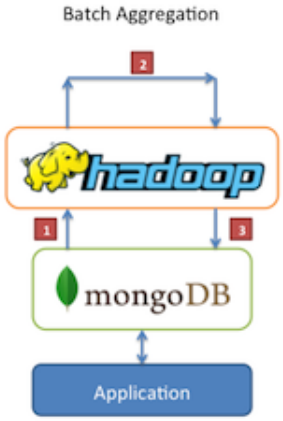


Figura 1.4: Batch Aggregation [9]

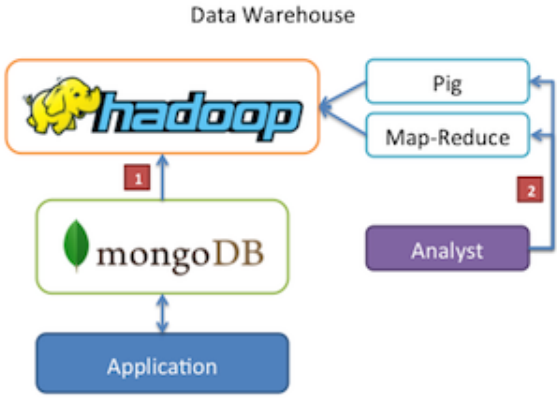


Figura 1.5: Data Warehouse [9]

potrebbe essere utilizzata come archivio centralizzato contenente i dati provenienti dalle diverse fonti. In questo scenario MongoDB fa parte delle fonti che vanno ad alimentare Hadoop (vedi Figura1.5).

- Infine, MongoDB può essere utilizzato come archivio sorgente oppure destinazione di un processo di ETL (Extract, Transform, Load) (vedi Figura1.6)

Graph database

I graph database rappresentano una particolare categoria di database NoSql, in cui le "relazioni" vengono rappresentate come *grafi*. Il concetto matematico di grafo consiste in un insieme di elementi detti *nodi* collegati fra loro da *archi*. Nell'ambito informatico il grafo rappresenta una struttura dati costituita da un insieme finito di coppie ordinate di oggetti. Le strutture a grafo si prestano molto bene per la rappresentazione di determinati dati semistrutturati e altamente interconnessi come, ad esempio, i dati dei social network e del Web. Tutte le categoria di database NoSql fin'ora descritte, sono in grado di rappresentare le relazioni di interconnessioni tra i dati, ma questo porterebbe ad avere scenari complessi sia da interrogare, sia da aggiornare a fronte di cambiamenti nei dati. I graph database sono stati pensati appositamente per rappresentare e navigare in maniera efficiente

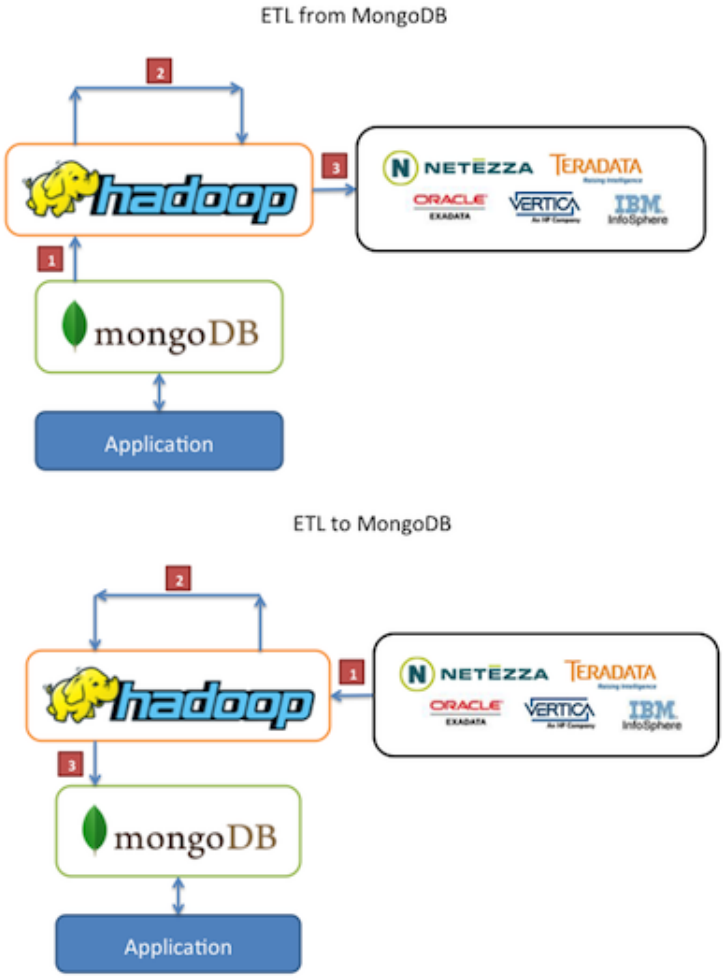


Figura 1.6: ETL from/to MongoDB [9]

dati altamente interconnessi, adattandosi facilmente ai cambiamenti delle strutture dei dati.

Quando si parla di modelli basati sul concetto di grafo è inevitabile fare riferimento alla teoria dei grafi. Attualmente, i due modelli di riferimento per l'implementazione dei database a grafo sono due: *property graph* e il *resource description framework graph* (RDF). Esistono diverse implementazioni di database graph, tra i più utilizzati vi sono: **Nao4j** basato sul modello property graph e **AllegroGraph** basato, invece, sul modello RDF.

1.4.3 Analisi e Processamento dei Big Data

La varietà, il volume e la velocità che caratterizzano i big data hanno progressivamente cambiato il modo di vedere e fare analisi sui dati. La numerosità e la varietà delle fonti che ad oggi possono essere utilizzate per alimentare i big data, hanno portato nuove opportunità per le organizzazioni, che dovranno o si sono già organizzate per gestire e analizzare tale mole di dati. I dati sono stati raccolti nel tempo e di questi sempre più sono raccolti in tempo reale; la chiave per trasformarli in risorse utile, sta tutta nella capacità di estrarre informazioni nuove ed di valore a supporto dei proccessi decisionali. Serve perciò comprendere quali strumenti e tecnologie utilizzare per ottenere queste informazioni.

In principio, le aziende estrapolavono informazioni e provvedevano ad esplorare e analizzare i dati a consuntivo mediante strumenti OLAP, SQL, Excel, etc. Negli ultimi anni, la competitività sempre maggiore e la velocità con cui cambiano i fatti, ha fatto nascere tecniche di analisi predittive e di monitoring in tempo reale: si applicano strumenti di data mining e machine learning per la creazione di modelli che permettano di identificare patterns comportamentali e tendenze che possono essere utilizzati per prevedere eventi futuri e ottimizzare i processi di business. L'analisi avanzata dei big data potrebbe avere degli impatti rivoluzionari sul business e sui sistemi informativi di governi, imprese e individui; potrebbe contribuire a risolvere problemi che affliggono da molto tempo le aziende. Gli aspetti che caratterizzano i big data hanno portato allo sviluppo di nuovi strumenti, da poter applicare ed utilizzare per estrapolare valore dai dati. Questi nuovi strumenti, anche se paragonalibi agli strumenti di business analitycs, dovendo trattare grandi quantità di dati semistrutturati ad alta velocità, devono far fronte ad una serie di caratteristiche come, ad esempio, la flessibilità, la

complessità e la velocità.

Nel contesto dei big data, molto spesso si parla di Big Data Analitycs: insieme di processi che applicano strumenti di *business analitycs* in grado di esaminare grandi quantità di dati, caratterizzati da una struttura non fissa, al fine di identificare patterns, correlazioni o andamenti nascosti nella moltitudine di dati grezzi. Gli aspetti che stanno alla base degli strumenti di business analitycs sono gli stessi che vengono applicati nel contesto dei big data, considerando però varietà, velocità e volume, non contemplati negli strumenti tradizionali, ma fondamentali per le performance dei processi in ambito big data.

Molti sono gli strumenti che possono essere applicati, ognuno con le proprie caratteristiche e il proprio obiettivo. In base alla piattaforma adottata in fase di gestione dei big data e a seconda di cosa si intende esaminare, individuare o valutare si sceglierà lo strumento di analitycs che meglio si addice. Fra i principali strumenti utilizzati in ambito big data analitycs è possibile individuare:

- Pig
- Hive
- R
- Presto
- Impala
- Hadoop MapReduce
- Mahout
- Drill

ma ve ne sono molti altri, che non vengono trattati nel presente elaborato di tesi.

Molti degli strumento sopra elencati fanno parte dell'ecosistema di Hadoop, e come tali verranno trattati più nel dettaglio nel quarto capitolo, nel paragrafo dedicato ad Hadoop. Fra questi vi sono anche strumenti che,

nonostante non facciano parte dell'ecosistema di Hadoop, poggiano e utilizzano le funzionalità di suddivisione e distribuzione del lavoro di MapReduce e HDFS. Alcuni di questi sono descritti qui di seguito.

Impala è un motore di interrogazione SQL open source per l'elaborazione massiva in parallelo (MPP) di dati gestiti su un cluster Hadoop. Impala permette di eseguire in tempo reale interrogazioni SQL-like su dati memorizzati e gestiti da HDFS o HBase, facendo uso di funzioni di aggregazione, select e join. Fa uso della stessa sintassi SQL, interfaccia utente e degli stessi driver ODBC/JDBS di Apache Hive. Per ridurre la latenza, Impala aggira la logica di MapReduce e accede direttamente ai dati attraverso un motore distribuito di query specifico (*Impala Deamon*), molto simile a quello utilizzato negli RDBMS paralleli (Vedi Figura1.7). Impala non pre-

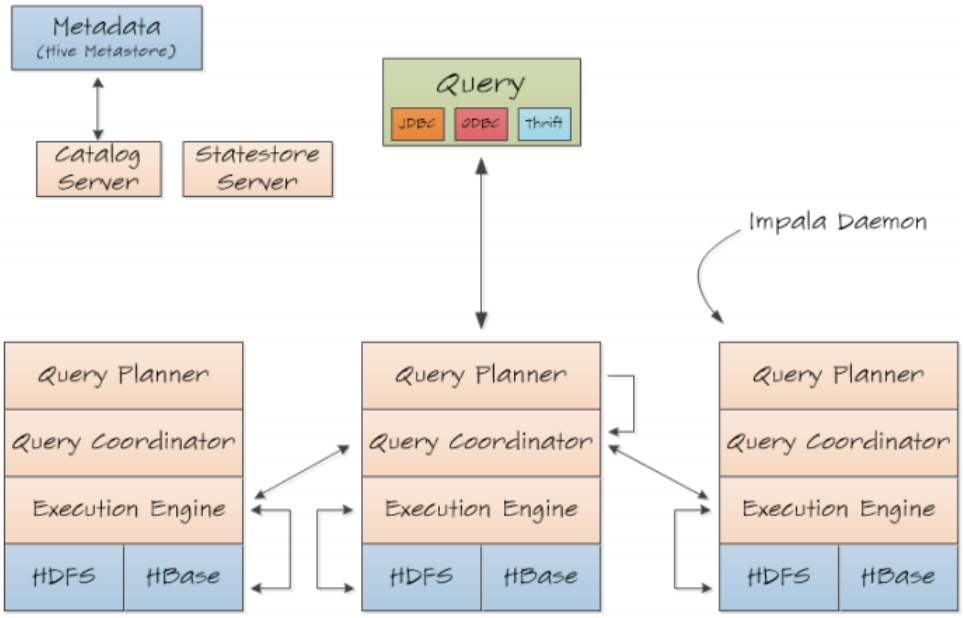


Figura 1.7: Architettura Impala [11]

suppone un servizio "master" che coordina le interrogazioni distribuite, ma prevede un insieme di *impala deamon*, uno per ogni nodo del cluster, ognuno dei quali è caratterizzato da tre ruoli principali: Planner, Coordinator e Execution. Un demone impala non è altro che un processo in esecuzione,

che accetta ed esegue le query richieste dell'utene e ricompone i risultati parziali ottenuti dai singoli nodi. Grazie all'esecuzione distribuita di query si evita di rendere la rete un collo di bottiglia; inoltre tutte le risorse a messe a dispozione dal cluster sono riservate per l'esecuzione dei demoni Impala.

Diversamente da Impala, **Presto** il motore open source di query SQL distribuite sviluppato da Facebook non si limita al solo accesso ad HDFS, ma è stato pensato per operare su differenti tipi di sorgenti dati, inclusi i tradizionali database relazionali e altre sorgenti, come ad esempio Cassandra. Presto è stato progettato per la gestione di data warehouse e l'analisi, su grandi quantità di dati. L'architettura di Presto, rispetto a quella di Impala, è caratterizzata da un coordinatore principale (*Presto Coordinator*) che gestisce, analizza e pianifica l'esecuzione delle query, distribuendo il carico di lavoro ai singoli worker (*Presto Worker*). Come per Impala, non fa uso della logica di MapReduce, ma la evita accedendo direttamente ad HDFS. (Vedi Figura1.8).

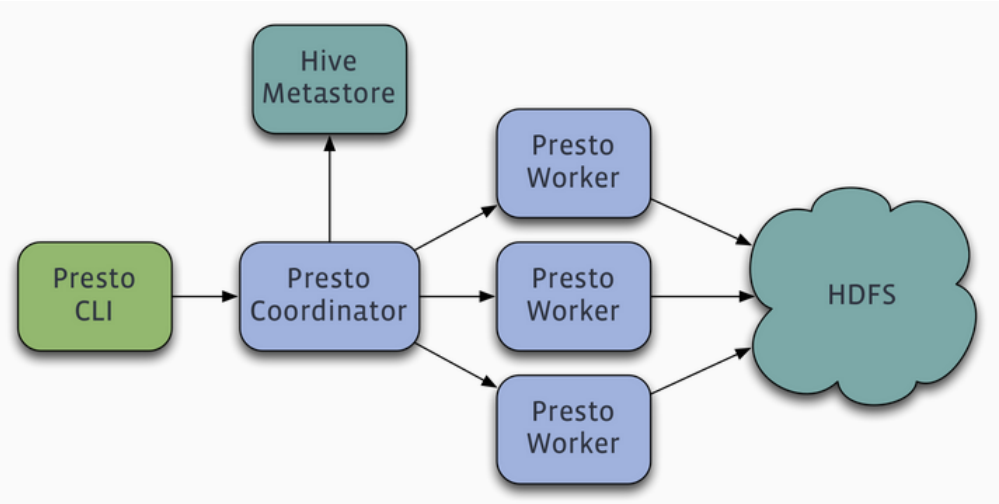


Figura 1.8: Architettura Presto [12]

Negli ultimi tempi, molto utilizzato è l'ambiente **R**, un software di calcolo statistico *free*. R non è uno strumento che nasce per l'elaborazione dei big data, ma tramite uno specifico linguaggio di programmazione, consente di eseguire analisi statistiche di base e avanzate. R è in grado di leggere dati da file e da database relazionali, grazie a un package che implementa

la connettività via ODBC/JDBC. Nel corso degli anni, infatti, sono stati sviluppati numerosi package che ne estendono le funzionalità, rendendolo uno strumento completo e versatile. Quando le dimensioni dei dati da elaborare assumono dimensioni enormi, tali da qualificare i dati come big data, è preferibile utilizzare sistemi, come Hadoop, in gradi di lavorare su enormi moli di dati. Le funzionalità analitiche di R sono piuttosto avanzate e non semplici da replicare attraverso job MapReduce, ma la presenza di package che consentono di utilizzare R su dati presenti dell'ecosistema Hadoop semplifica notevolmente questo passaggio. Uno di questi, RHadoop consente di utilizzare Hadoop e MapReduce in combinazione con R, ampliandone notevolmente le capacità di calcolo. RHadoop è composto da diverse parti:

- *Rmr* fornisce le funzionalità di Hadoop MapReduce
- *Rhdfs* permette la gestione dei file in HDFS
- *Rhbase* permette l'utilizzare del database HBase da R

L'altro package disponibile in R è RHive, che fornisce un'interfaccia verso Hive, lo strumento di data warehousing di Hadoop. RHive contiene diversi gruppi di funzioni che gestiscono funzionalità base di querying con Hive, l'accesso a HDFS o la creazione di funzioni da eseguire su MapReduce.

Grazie ai numerosi package che estendono le funzionalità e consentono l'utilizzo su Hadoop, R è adatto a numerose tipologie di analisi. Oltre ai calcoli statistici di base, con R è possibile svolgere:

- analisi statistiche avanzate;
- machine learning;
- text mining;
- analisi di serie storiche;
- analisi dei grafi;
- etc..;

R non è l'unico strumento che permette di svolgere queste tipologie di analisi su grandi moli di dati, ve ne sono molti altri. A seconda della tipologia di analisi che si vuol fare, del contesto di riferimento e degli strumenti utilizzati è possibile scegliere lo strumento di analisi ideale per le proprie esigenze.

1.5 Big Data: Lo stato dell'arte

Negli ultimi tempi uno dei termini più ricorrenti, non solo nell'ambito IT, è quello dei Big Data. A seconda del contesto e dell'idea percepita il concetto di big data assume un valore diverso poiché le correnti di pensiero su questo argomento sono molteplici e molte volte discordanti fra di loro.

Di seguito si cercherà di esaminare le percezioni in Italia e all'estero sull'argomento dei big data, del loro uso e delle problematiche che circonda il loro mondo.

1.5.1 In Italia

In Italia il fenomeno e il concetto dei Big Data a poco a poco si sta diffondendo fra le aziende, i media e gli enti pubblici. Questo è riscontrabile nei risultati che sono emersi da una ricerca svolta dall'Osservatorio Big Data Analitycs & Business Intelligence [13], in cui si mostra come nell'arco del 2014 il mercato dei big data in Italia è cresciuto del 25%. Il grande successo mediatico e la consapevolezza che il mondo dei dati sta evolvendo, ha portato molti vendor ad integrare nelle loro soluzioni moduli a supporto dei big data, e dualmente molte aziende a valutare possibili cambiamenti nelle proprie logiche di analisi. Nonostante ciò, ad oggi, sono ancora poche le aziende che hanno realmente integrato nei propri sistemi e modelli di governance i flussi big data. Le cause principali possono essere ricercate nella:

- mancanza di *skill* (di competenze) e poca dimestichezza con le nuove tecnologie;
- mancanza di standard tecnologici.

Vi sono poi moltre altre aziende, che diversamente dagli aspetti appena citati, non hanno interesse in tali tecnologie o ritengono di non poterne trarre benefici.

Su altri fronti invece, c'è chi sostiene che i big data in Italia non esistono [15], affermando che le aziende italiane non producono una mole di dati tale da essere considerati "big", per cui facilmente gestibili con semplici fogli di calcolo o RDBMS tradizionali. Questo mostra come il concetto dei big data, nonostante sia di dominio pubblico, non abbia raggiunto una comune definizione, secondo cui la sola caratteristica della quantità non basta a descrivere questo fenomeno complesso e variegato. Non è l'unico esempio, osservando blog e giornali online è possibile notare come le molte inesattezze siano dovute probabilmente alla scarsa conoscenza del fenomeno, delle tecnologie e alla dimenticanza di molte altre componenti.

Nel nostro paese, però alcune importanti aziende italiane fra cui Telecom Italia e Mediaset S.p.a. hanno intravisto nei Big Data un'ambito su cui poter investire tempo e risorse.

- Mediaset S.p.A., azienda privata italiana, considerando i social media una nuova fonte estremamente interessante e rilevante per comprendere le opinioni dei propri clienti, ha deciso di testare una soluzione [16] in grado di raccogliere e analizzare i dati non strutturati dei social e successivamente compararli sia con i dati oggi disponibili in azienda sia con i dati di mercato provenienti dalle fonti più tradizionali. Mediaset si è posta come obiettivo principale quello di valutare se l'analisi dei big data possa affiancare gli altri strumenti di marketing a disposizione dell'azienda e di misurare il grado di soddisfazione dei clienti rispetto ai propri prodotti e rispetto a quelli dei concorrenti. La soluzione prodotta ha permesso di analizzare molteplici fonti dati non strutturate, individuare "hot words", trend dei prodotti e servizi offerti da Mediaset, comparando tali informazioni con le altre disponibili in azienda per verificare la valenza di queste rispetto alle logiche di business prefissate.

- Telecom Italia, invece, nei primi mesi del 2014 ha dato vita al contest Big Data Challenge [17], evento nato per stimolare la creazione e lo sviluppo di idee tecnologiche innovative nel campo dei Big Data, permettendo ai professionisti e agli appassionati del settore di confron-

tarsi a livello globale, scegliendo una delle tre aree d'azione proposte di sviluppo: applicazioni, data analytics e visualizzazione di dati.

Dalla valutazione dei punti di vista sopra trattati, si può notare come nel nostro paese i Big Data vengono visti ancora con diffidenza e pochi addetti ai lavori scorgono invece la loro grande potenzialità per una nuova forma di business. I big data rappresentano un motore di innovazione e sarà una delle maggiori sfide che l'Italia dovrà affrontare nel prossimo futuro; diversamente, paesi come gli Stati Uniti, stanno già affrontando questa sfida.

1.5.2 All'Estero

All'estero, e in particolare negli Stati Uniti, già da tempo i big data vengono percepiti come un'opportunità a supporto dei processi di business. Secondo una ricerca del TWDI (*The Data Warehouse Institute*) [18], il 33% del campione preso in esame, dichiara di svolgere già analisi sui big data e un'altro 33% ha dichiarato di aver già pianificato l'integrazione dei big data nei propri processi di analisi entro i prossimi due anni.

Le elezioni americane del 2012 hanno reso palpabile la potenza dei big data. Gli strateghi delle due campagne elettorali si sono avvalsi di sistemi in grado di sfruttare i big data per censire gli elettori, capirne gli umori e indirizzare la campagna. Come si è visto poi, la macchina organizzativa di Barack Obama ha avuto la meglio sui sistemi utilizzati nella campagna di Mitt Romney, andati in "panne" mentre le urne erano ancora aperte.

Fino ad oggi, a differenza del contesto italiano, le istituzioni americane si sono dimostrate più efficienti nel supportare i processi di innovazione. Le iniziative prese e i progetti attivi, in ambito big data, sono numerosi. Un esempio è dato dalla Chicago Architecture Foundation che, basandosi sulla convinzione che l'esplosione di dati a cui si sta assistendo sta a poco a poco trasformando il modo di costruire e vivera la città, ha allestito la mostra "Chicago: City of Big Data" [19] per diffondere e illustrare come le istituzioni e la città di Chicago utilizzano i dati raccolti, per comprendere e migliorare i problemi legati alla vivibilità della città. Ad oggi la città di Chicago ha messo in pratica tutta una serie di iniziative che l'hanno portata ad essere considerata il "cuore digitale" degli Stati Uniti, mostrando così come le istituzioni sfruttano i processi di innovazione e i mezzi a loro disposizione, a supporto della colletività.

I campi in cui l'utilizzo dei big data sta prendendo piede sono svariati, dalla sicurezza per la prevenzione contro il crimine e il terrorismo, alla sanità per l'analisi delle epidemie influenzali. Sono molti i benefici che è possibile ottenere dal trattamento e dall'integrazione dei big data nei processi che regolano le aziende e la comunità, ma vi sono anche criticità che non possono essere sottovalutate. La raccolta e l'analisi di enormi banche dati da parte di molti soggetti, istituzionali e non, mette in serio pericolo la *privacy*, aspetto che negli ultimi anni ha portato l'america al centro del mirino mondiale. Privacy, proprietà dei dati e, di conseguenza, la possibilità del loro utilizzo da parte di terzi sono problemi da non trascurare quando si affrontano le tematiche legate ai big data, e riguardano sia le tipologie di dati, sia le informazioni che è possibile estrarne attraverso l'analisi. La semplice trasformazione in forma anonima delle singole informazioni raccolte non garantisce contro eventuali abusi in fase di elaborazioni. Le istituzioni e gruppi di ricerca si sono messi all'opera per studiare e valutare potenziali soluzioni al problema della violazione della privacy. Nel corso del 2014, l'Ufficio per la politica della scienza e della tecnologie della Casa Bianca in collaborazione con il Massachusetts Institute of Technology (MIT) hanno valutato in che modo la crittografia e altre tecnologie orientate alla privacy possono proteggere le informazioni coinvolte nell'elaborazione dei big data. Diverse sono le soluzioni prodotte, ma ognuna presenta vincoli e/o problematiche che vanno in conflitto o con la logica dei big data o con il concetto di privacy. L'individuazione di una possibile soluzione porterebbe a risolvere le molte questioni politiche e filosofiche che circonda il contesto dei big data.

Nonostante le varie problematiche, i big data si sono dimostrati, sia nel contesto italiano che all'estero, un valido strumento innovativo su cui sarebbe opportuno scommettere e investire per dare una svolta al sistema, che molto spesso non coglie il potenziale di tali strumenti.

Capitolo 2

Hadoop 2.x

2.1 Introduzione ad Hadoop

Hadoop è un framework Open Source di Apache, affidabile e scalabile, finalizzato al calcolo distribuito di grandi quantità di dati.
Hadoop nasce all'interno del progetto Nutch (sotto-progetto di Apache Lucene), *crowler* open source che si occupa di navigare il Web in modo sistematico, recuperando i contenuti delle pagine da fornire al motore di ricerca per l'indicizzazione. Gli stessi creatori di Nutch, Doug Cutting e Michael J. Cafarella, a partire dal 2004, sfruttando le tecnologie di Google File System e Google MapReduce, svilupparono il primo prototipo di Hadoop. Allora rappresentava solamente un componente di Nutch in grado di migliorarne la scalabilità e le prestazioni.
Hadoop divenne un progetto indipendete di Apache solamente quando Yahoo!, durante la ristrutturazione del sistema di generazione degli indici per il proprio motore di ricerca, assunse nel 2008 Doug Cutting, al quale fu assegnato un team di sviluppo dedicato e le risorse necessarie per sviluppare la prima release di Hadoop.
Prima di Hadoop, le elaborazioni su grandi quantità di dati erano realizzate esclusivamente da sistemi di *High Performance Computing* (HPC) e *Grid Computing*. Hadoop diversamente da questi sistemi, oltre ad offrire un insieme di librerie di alto livello più semplici da utilizzare, sfrutta la replicazione dei dati sui singoli nodi per migliorare i tempi di accesso, trascurando così la latenza dovuta alla rete. Le attività (in lettura) di gestione dei calcoli e di elaborazione di grandi moli di dati, che caratterizzano Hadoop, risulta-

no essere esattamente l'opposto rispetto alle attività svolte da un database relazionale OLTP, dove le singole transazioni interessano solamente pochi record. L'utilizzo di Hadoop in tali scenari non sarebbe efficiente poichè si tratta di attività gestite in modo ottimale dagli RDBMS.

Attualmente sono state sviluppate due versioni base di Hadoop:

- Hadoop 1.x, versione "classica"
- Hadoop 2.x "YARN", detta anche versione di "nuova generazione"

Nonostante la versione 1.0 siano ancora oggi molto utilizzata, nei paragrafi successivi verranno descritti più nel dettaglio componenti e logiche di funzionamento di Hadoop 2.0, con qualche riferimento alla versione precedente.

2.2 Caratteristiche di Hadoop

Entrambe le versioni presentano 3 componenti essenziali che costituiscono il nucleo centrale della piattaforma:

- **Hadoop Common**: rappresenta lo strato di software comune che fornisce le funzioni di supporto agli altri moduli;
- **HDFS (Hadoop Distributed File System)**: come riportato nella documentazione ufficiale [21] HDFS è il *filesystem* distribuito di Hadoop progettato appositamente per essere eseguito su *commodity hardware*. Quando la mole di dati diventa "troppo grande" per la capacità di memorizzazione di una singola macchina, diventa necessario partizionare i dati su un certo numero di macchine separate. I *filesystem* che gestiscono l'archiviazione dei dati mediante una rete di macchine sono chiamati *filesystem* distribuiti. Rispetto ai normali filesystem, i distribuiti si basano sulla comunicazione in rete, per questo risultano essere più complessi.
- **MapReduce**: si occupa della schedulazione ed esecuzione dei calcoli. Lavora secondo il principio "*divid aet impera*": un problema complesso, che utilizza una gran mole di dati, viene suddiviso, assieme ia relativi dati, in piccole parti processate in modo autonomo e, una volta che ciascuna parte del problema viene calcolata, i vari risultati parziali sono "ridotti" a un unico risultato finale.

HDFS e MapReduce rappresentano il cuore del framework Hadoop, affinché la computazione possa essere portata a termine HDFS e MapReduce devono collaborare fra loro. A questi è poi possibile aggiungere tutti i componenti che fanno parte dell'ecosistema di Hadoop e che svolgono numerose differenti funzionalità, innestate sulla parte *core*.
Hadoop è un sistema:

- **altamente affidabile**: essendo pensato per un cluster di commodity hardware, che può essere frequentemente soggetto a problemi, permette di facilitare la sotituzione di un o più nodi in caso di guasti.

- **scalabile**: la capacità computazionale del cluster Hadoop può essere incrementata o decrementata semplicemente aggiungendo o togliendo nodi al cluster.

Dal punto di vista architetturale in un cluster Hadoop non tutti i nodi sono uguali, ma esistono due tipologie di nodi:

- ***master***

- ***worker***

Sui primi vengono eseguiti i processi di coordinamento di HDFS e MapReduce; i secondi invece vengono utilizzati per la memorizzazione e il calcolo. A seconda della versione di Hadoop (1.x oppure 2.x) che si considera i ruoli e le attività dei processi che vengono eseguiti sui vari nodi cambiano.

Nei prossimi due paragrafi verranno descritte architetture e funzionamento dei componenti core: HDFS e MapReduce che, anche se con aspetti differenti, caratterizzano entrambe le versione di Hadoop.

2.3 HDFS: Architettura e Funzionamento

Hadoop Distributed File System (HDFS) è stato progettato per la gestione dei flussi e memorizzazione affidabile di grandi volumi di dati; in particolare, ha lo scopo primario di gestire l'input e l'output dei job mapreduce. Gli aspetti principali che lo caratterizzano sono:

- *Very Large Files*: non esiste un limite esplicito sulle dimensioni dei file contenuti al suo interno. Ad oggi vi sono cluster Hadoop, come per esempio quello di Yahoo!, che arrivano a gestire petabytes di dati;
- *Streaming Data Access*: è particolarmente adatto per applicazioni che elaborano grandi quantità di dati. Questo perché il tempo che occorre per accedere all'intero set di dati è relativamente trascurabile rispetto al tempo di latenza dovuto alla lettura di un solo record;
- *Commodity Hardware*: è stato progettato per essere eseguito su cluster di commodity hardware, ovvero hardware a basso costo, in modo tale da aumentare la tolleranza ai guasti (faut-tolerance), molto probabili quando si ha a che fare con cluster di grandi dimensioni.

Descritte le caratteristiche, è possibile proseguire con la descrizione dell'architettura di HDFS.

2.3.1 Architettura

I file, all'interno di HDFS, vengono partizionati in uno o più **blocchi** (*blocks*), ognuno, di default da 128 MB (dimensione modificabile). Diversamente da altri filesystem, se un file risulta essere più piccolo della dimensione del blocco, non viene allocato un blocco "intero", ma soltanto la dimensione necessaria al file in questione, risparmiando così spazio utilizzabile.
Affinchè venga mantenuto un certo grado di tolleranza ai guasti (*fault-tollerance*) e disponibilità (*availability*), HDFS prevede che i blocchi dei file vengano replicati e memorizzati, come unità indipendenti, fra i nodi del cluster. Se un blocco risulta non più disponibile, la copia che risiede su un'altro nodo ne prende il posto, in modo completamente trasparente all'utente. Le repliche sono utilizzate sia per garantire l'accesso a tutti i dati, anche in presenza di problemi a uno o più nodi, sia per migliorare il recupero dei dati. Sia la dimensione dei blocchi, sia il numero di repliche possono essere configurati dall'utente.

Come accennnato precedentemente, ogni cluster Hadoop presenta due tipologie di nodi, che operano secondo il pattern ***master-slave***. HDFS presenta un'architettura in cui un nodo master identifica il *NameNode* e un certo numero di nodi slave identificano i *DataNode* (Vedi Figura2.1).
Come per molti altri DFS (*Distributed File System*), ad esempio GFS (*Google File System*), anche HDFS gestisce separatamente i dati applicativi dai

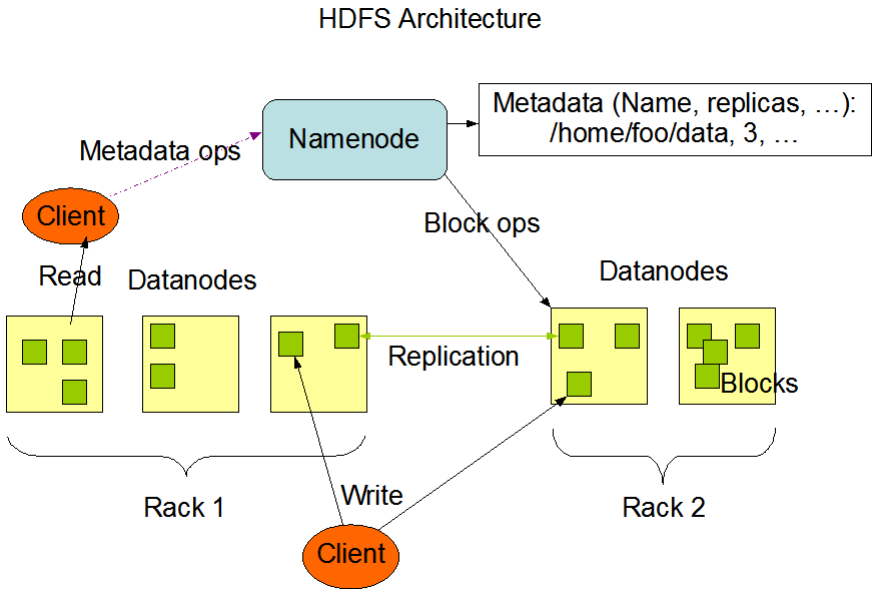


Figura 2.1: HDFS Architecture [21]

metadati, questi ultimi vengono memorizzate su un server dedicato, chiamato *NameNode*, invece i dati applicativi vengono gestiti da altri server, detti *DataNode*. Tutti i server in questione sono totalmente collegati e comunicanti fra di loro.

Il **NameNode** gestisce lo spazio dei nomi (*namespace*) del filesystem, ovvero una struttura gerarchica di file e directory, sul quale vengono mappati tutti i singoli blocchi dei file presenti all'interno del filesystem. Per far si che il fattore di replicazione (di default pari a 3) di ogni blocco sia mantenuto, il namenode memorizza per ognuno di questi, la lista dei datanode che ne possiedono una copia. Tale configurazione del namespace però, non è permanente, le informazioni relative alle replicazioni dei blocchi vengono ricostruite ad ogni avvio del sistema.
Senza il namenode, il filesystem non sarebbe utilizzabile; se il nodo su cui è in esecuzione il namenode "cade", tutti i file contenuti nel filesystem non sarebbero raggiungibili perché non sarebbe possibile dedurre su quale macchina del cluster sono collocati. Hadoop ha sviluppato due meccanismi distinti per rendere il namenode maggiormente resistente ai guasti. Il primo

modo consiste nel fare il *Backup* dei file che compongono lo stato persistente dei metadati del filesystem. Il secondo metodo prevede l'esecuzione di un *Secondary Namenode*, componente che, nonostante il nome, non agisce come un namenode, ma ha il compito di integrare il contenuto del namenode con quello del log delle modifiche. Il secondary namenode solitamente viene eseguito su uno nodo fisicamente separato da quello che contiene il namenode principale, e in casi di malfunzionamento del namenode principale può essere utilizzato come suo sostituto. Tuttavia, lo stato del secondary namenode non è perfettamente allineato con il namenode principale, per cui in caso si guasto, è molto probabile una certa percentuale di perdita di dati.

I **DataNode**, collocati sui nodi worker, gestiscono fisicamente lo storage dei blocchi di dati su ciascuno nodo. Periodicamente comunicano al namenode la lista dei blocchi che memorizzano e, all'occorenza, eseguono le operazioni richieste dai client, entità che interagiscono in lettura e scrittura con HDFS. Solitamente ogni datanode viene posizionato su una macchina distinta del cluster; queste a loro volta possono essere raggruppate in *rack*, strutture simili ad armadi che possono ospitare più server. I rack vengono definiti in fase di setup e di configurazione del cluster. La strategia di replicazione di Hadoop sfrutta questa configurazione del cluster copiando i blocchi su altri datanode appartenenti a rack diversi da quello di origine. Ciò minimizza i rischi legati ai guasti (sia dei rack, sia dei nodi), massimizza le perfomance di lettura ma appesantisce la scrittura.

La versione 2.x di Hadoop ha introdotto due nuovi concetti, non presenti nella versione precedente, che vanno a migliorare l'architettura di HDFS:

- *HDFS Federation*
  Per cluster di grandi dimensioni, la memoria assegnata al namenode diventa un fattore limitante. HDFS Federation, allo scalare del cluster, permette l'aggiunta di namenode, ognuno dei quali gestisce una porzione del namespace del filesystem. I vari namenode non comunicano fra di loro, per cui il guasto di uno di questi non influisce sulla disponibilità degli altri. In questo contesto, ogni namenode gestisce una *namespace volume*, composta dai metadati del namespace e da una *block pools* contenente tutti i blocchi dei file contenuti nel namespace.

- *HDFS High-Availability*

Nonostante l'introduzione dell'HDFS Federation e l'utilizzo del Secondary Namenode per evitare la perdita di dati, il Namenode rappresenta ancora un *single point of failure*. Se , tutti i client, inclusi i job di mapreduce, non sarebbero in grado di leggere, scrivere o visualizzare l'elenco dei file, perché è il solo repository..... In tal caso l'intero sistema di Hadoop potrebbe risultare fuori servizio fino a quando non viene ripristinato nuovamente il Namenode. La versione di Hadoop 2.x rimedia a questa situazione introducendo HDFS High-Availability (HA). Tale implementazione prevede una coppia di namenode, uno *attivo* e l'altro in *standby*. Se il primo dovesse, per qualche motivo, fallire, il namenode in standby prende il posto di quello attivo, continuando a servire le richieste dei client senza interruzioni significative.

2.3.2 Strategia di Replicazione

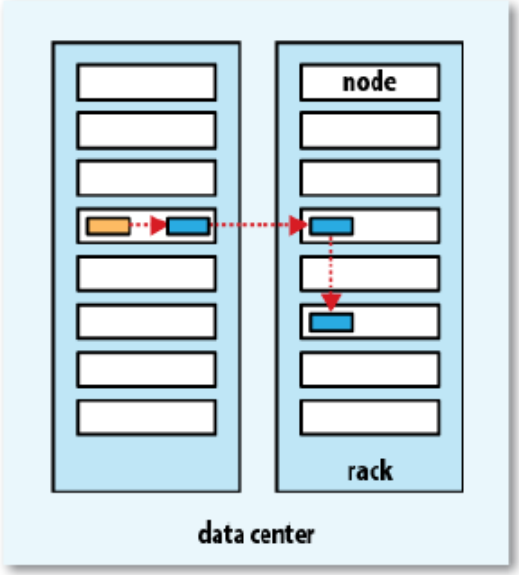


Figura 2.2: Strategia predefinita di replicazione [21]

Durante la fase di replicazione dei blocchi, è il namenode che sceglie i datanode sui quali memorizzare le repliche.
Il namenode seleziona i datanode sulla base di un compromesso tra l'affida-

bilità e la banda a disposizione per la lettura e la scrittura. Se per esempio, tutte le repliche fossero mantenute su un singolo nodo, non si avrebbero problemi di banda in fase di lettura e scrittura, ma al primo malfunzionamento del nodo i dati del blocco replicato andrebbero persi. All'estremo opposto, posizionare le repliche su diversi data center massimizzerebbe l'affidabilità, a discapito però della banda.
La strategia predefinita di Hadoop (vedi Figura2.2) prevede di mantenere la prima replica sullo stesso nodo da cui proviene la richiesta del client; la seconda replica viene posta su un rack diverso dal primo (*off-rack*), scelto a caso; la terza replica viene posta sullo stesso rack della seconda, ma su un'altro nodo, scelto a caso. Ulteriori repliche vengono distribuite casualmente sui nodi del cluster, evitando di posizionare troppe repliche sullo stesso rack.
Nel complesso, la strategia adottata da Hadoop permette di bilanciare l'affidabilità, la larghezza della banda, le performance in lettura e la distribuzione dei blocchi nel cluster.

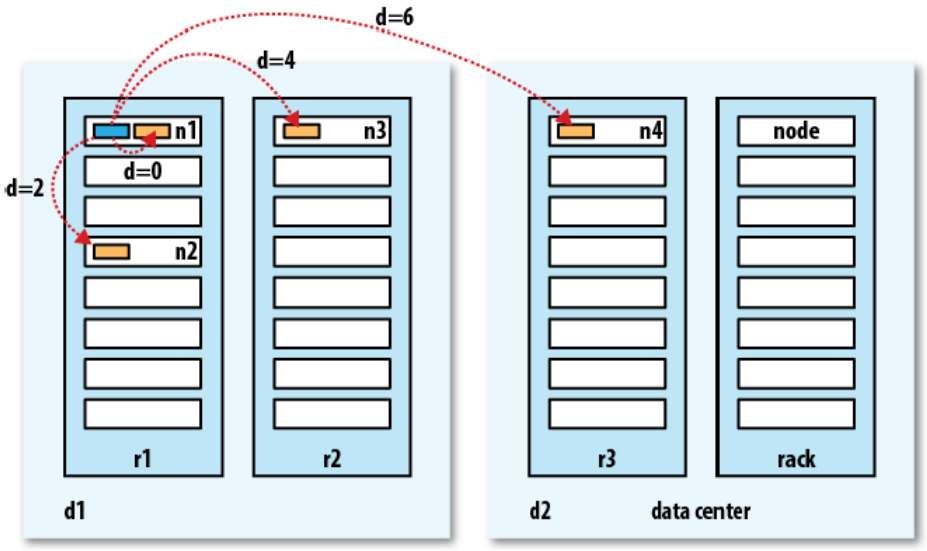


Figura 2.3: Distanze di Rete di Hadoop [21]

Dovendo scegliere i datanode in modo tale da gestire le replicazioni, è necessario che il namenode abbia coscienza della topologia del cluster di

cui fa parte. Solitamente, per misurare la distanza tra due nodi della rete viene utilizzata la langhezza di banda, ma nella pratica risulta una procedura difficoltosa, soprattutto quando si ha a che fare con cluster che possono crescere nel tempo. Hadoop adotta un approccio molto più semplice: rappresenta la rete del cluster come un albero e a seconda dell'organizzazione dei nodi assegna dei valori di distanza in base al livello di appartenenza. I livelli non sono predefiniti, anche se comunemente ne viene assegnato uno al data center, uno al rack e uno al nodo su cui un processo è in esecuzione. A seconda dello scenario che si presenta (vedi Figura2.3), le distanze progressivamente aumentano:

- processo in esecuzione e dati sullo stesso nodo;
- processo e dati su nodi diversi ma sullo stesso rack;
- processo e dati su nodi appartenenti a rack diversi, ma sullo stesso data center;
- processo e dati su diversi data center.

Concettualmente quindi, la larghezza di banda disponibile diventa progressivamente meno, man mano che ci si allontana dal nodo su cui è in esecuzione il processo.

2.3.3 Lettura e Scrittura

HDFS è l'implementazione della concetto astratto di filesystem di Hadoop . La classe astratta Java *org.apache.hadoop.fs.FileSystem* rappresenta l'interfaccia utente del filesystem di Hadoop. Sfruttando l'implementazione di tale interfaccia, un possibile client può interagire in lettura o scrittura con il filesystem.
Nel caso particolare di HDFS, l'istanza che implementa l'interfaccia messa a disposizione da Hadoop è *DistributedFileSystem*. Questa, a seconda dell'operazione che viene richiesta, restitusce un *FSDataInputStream* oppure un *FSDataOutputStream*, flussi dati da o verso il filesystem.
Qui di seguito verranno descritti i passaggi che si verificano tra client, namenode e datanode, generati dalle operazioni di lettura e scrittura richieste dall'utente.

In fase di **lettura**, come mostra la Figura2.4:

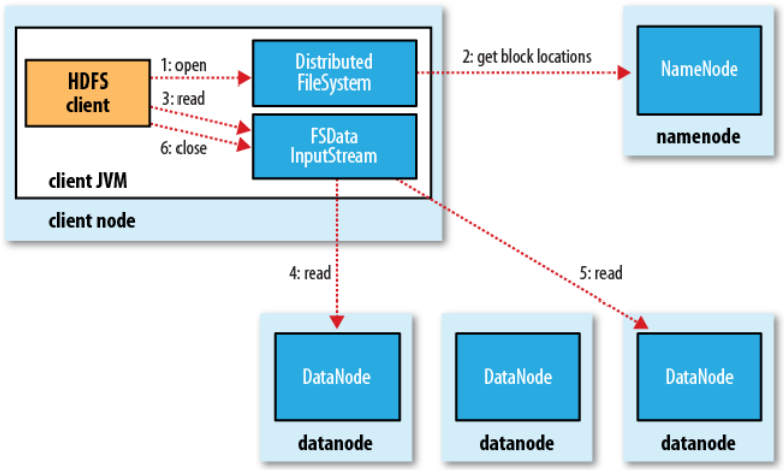


Figura 2.4: Lettura da HDFS [21]

1. il client richiede al DistributedFileSystem, mediante l'operazione *open()*, l'apertura di un file;

2. il DistributedFileSystem a sua volta, richiede al NameNode le posizioni dove risiedono i blocchi del file richiesto dal client. Per ogni blocco, il NameNode restituisce gli indirizzi dei datanode che possiedono una copia di quel blocco, ordinati in base alla vicinanza al client. Il DistributedFileSystem restituisce così al client un FSDataInputStream per la lettura dei dati;

3. l'FSDataInputStream, a seguito della chiamata di read() da parte del client, si collega al primo datanode (quello di vicino) che memorizza il primo blocco del file;

4. tra client e datanode si istaura così un streaming di dati, attraverso il quale il client dovrà ripetutamente lanciare l'operazione di *read()*, fino alla lettura dell'intero blocco;

5. al termina della lettura del blocco, FSDataInputStream provvede a chiudere la connessione con il DataNode, e a cercare il prossimo datanode (più vicino) da cui leggere il blocco successivo. Questo accade in modo trasparente al client, che dal suo punto di vista la lettura viene svolta come un flusso continuo;

6. letti tutti i blocchi relativi al file, il client provvede a chiudere la connessione con lo straming FSDataInputStrem.

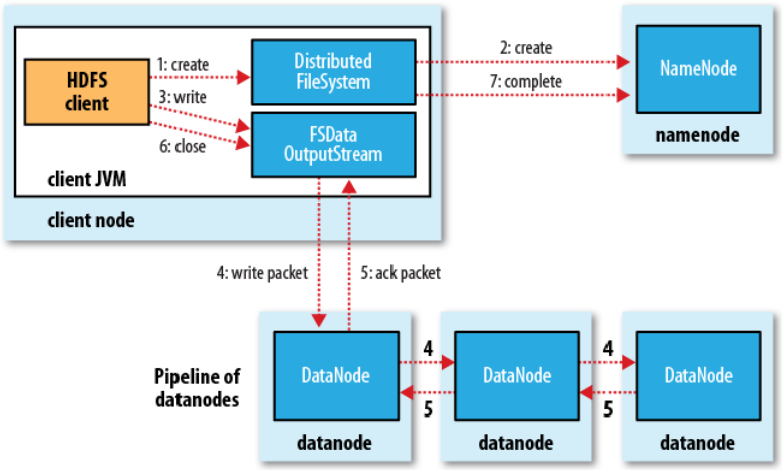


Figura 2.5: Scrittura su HDFS [21]

Dopo aver visto come HDFS gestisce l'operazione di lettura, è opportuno osservare come gestisce l'operazione di **creazione e scrittura** di un file:

1. il client richiede la creazione di un nuovo file tramite l'operazione *create()*;

2. il DistributedFileSystem comunica la creazione di un nuovo file al NameNode, il quale provvederà ad inserirlo nel namespace del filesystem. Così come nel caso della lettura, il DistributedFileSystem restituisce un FSDataOutputStream, attraverso il quale il client può iniziare a scrivere i dati;

3. man mano che il cliente scive i dati, il FSDataOutputStream li suddivide in pacchetti e li posiziona in una *data queue*;

4. i dati all'interno della *data queue* vengono gestiti dal *Data Streamer*, il quale, dopo aver ricevuto dal namenode la lista dei datanode su cui replicare, passa i dati al primo datanode. Questo li memorizza e li inoltra al secondo della lista. Allo atesso modo, il secondo datanode memorizza i pacchetti e li inoltra al terzo (e ultimo) datanode della lista;

5. il FSDataOutputStream al suo interno, mantiene una coda di pacchetti, detta *ack queue*, in attesa di essere "accettati" dai datanode. Un pacchetto viene rimosso dalla *ack queue* solo quando tutti i datanode della lista, fornita dal namenode, lo hanno in memoria;

6. al termine della scrittura il client provvede a chiudere il flusso dati, tramite il comando close();

7. alla chiusura del flusso dati viene segnalato al namenode il completamento della scrittura del file.

HDFS fornisce le funzionalità di memorizzazione e accesso ai dati, le elaborazioni degli stessi però avvengono attraverso la componente di MapReduce descritta nel prossimo paragrafo.

2.4 MapReduce

MapReduce è il cuore del sistema di calcolo distribuito di Hadoop. Rappresenta il framework attraverso il quale è possibile creare applicazioni in grado di elaborare grandi quantità di dati in parallelo su grandi cluster. MapReduce lavora secondo il principio *divid et impera*, ovvero prevede la suddivisione di un'operazione di calcolo in diverse parti processate in modo autonomo. Al termine del calcolo di ciascuna parte, i vari risultati parziali vengono "ricomposti" in un unico risultato finale.

L'applicazione in grado di essere eseguita sull'ambiente Hadoop viene definita come Job MapReduce, composto in generale da quattro elementi:

- dati di *input*;
- una fase di *map*;
- una fase di *reduce*;
- dati di *output*.

Le fasi di map e di reduce, che compongono il Job MapReduce, vengono suddivise in un certo numero di task, ovvero sotto-attività schedulate e gestite da YARN ed eseguite in parallelo sul cluster Hadoop. A seconda dell'attività (di reduce o di map) che verrà svolta, i task vengono classificati come *map task* oppure come *reduce task*.

I singoli task vengono eseguiti sui nodi del cluster adibiti al calcolo. Tipicamente all'interno di un cluster, i nodi adibiti al calcolo e i nodi di storage sono gli stessi, infatti MapReduce e HDFS condividono lo stesso insieme di nodi. Questa configurazione permette al framework di organizzare le attività in modo tale da ridurre la quantità di banda utilizzata per il trasferimento dei dati. MapReduce, infatti, utilizza un meccanismo, chiamato *data locality optimization*, che permette di allocare, in modo efficiente, i task sui nodi dove risiedono i dati necessari alla computazione.

In generale, l'esecuzione di un Job viene presa in carico dall'architettura MapReduce, caratterizzata da cinque entità indipendenti:

- il **client**, che richiede l'esecuzione del Job MapReduce;
- il **ResourceManager**, che coordina l'allocazione delle risorse computazionali per ogni singolo container del cluster.
- il **NodeManager**, che lancia e monitora la computazione dei container sul nodo del cluster;
- l'**ApplicationMaster**, che coordina i task di map e di reduce del Job MapReduce. In particolare l'ApplicationMaster ha il compito di negoziare con il ResourceManager le risorse necessarie per l'allocazione dei container, sui quali verranno eseguiti, uno per volta, i singoli task. Inoltre coopera con i NodeManager per eseguire e monitorare i task in esecuzione;
- l'**HDFS**, utilizzato per la condivisione dei file fra le entità.

Per ogni client, che richiede l'esecuzione di un Job MapReduce, viene allocato un ApplicationMaster, responsabile della specifica applicazione richiesta. Diversamente accade per il ResourceManager e per i NodeManager, perché non dipendono strettamente dalle applicazioni in esecuzioni, ma dipendono dalla struttura del cluster Hadoop. Ogni cluster infatti, è caratterizzato da un ResourceManager e da un NodeManager per nodo.

Di seguito verrà descritto cosa avviene l'esecuzione di un job mapreduce, soffermandosi sulle fasi di map e di reduce che lo caratterizzano e sui processi che legano tali elementi.

2.4.1 Flusso di esecuzione

Descrivere e capire come si svolge il flusso di esecuzione di un job mapreduce (vedi Figura2.6) è importante per comprendere come progettarne e svilupparne il codice e per migliorarne le performance di esecuzione.

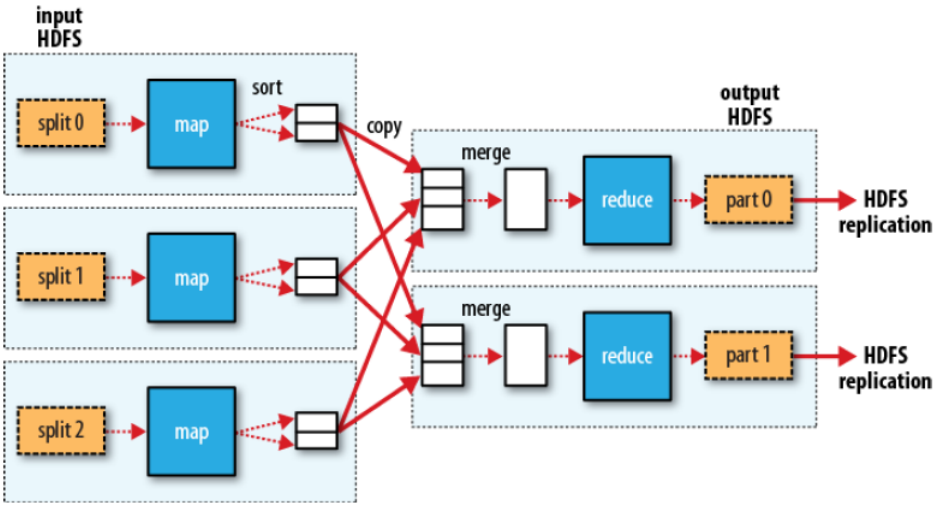


Figura 2.6: Flusso di Esecuzione di un Job MapReduce [21]

All'avvio della computazione, l'input del Job viene suddiviso in porzioni di dimensioni fisse, chiamate *splits*. La dimensione di questi, di default è pari a 128MB, ma può essere ridimensionata dall'utente. Suddiviso l'input, l'ApplicationMaster provvede alla creazione di tanti map task quanti sono gli split ottenuti dalla suddivisione; di questi ne alloca tanti quanti sono i container a disposizione per l'esecuzione. I container provvedono all'esecuzione di un singolo task per volta. Ma mano che la computazione di ogni singolo task termina, l'ApplicationMaster provvede ad assegnare dinamicamente, ai container liberi, i task ancora in sospeso per l'esecuzione.

Terminata la computazione, ogni singolo map task restituisce un output composto da una coppia chiave/valore, memorizzata sul disco locale del nodo e ordinata rispetto le altre coppie chiave/valore presenti. I map task non scrivono direttamente su HDFS, perché forniscono output intermedi, che dovranno poi essere processati dai task di reduce e cancellati al termine dell'applicazione. La scelta implementativa di scrivere gli output intermedi sul disco locale dei singoli nodi, evita l'innesco delle procedure

di replicazione non necessarie, perché si ha a che fare con dati temporanei di computazione e non con dati persistenti da mantenere in memoria anche dopo la terminazione dell'applicazione.

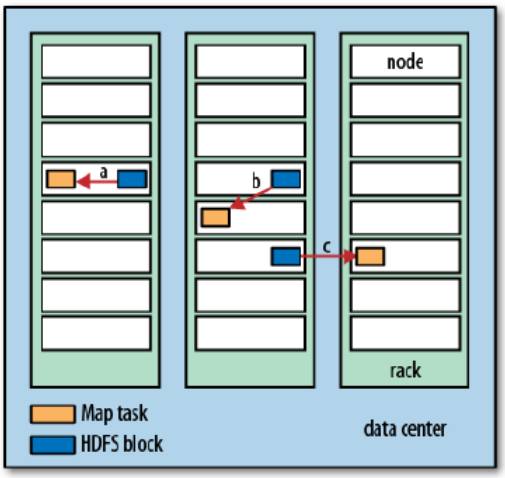


Figura 2.7: Allocazione Map Task [21]

I task di map, diversamente da quelli di reduce, traggono benefici dal meccanismo di *data locality* di Hadoop; infatti in fase di allocazione dei map task, l'ApplicationMaster tiene in considerazione la posizione dei dati necessari alla computazione, cercando così di minimizzare il più possibile l'utilizzo della banda. A seconda dei container disponibili per l'esecuzione, l'ApplicationMaster potrebbe trovarsi di fronte a tre diversi possibili scenari (vedi Figura2.7), ognuno dei quali caratterizzato da priorità diversa; infatti, si cerca di privilegiare le allocazioni che posizionano i task il più vicino possibile ai dati necessari all'input.

I reduce task, invece, non hanno la possibilità di sfruttare i vantaggi offerti dalla data locality, perchè ogni singolo reduce, indipendentemente dal nodo su cui è in esecuzione, viene alimentato dall'output di molti map task. Ogni map task infatti, *partiziona* il proprio output, creando una partizione per ogni reduce task. All'interno di queste partizioni possono essere presenti chiavi diverse, ma output caratterizzati dalla stessa chiave vengono posizionati nella stessa partizione. Al termine della computazione di map, le partizioni generate vengono prelevate dai reduce task per lo svolgimento delle propria computazione. Questi elaborano l'input, proveniente dalla

fase di map, sulla base delle chiavi. L'output ottenuto dalla computazione dei reduce, viene poi memorizzato su HDFS e contestualmente replicato secondo la strategia di replicazione di HDFS.

Il numero di reduce task, diversamente da quello dei map, non dipende dalla dimensione dell'input che si intende elaborare, ma viene specificato "dall'esterno". Durante l'avvio del job, sfruttando le apposite interfaccie, è possibile definite il numero di reduce task da allocare. I numero di map task invece, viene scelto internamente dal framework, in base alla dimensione dell'input del job.

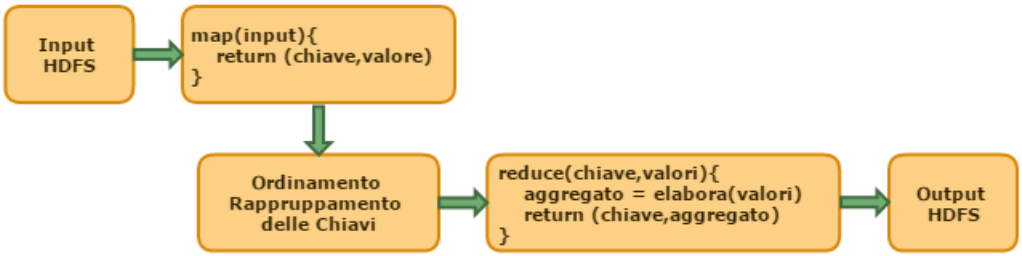


Figura 2.8: Flusso Dati MapReduce

La Figura2.8 mostra il comportamento generale, descritto in precedenza, del flusso dati che si verifica nel corso dell'esecuzione di un Job MapReduce. Le attività che il framework svolge a seguito della map e prima della reduce, fanno parte della fase di *Shuffle*, descritta più nel dettaglio qui di seguito.

Fase di Shuffle

Il framework MapReduce è stato progettato in modo tale da garantire che l'input di ogni reduce sia ordinato in base alla chiave. Il processo attraverso il quale il sistema esegue l'ordinamento e il trasferimento dell'output dei map a input dei reduce è detta *Shuffle*. Questa coinvolge sia la fase terminale del processo di map, che la fase iniziale del processo di reduce (vedi Figura2.9).

L'output prodotto dalla fase di map non viene semplicemente scritto sul disco locale. Ogni map task è provvisto di un buffer circolare (di default 100MB) su cui viene riposto l'output ottenuto dall'elaborazione di map. Quando il contenuto del buffer raggiunge una determinata soglia (di default l'80%) un processo in background provvede a riversare il contenuto

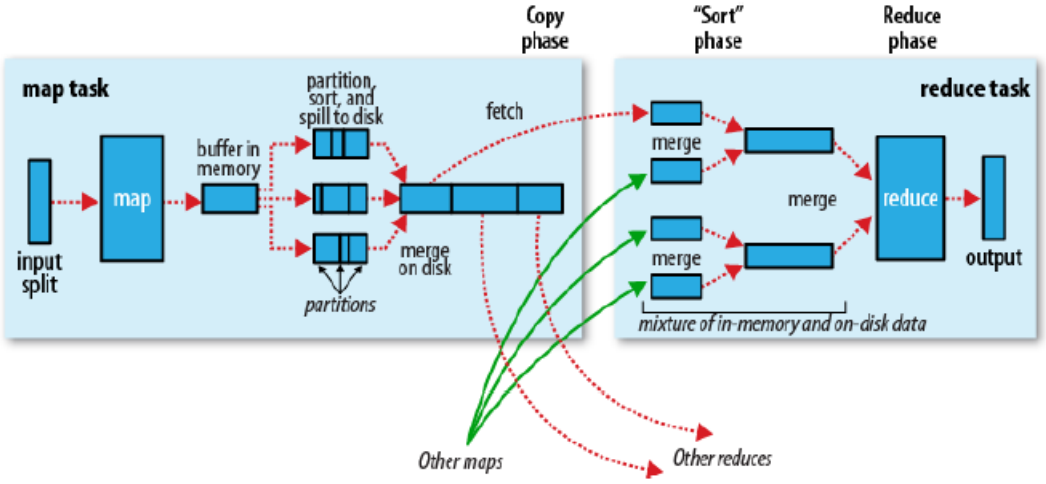


Figura 2.9: Fase di Shuffle [21]

del buffer sul disco locale, ovvero esegue un'operazione di *spill*. Se durante la computazione il buffer dovesse riempirsi, il map task si bloccherebbe in attesa che il processo termini la propria attività di spill. Il processo in background che esegue l'operazione di spill, all'avvio, suddivide i dati in partizioni, una per ogni reduce task. Ognuna della partizioni, potendo contenere valori con chiavi diverse, viene progressivamente ordinata per chiave, dal processo che si occupa dello spill dei dati. Al termine del map task, le partizioni degli output vengono uniti in una singola partizione ordinata, resa poi disponibile ai reduce task.

Terminata la fase di map, nel disco locale di ogni macchina, su cui è stato eseguito un task di map, è possibile individuare la partizione totale dell'output ottenuto dall'elaborazione. Queste partizioni vengono utilizzate dai reduce task come input della computazione. I map task non terminano l'elaborazione tutti nello stesso momento, vi sono task che terminano prima di altri; per questo motivo ogni singolo reduce task, al completamento di ogni map task, provvede a prelevare le partizioni di output, da utilizzare come input. Questa è conosciuta come *fase di copy* di ogni singolo reduces: insieme di processi in parallelo che copiano le partizioni di output dei map task. Al termine della copia di tutti gli output, il reduce task passa alla *fase di sort*, dove le partizioni copiate vengono ordinate ed unite in un unico grande file, che va ad alimentare direttamente la funzione di reduce,

in quella che viene detta la *fase di reduce*. Durante tale fase, la funzione di reduce viene invocata per ogni chiave distinta passata in input, e restituisce un output che viene direttamente scritto su HDFS.

La fase di Shuffle appena descritta e le relative fasi di Map e di Reduce rappresentano il cuore dell'esecuzione di un Job MapReduce. Nella sezione successiva invece, viene descritta l'avvio dell'esecuzione di un Job MapReduce e l'architettura YARN che regola l'intera applicazione.

2.4.2 YARN

Il passaggio da Hadoop 1.x a Hadoop 2.x, come mostra la Figura2.10, è stato segnato dall'introduzione di YARN (Yet Another Resource Negotiator), il nuovo sistema di gestione delle risorse del cluster Hadoop. YARN è stato introdotto in Hadoop con l'obiettivo si migliorarne le performance di elaborazione. È un sistema molto più generico rispetto a MapReduce,

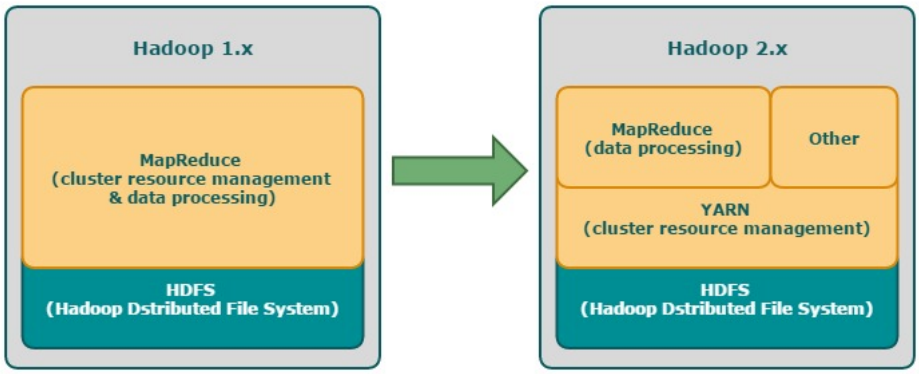


Figura 2.10: Hadoop 1.x vs. Hadoop 2.x

nelle sue versioni precedenti. Mette a disposizione API per la richiesta e la gestione di risorse del cluster che, non sono in genere utilizzate direttamente dal codice utente. In questa versione di "nuova generazione", MapReduce è stato scritto come un'applicazione YARN.

Nella versione precedente di MapReduce, le due componenti principali, che ne caratterizzavano l'architettura, erano il JobTracker e il TaskTracker. La nuova architettura messa a disposizione da YARN, sostituisce le com-

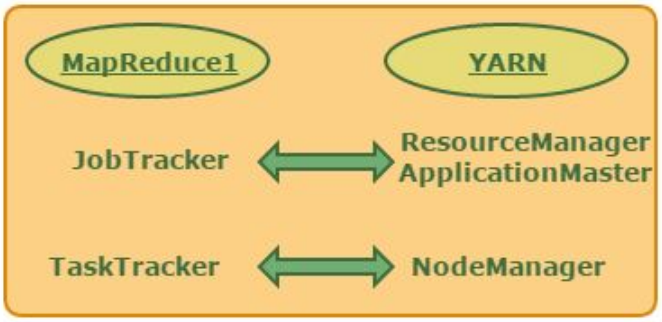


Figura 2.11: MapReduce1 vs. YARN

ponenti di JobTracker e di TaskTracker con quelle di **ResourceManager**, **ApplicationMaster** e di **NodeManager** (vedi Figura2.11). Anche se con qualche aspetto in più, le attività svolte dai TaskTracker vengono ora eseguite dai NodeManager, invece le attività destinate al JobTracker vengono ora suddivise fra il ResourceManager e l'ApplicationMaster. Dal un lato il ResourceManager si occupa di gestire l'uso delle risorse del cluster, invece i NodeManager si occupano di lanciare e monitorare i containers, ovvero componenti che eseguono processi specifici dell'applicazione in esecuzione sul framework, a cui viene riservato un certo insieme di risorse del cluster (memoria, CPU, ecc).

Esecuzione di un'Applicazione YARN: MapReduce

Ogni volta che si vuole eseguire un'applicazione su YARN, occorre che il client contatti il ResourceManager richiedendo l'esecuzione di un processo *ApplicationMaster*. Il ResourceManager provvede ad individuare il NodeManager sul quale verrà lanciata l'esecuzione dell'ApplicationMaster. Ciò che l'ApplicationMaster svolge dipende strettamente dall'applicazione MapReduce che si sta eseguendo. Un esempio classico di applicazione YARN è MapReduce (vedi Figura2.12).

L'avvio di un'applicazione MapReduce è possibile grazie alla semplice chiamata del metodo *submit()* sull'oggetto *Job* (*Step 1*), messo a disposizione dalle API di Hadoop. Tale chiamata innesca tutta una serie di attività, descritte qui di seguito, necessarie affinchè il job possa essere ese-

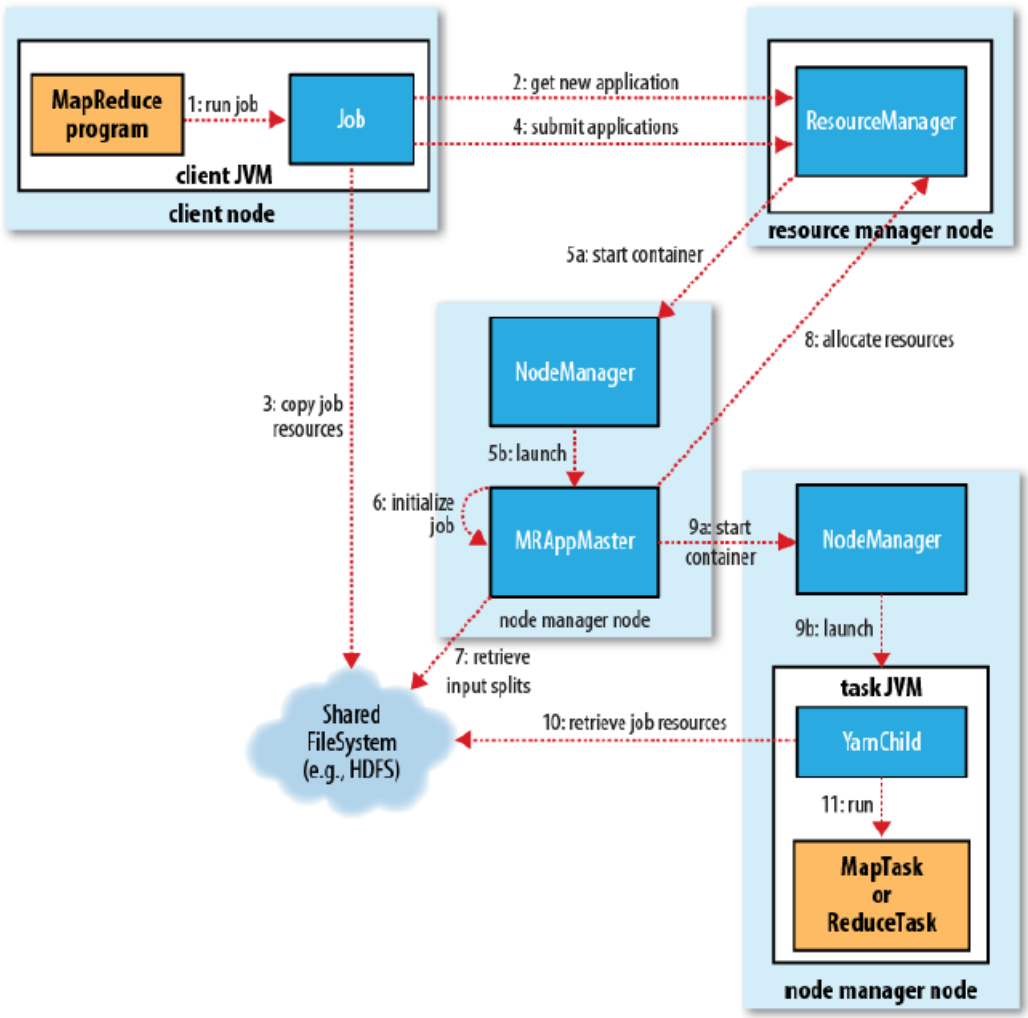


Figura 2.12: MapReduce: Applicazione YARN [21]

guito. Queste attività possono essere raggruppate in cinque macro-insieme, ognuno delle quali rappresenta una delle fasi dall'evoluzione del Job:

1. **Job Submissing**
   Inizialmente al ResourceManager viene richiesto un nuovo application ID da associare al job che si sta per eseguire (*Step 2*). L'input del job viene suddiviso in slip di dimensioni fisse e, contestualmente, viene controllata l'esistenza della directory di output specificata dal job. Quest'ultima infatti non deve esistere, se così non fosse, il job interromperebbe la propria esecuzione. Tutte le risorse necessarie alla computazione (come ad esempio i file JAR, le configurazioni, ecc) vengono distribuite sul file system, in una directory nominata esattamente come l'application ID (*Step 3*). Fatto tutto ciò, al ResourceManager viene confermato l'avvio del job (*Step 4*), che sarà poi preso in carico dallo scheduler di YARN;

2. **Job Initialization**
   Lo scheduler si occupa di allocare un container, sul quale il ResourceManager provvederà ad avviare l'application master MRAppMaster, sotto la gestione del NodeManager (*Step 5a* e *5b*). MRAppMaster, la main class dell'applicazione Java che implementa il job MapReduce, inizializza il job, creando una serie di oggetti che tengono traccia dei progressi e del completamento del job (*Step 6*); dopo di che, si occupa di recuperare, dal filesystem, gli split di input e per ognuno di questi provvederà alla creazione dei map task (*Step 7*). Il numero di questi, è pari al numero di split a disposizione, invece, per i reduce, il numero di task dipende strettamente da ciò che viene indicato nella proprietà *mapreduce.job.reduces*, oppure all'interno dell'ApplicationMaster, mediante il metodo $setNumReduceTasks()$ dell'oggetto *Job*. In particolare, l'ApplicationMaster ha il compito di valutare la gestione dei task del Job MapReduce: se il job risulta essere di "piccole dimensioni", l'application master potrebbe decidere di eseguire localmente su un solo nodo i task. Questo accade quando si considera che l'overhead dovuto all'assegnazione e all'esecuzione dei task su nuovi container, supera il guadagno dato dall'esecuzione in parallelo, rispetto all'esecuzione sequenziale su un solo nodo. Job di questo tipo vengono detti *uberized*. Di default, questi job sono caratterizzati da

meno di 10 mapper, un solo reducer e la dimensione dell'input risulta essere inferiore alla dimensione di un blocco HDFS;

3. **Task Assignment**
   Nel momento in cui il job non viene qualificato come *uberized*, l'Application Master provvederà a richiedere al Resource Manager i container necessari all'esecuzione dei task del job (Step 8). Il numero di container messo a disposizione, non dipende dal numero di map task definiti dall'Application Master, ma dipende dalle risorse disponibile del cluster. Le richieste dei container, relativi ai map task, vengono fatte prima e con priorità maggiore rispetto a quelli dei task di reduce. Inoltre, la richiesta dei cointainer per i reduce task, non viene fatta finché il 5% dei map task non sono effettivamente completati. I reduce task possono essere eseguiti indipendentemente su qualsiasi nodo del cluster, cosa non vera per i map task, per i quali lo scheduler cerca di rispettare il vincolo di data locality. Assieme alle richieste dei container inoltre, vengono specificati i requisiti di memoria e CPU necessari ai task. Di default, per ogni attività di map e di reduce vengono allocati 1024MB di memoria e un core virtuale, ma questi parametri possono anche essere modificati, distinguendo i task relati ai map e quelli relativi ai reduce.

4. **Task Execution**
   Assegnato il container e le relative risorse, ad un particolare nodo del cluster, l'Application Master provvede ad avviare il container contattando il NodeManager (*Step 9a* e *9b*). Il task viene eseguito da un'applicazione Java, la cui classe principale è YarnChild. Prima dell'esecuzione però, vengono localizzati e recuperati i file JAR e le configurazioni eventualmente necessarie alla computazione (*Step 10*). Fatto ciò il nodo procede all'esecuzione del task di reduce o di map (*Step 11*).
   In generale i Job MapReduce sono processi batch che, a seconda di ciò che devono svolgere, possono impiegare qualche decina di secondi a qualche ora per completare l'esecuzione. Dato che tale periodo di tempo può risultare significativo, occorre che l'utente abbia percezioni dello stato di avanzamento del job. Per questo motivo il job e ognuno dei suoi task sono caratterizati da uno *status*, che ne descrive lo stato

di avanzamento (per esempio, *running*, *successfully completed*, *failed*, ecc). Questi stati vengono modificati durante l'esecuzione del job.

5. **Job Completion**
   Quando l'ApplicationMaster riceve la notifica che l'ultimo task del job è stato completato, cambia lo stato principale del job in "successful", dualmente viene mostrato a video un messaggio, che permette di avvisare l'utente del completamento del job. Al termine del job, l'ApplicationMaster e i container allocati per i task ripuliscono il loro stato, utile per esempio per eliminare l'output intermedio prodotto dai map task. Infine, tutte le informazioni relative al job eseguito, vengono archiviate cronologicamente, per consentire all'utente visualizzazioni successive.

Come è possibile notare dall'architettura appena descritta, l'introduzione di YARN ha portato notevoli cambiamenti dell'architettura di MapReduce, i componenti che la caratterizzano sono cambiati. Nelle versioni precedenti di Hadoop, MapReduce sfrutta un JobTracker e uno o più TaskTracker, come componenti in grado di controllare e gestire il Job. Il JobTracker coordina i job in esecuzione sul sistema e funge da scheduler per i task in esecuzione sui TaskTracker. Questi infatti, eseguono i task di map e di reduce ed inviano il relativo stato di avanzamento al JobTracker, responsabile di monitorare lo stato generale del job. Se un task fallisce, il JobTracker ha la capacità di ri-schedulare l'attività su un'altro TaskTracker. Quindi in generale, in Hadoop 1.x, il JobTracker ha il duplice compito di Job Scheduler (abbinare ai TaskTracker i task di map o di reduce) e di Task Monitoring (tenere traccia dei task, ri-esecuzione dei task falliti, ecc). Diveramente, in YARN tali responsabilità vengono suddivise fra il ResourceManager e l'Application Master (uno per ogni Job MapReduce). Invece per quanto riuarda il TaskTracker, anch'esso non presente nella nuova versione, viene "sostituito" dal NodeManager.

YARN è infatti stato progettato per supportare le molte limitazione di MapReduce nella sua vecchia versione. L'architettura progettata, rispetto a quella precedente, presenta diversi vantaggi:

- YARN può essere eseguito su cluster di dimensioni maggiori rispetto a quelli su cui è in esecuzione la precedente versione di MapReduce; questo deriva dal fatto che il JobTracker deve gestire contemporaneamente sia il job che i task. Invece nella versione attuale, ogni istanza

di un'applicazione MapReduce ha un Application Master dedicato, attivo per tutta la durata dell'applicazione;

- In Hadoop 1.x, ogni TaskTracker è caratterizzato da "slot" statici, di dimensioni fisse, definite in fase di configurazione. Alcuni di questi sono utilizzati solo per l'esecuzione di map task, altri solo per l'esecuzione di reduce task. YARN, invece, gestisce un "pool" di risorse, che vengono assegnate dinamicamente dall'ApplicationMaster, piuttosto che in maniera statica, in fase di configurazione. Questo permette di sfruttare al meglio le risorse messe a disposizione dal cluster per la computazione;
- Infine, l'introduzione di YARN ha permesso di rendere disponibile l'ambiente di Hadoop, oltre che a MapReduce, anche ad altri tipi di applicazioni distribuite. MapReduce è una delle applicazioni YARN esistenti. Grazie alle API messe a disposizione dalla nuova versione di Hadoop, sono molte le applicazioni che possono essere implementate come applicazioni YARN.

Dopo aver descritto e osservato i meccanismi alla base dell'architettura di Hadoop e le logiche di esecuzione di applicazioni MapReduce è possibile, nel paragrafo successivo, osservare una panoramica degli strumenti che sfruttano tali architetture per l'analisi e la gestione dei dati.

2.5 L'Ecosistema di Hadoop

Al nucleo centrale di Hadoop, si aggiungono tutta una serie di altri componenti software che vanno a completare quello che può essere definito l'**Ecosistema di Hadoop**. Tutti i principali e più utilizzati sistemi dell'ecosistema di Hadoop sono stati sviluppati all'interno di Apache Foundation. Alcune di queste componenti, come viene mostrato in Figura2.13, sono a supporto del nucleo principale di Hadoop. Fra questi è possibile individuare gli strumenti **Flume** e **Sqoop**, già introdotti nel capitolo precedente. Questi fanno parte degli strumenti dell'ecosistema di Hadoop, che svolgono compiti importanti ma non essenziali al fine del funzionamento della piattaforma. Oltre a Flume e Sqoop, è possibile individuare:

- **Zookeeper**: strumento che fornisce un servizio centralizzato al fine di sincronizzare gli "oggetti" comuni nel cluster, come per esempio

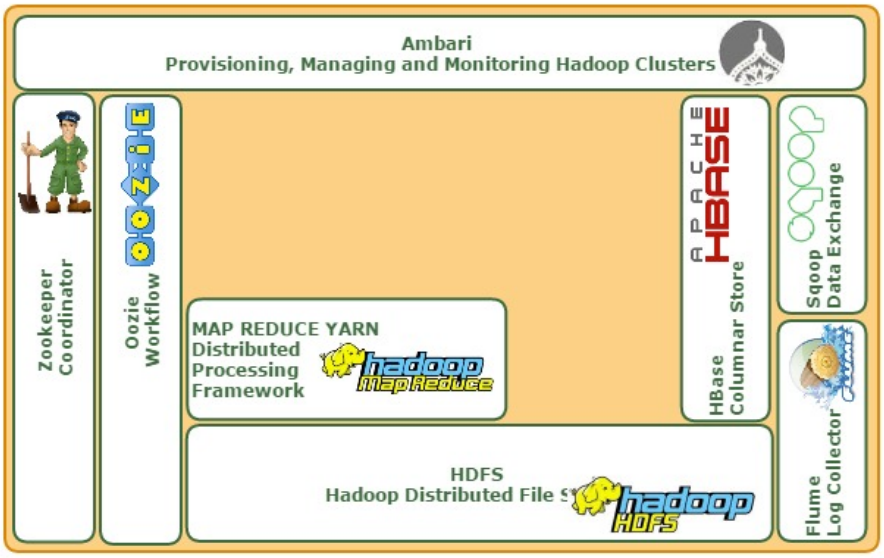


Figura 2.13: Ecosystem 1

le informazioni di configurazione presenti su tutti i nodi. Zookeeper ha l'obiettivo di implementare tutti quei servizi di sincronizzazione distribuita, che a causa della loro complessità di implementazione, vengono inizialmente trascurate nelle applicazioni.

- **Oozie**: è un motore di *workflow* specializzato nella schedulazione dei Job MapReduce. Un workflow non è altro che un insieme di azioni organizzate secondo un grafo DAG (*Directed Acyclic Graph*), secondo il quale nessuna azione può essere eseguita se quella precedente non è ancora terminata. I workflow gestiscono l'esecuzione dei Job MapReduce in remoto, e al termine di questi il sistema da remoto notifica a Oozie il completamento dell'azione, in modo tale che il workflow possa proseguire con l'azione successiva.

- **Ambari**: è lo strumento per la gestione e il monitoraggio dell'interno cluster Hadoop. Fornisce un'interfaccia web attraverso la quale è possibile svolgere le operazioni amministrative, e una dashboard per la visualizzazione dello "stato di saluto" del cluster. Inoltre permet-

te di visualizzare le performance di esecuzione di MapReduce, e dei principali strumenti di data access.

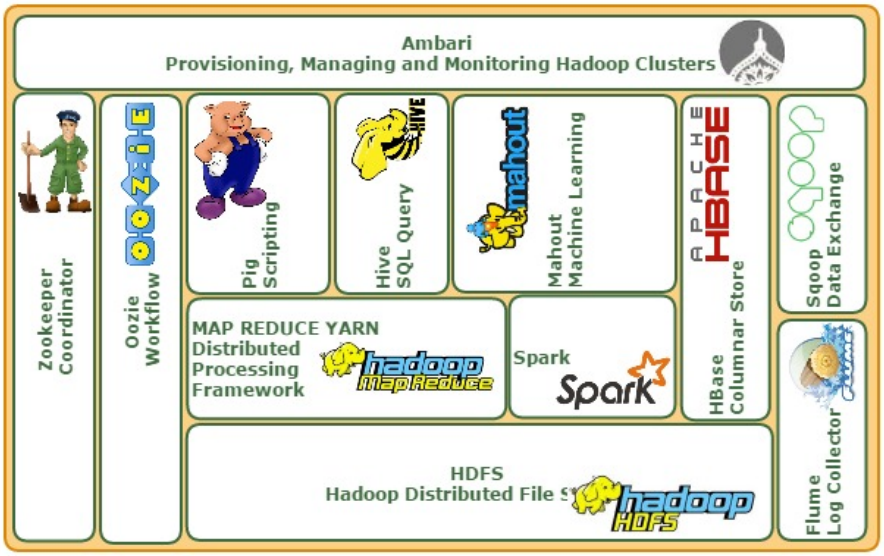


Figura 2.14: Ecosystem 2

Altri strumenti (vedi Figura2.14), diversamente da quelli appena descritti, sfruttano direttamente le funzionalità offerte dal core di Hadoop per svolgere attività di elaborazione ed accesso ai dati:

- **Pig**: è una piattaforma per l'analisi di grandi quantità di dati, caratterizzata da un linguaggio di alto livello, chiamato Pig Latin, per la gestione e l'elaborazione dei flussi dati. L'infrastruttura di Pig consiste in un compilatore che, a partire da istruzioni scritte in Pig Latin, produce una sequenza di applicazioni MapReduce. Rispetto all'utilizzo diretto di MapReduce, Pig risulta essere:
  - sintatticamente più semplice
  - capace di ottimizzare l'esecuzione delle sequenze di operazioni
  - estensibile, perchè utilizzando le proprio librerie, può essere arricchito creando file Jar da importare ed utilizzare all'interno degli script.

- **Hive**: è un software di data warehouse, in grado di facilita l'esecuzione di query e la gestione di grandi quantità di dati su repository distribuiti. Hive utilizza HiveQL, un linguaggio SQL-Like, per la gestione e l'interrogazione dei dati. Nonostante la sua somiglianza con SQL, Hive rimane un sistema per la generazione di job MapReduce che elaborano grandi moli di dati, e non uno strumento in grado di rimpiazzare SQL e i database relazionali nell'esecuzione di attività transazionali.

- **Spark**: è un framework di calcolo su cluster per l'elaborazione di dati su larga scala. Diversamente dalla maggior parte delle componenti fin'ora descritte, Spark non usa MapReduce come motore di esecuzione, ma nonostante ciò presenta molte somiglianze, sia in termini di API che per l'esecuzione. Spark è conosciuto per l'*in-memory caching*, ovvero la capacità di mantenere i dati in memoria tra un Job e l'altro. A parità di workflow, questa capacità consente a Spark di avere performance migliori rispetto a MapReduce, dove i dati sono sempre caricati su disco. Molte applicazioni possono beneficiare delle migliori performance che Spark mette a disposizione, in particolare, gli algoritmi iterativi (funzione applicata più volte a un set di dati fino a quando non viene soddisfatta una certa condizione di terminazione) e l'analisi interattiva (analisi esplorativa ad hoc su un certo set di dati).

- **Mahout**: costituisce un insieme di librerie specifiche per il *scalable machine learning*. Il termine "scalable" si riferisce principalmente agli algoritmi di clustering, classification, collaborative filtering implementati sulla base di sistemi distribuiti scalabili. I primi sviluppi di Mahout sfruttavano il framework di MapReduce per l'elaborazione dei dati. Nel corso dell'ultimo anni però, le nuove implementazioni di Mahout stanno a poco a poco abbandonando il framework di MapReduce per quello di Spark, in grado di fornire una modello di programmazione più ricco e un'esecuzione più efficiente.

Vi sono molti altri componenti che fanno parte dell'ecosistema di Hadoop, nonostante ciò, non essendo argomento del presente elaborato, non sono stati trattati perché meno utilizzati e popolari rispetto a quelli descritti.

2.6 Le Distribuzioni di Hadoop

Negli ultimi anni sempre più vendor sfruttano il core di Hadoop al fine di realizzare distribuzioni proprietarie o open source. I primi e più affermati "distributori" di Hadoop sono **Cloudera** e **Hortonworks**, inseguiti da vicino da **Amazon EMR** e da **MapR** che sin dall'inizio hanno puntato allo sviluppo di soluzioni appositamente per ambienti Cloud. Diversamente, Cloudera e Hortonworks hanno sviluppato inizialmente distribuzioni per ambienti in-house, e solo in seguito queste soluzioni sono state adattate per essere supportate dagli ambienti Cloud.

Prima di entrare nel dettaglio dei servizi offerti dai vendor appena citati, occorre valutare i benefici e gli svantaggi derivanti dall'utilizzo di una soluzione Hadoop su cluster in Cloud piuttosto che su cluster in-house. Le soluzioni Hadoop su Cloud offrono indiscutibilmente dei vantaggi percepibili che mostrano l'importanza di tale scelta:

- L'istallazione del cluster Hadoop è già presente sull'ambiente;
- Le operazioni di manutenzione hardware o software sono a carico del fornitore del servizio;
- I costi iniziali per l'avvio dell'impianto sono trascurabili rispetto a quelli previsti per l'acquisto di hardware;
- I costi previsti per le soluzioni Cloud solitamente sono proporzionali all'utilizzo di risorse (memoria, CPU, ecc);
- Ultima e non meno importante è la scalabilità, che in ambienti cloud viene gestita in maniera trasparente agli occhi dell'utente.

Tutti questi aspetti consentono l'utilizzo di soluzioni Cloud basate su Hadoop senza preoccuparsi dell'aspetto economico o gestionale. Vi sono aspetti però, come la fase di caricamento dei dati sul sistema remoto, la geolocalizzazione e la sicurezza dei dati, che rappresentano delle problematiche importanti che impediscono a molte aziende di adottare soluzioni Cloud.

2.6.1 Cloudera

Cloudera è una società di software americana che fornisce una delle distribuzioni di Apache Hadoop più utilizzate: **CDH** (*Cloudera Distribution Including Apache Hadoop*).

CDH è una piattaforma flessibile, scalabile e integrata che rende semplice la gestione e la memorizzazione di grandi quantità di dati di diversi formati. Basandosi sul core di Hadoop, consente di implementare e gestire progetti MapReduce e correlati. Infatti, da un lato fornisce gli elementi fondamentali di Hadoop (storage, scalabilità e calcolo distribuito), dall'altro presenta un'insieme di interfacce Web utente, tramite le quali è possibile gestire in maniera "user-friendly" le funzionalità offerte dalla distribuzione.

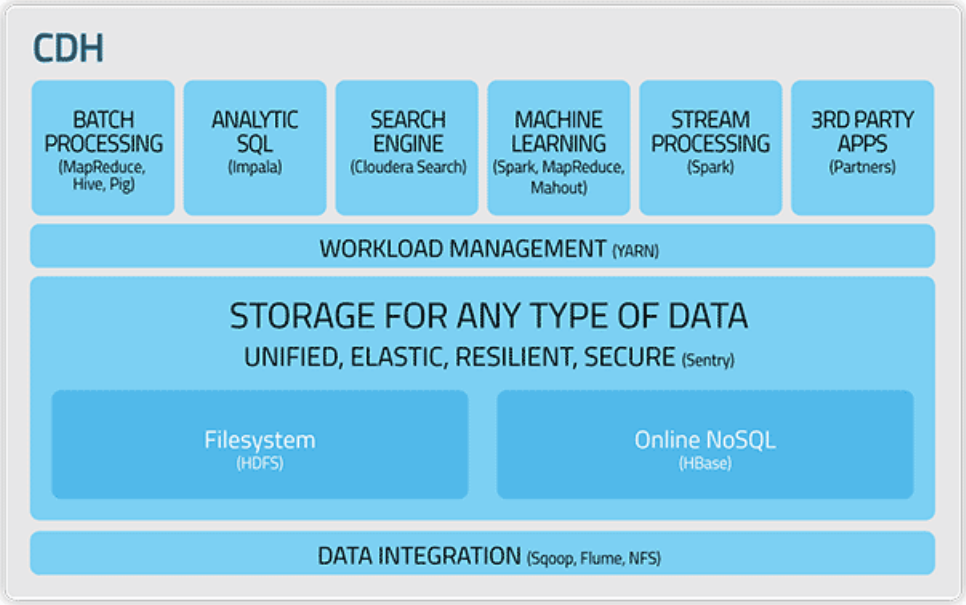


Figura 2.15: Architettura CDH [25]

La figura Figura2.15 mostra le principali funzionalità integrate all'interno della piattaforma CDH di Cloudera. Come è possibile osservare, CDH oltre ad integrare le principali e più utilizzate componenti di Hadoop (Hive, Pig, Spark e Mahout), prevede l'utilizzo di Impala e Cloudera Search, due componenti sviluppate interamente da Cloudera; queste sfruttano i vantaggi offerti dall'architettura del cluster e di Hadoop per gestire ed elaborare dati.

Grazie agli strumenti di Cloudera Manager e Cloudera Navigator, sviluppati e resi disponibile da Cloudera, è possibile gestire in maniera flessibile e immediata la piattaforma CDH, l'intero cluster e l'esplorazione dei dati pre-

senti su Hadoop. La possibilità di sfruttare strumenti dotato di interfaccia grafica, per la gestione e la manutenzione del cluster, l'elaborazione dei dati e i relativi risultati, riduce notevolmente la complessità legata all'ambiente di Hadoop.

2.6.2 Hortonworks

Un'altra società che ha basato il proprio business sullo sviluppo di una soluzione avente alla base il core di Hadoop è Hortonworks. Questa ha sviluppato HDP (Hortonworks Data Platform) una piattaforma distribuita open source basata su Hadoop. Tutti i componenti integrati nella piattaforma provengono dallo stack di progetti di Apache. Diversamente da Cloudera, la piaffatorma offerta da Hortonworks è completamente open-source. Come mostra la Figura2.16, HDP mette a disposizione l'insieme

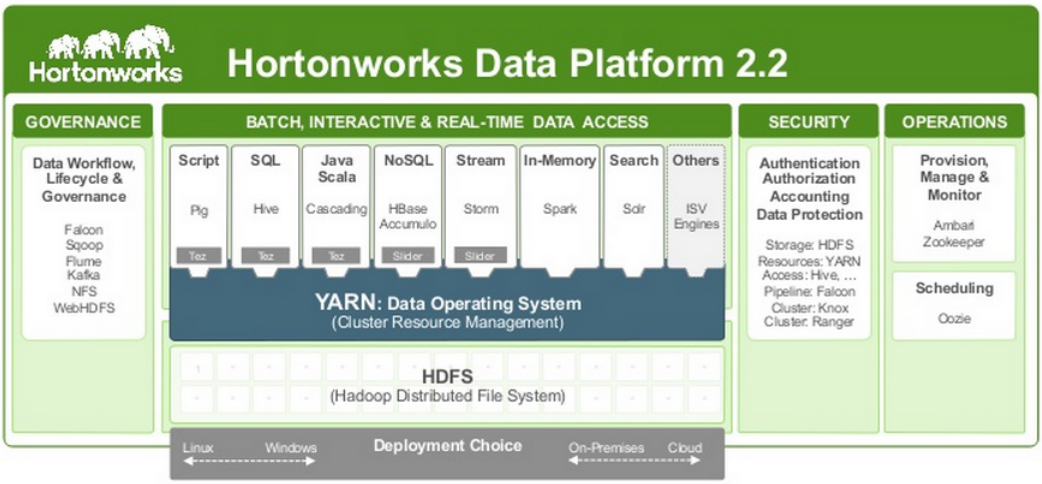


Figura 2.16: Architettura HDP [26]

delle funzionalità e strumenti della suite di Hadoop necessari alla gestione dell'architettura e dell'elaborazione dei dati. Questo insieme di strumenti viene categorizzato secondo cinque macro aree:

- **Data Management**: rappresenta il cuore centrale dell'architettura di HDP, caratterizzato dal nucleo principale di Hadoop, ovvero HDFS e YARN.

- **Data Access**: YARN rappresenta la base di tutta una serie di strumenti per l'elaborazione e l'interazione con i dati. A seconda dello strumento che si andrà ad utilizzare, le modalità di interazione ed elaborazione sono diverse, questo consente una più ampia scelta in fase di sviluppo dell'applicazione.

- **Data Governance and Integration**: HDP estende le funzionalità di gestione e accesso ai dati integrando una serie di strumenti per la governance e l'integration. Ovvero, mette a disposizione tutta una serie di componenti per la gestione dei flussi dati da o verso Hadoop. La gestione dei flussi dati è un aspetto importante, perchè da questo dipende il successo per una corretta integrazione di Hadoop con i tradizionali sistemi di data warehouse.

- **Security**: Per poter garantire l'autenticazione, l'autorizzazione, la responsabilità e la protezione dei dati, HDP prevede un insieme di meccanismi di sicurezza innestati sui vari livelli architetturali.

- **Operations**: HDP offre un insieme completo di funzionalità e strumenti in grado di fornire una visibilità completa sullo stato di saluto del cluster, sulla gestione delle configurazioni e le performance di esecuzione.

2.6.3 Amazon EMR

Amazon Elastic Map Reduce (Amazon EMR) è un servizio web, offerto da Amazon, per l'elaborazione efficiente di grandi quantità di dati. Amazon EMR combina il framework di Hadoop con diversi altri servizi per svolgere operazioni come l'indicizzazione delle pagine web, data mining, analisi dei log, machine learning, simulazioni scientifiche e il data warehousing. Amazon EMR svolge le proprie attività non solo utilizzando di base Hadoop, ma anche interagendo con altri servizio offerti da AWS (Amazon Web Service), il pacchetto di servizi web offerti dal Cloud Amazon. La Figura2.17 mostra come Amazon EMR interagisce con gli altri servizi di AWS:

- Amazon EC2: servizio di computazione in cloud tramite il quale è possibile ottenere istanze virtali per il calcolo computazionale. Amazon EMR distribuisce le attività di calcolo su un cluster di istanze

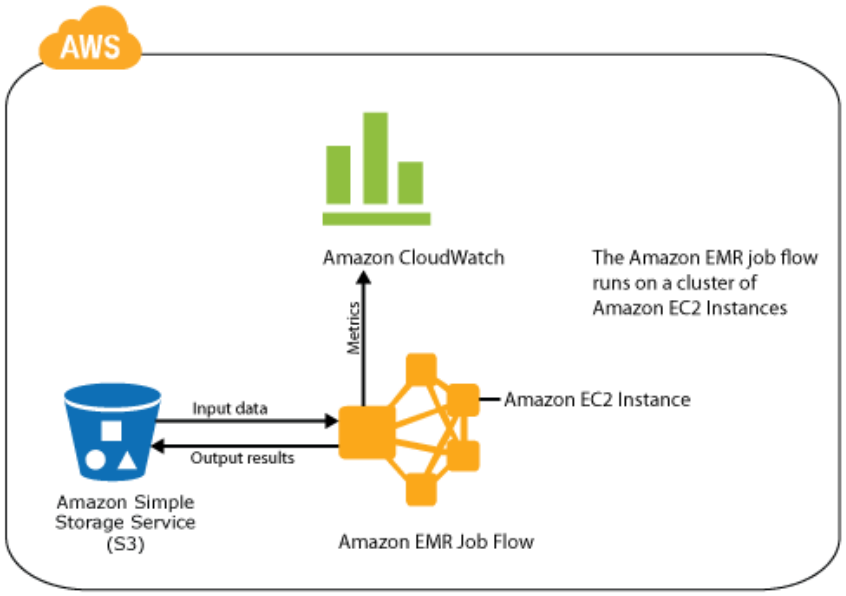


Figura 2.17: Amazon EMR Cluster [27]

virtuali EC2 in esecuzione sul cloud. Per una massima integrazione con Hadoop le istanze sono caratterizzate da macchine virtuali Linux.

- Amazon S3: servizio di storage in cloud integrabile e utilizzabile in combinazione con molti altri servizi AWS. Amazon EMR utilizza S3 come servizio di memorizzazione dell'input e dell'output delle computazioni di Hadoop. Amazon S3 è solo uno dei possibili servizi di memorizzazione utilizzabili con EMR, ad esempio, un possibile sostituto potrebbe essere il database NoDql DynamoDB, interrogabile mediante lo strumento Hive, compatibile con l'architettura di Amazon EMR.

- Amazon CloudWatch: servizio tramite il quale è possibile monitorare e gestire il cluster e le performance di computazione.

Tutti gli strumenti facente parte dell'ecosistema di Hadoop possono essere eseguiti su Amazon EMR. Le componenti più utilizzate, come Hive, Pig, HBase sono già integrate all'interno di Amazon EMR.

Avere a disposizione un ambiente già configurato e preinstallato è solo uno dei vantaggi nell'utilizzare una soluzione cloud. L'esecuzione di Hadoop su Amazon EMR permette di ottenere tutti i vantaggi che caratterizzano il mondo cloud:

- Provisioning: capacità del cluster di richiedere e ricevere in breve tempo macchine virtuali;
- Scalability: capacità di aumentare il numero di nodi virtuali del cluster per aumentare la potenza di calcolo, pagando solamente ciò che si utilizza;
- Integration: la capacità di integrarsi e interagire con altri servizi offerti da AWS.

Questi aspetti sono molto importanti quando si valuta il tipo di soluzione che si vuole realizzare.

2.6.4 MapR

L'ultima distribuzione che si provvederà ad osservare è MapR sviluppata e rilasciata da MapR Technologies. La distribuzione in questione integra le funzionalità di Apache Hadoop per l'archiviazione affidabile e il processamente di grandi quantità di dati. Come è possibile notare dalla Figura2.18, MapR di differenzia dalle distribuzioni viste fin'ora, perché invece di utilizzare HDFS come file system distribuito, sfrutta un'implementazione proprietaria chiamata MapR-FS. MapR-FS è stato progettato per riuscire ad ottenere da Hadoop prestazioni significativamente migliori in termini di affidabilità, efficienza, manutenibilità rispetto al classico HDFS. MapR-FS diversamente da HDFS supporta il protocollo NFS (Network File System) in modo che ogni possibile applicazione possa leggere o scrivere sul file system, indipendententemente che si tratti di un file system locale o di MapR-FS specifico per MapR. Il protocollo NFS necessita di un file system in grado di gestire le scritture casuali. MapR-FS, diversamente da da HDFS, è stato progettato appositamente per gestire le scritture casuali. Nelle ultime versione, a fianco di MapR-FS è stato introdotto il database NoSql MapR-DB, svluppanto anch'esso da MapR Technologies, e in grado di eseguire applicazioni esistenti di HBase, perchè ne incorpora le API.

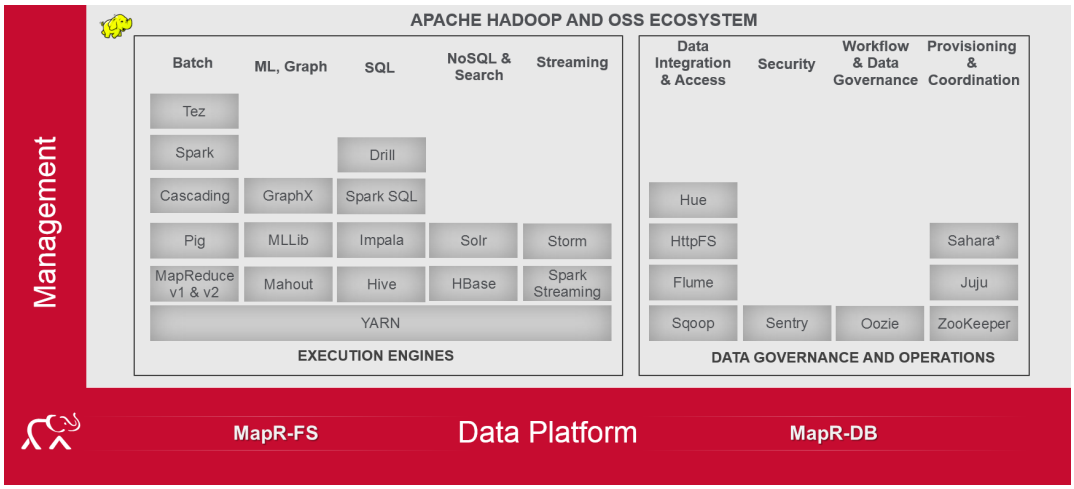


Figura 2.18: MapR Distribution [28]

Diversamente da Cloudera e Hortonworks, MapR è stata scelta da Amazon per affiancare la propria versione di Hadoop sul servizio cloud di Elastic Map Reduce (EMR), e da Google per l'integrazione con il servizio cloud Google Compute Engine.

Capitolo 3

Case Study

Il progetto che si è sviluppato, nel presente elaborato, è stato svolto all'interno del contesto di un'azienda leader nelle soluzioni di controllo di condizionamento, refrigerazione e riscaldamento. Fra i tanti prodotti, si occupa dello sviluppo e del commercio di dispositivi di controllo, installabili all'interno di Bottle Cooler. I **Bottle Cooler** sono tutti quei dispositivi di raffreddamento di bevande, che comunemente si trovano all'interno di bar, autogrill, supermercati e negozi di alimentari in generale (vedi Figura3.1). Nel contesto del progetto, i bottle cooler, per lo più, vengono prodotti e distribuiti dalle stesse aziende di bevande (dette *beverage company*) che li riforniscono. l'Azienda, in questo contesto, provvede a sviluppare i dispositivi di controllo termo-dinamici, acquistati dalle beverage company e installati all'interno dei Bottle Cooler. Questi dispositivi di controllo termo-dinamici, sono apparecchi caratterizzati da sensori, di varia natura, in grado di raccogliere misurazioni e produrre dati. L'obiettivo che l'azienda si è prefissato è quello di raccoglie le misurazioni generate dalla sensoristica dei dispositivi di controllo, al fine di monitorare i bottle cooler e l'utilizzo che ne viene fatto.

Nei paragrafi successivi, verrà definito meglio il concetto di monitoraggio dei bottle cooler, lo studio di fattibilità e la soluzione implementata per soddisfare tale requisito.

Figura 3.1: Esempio di Bottle Cooler

3.1 Monitoraggio dei Bottle Cooler

L'obiettivo del progetto consiste nello sviluppo di una piattaforma di Business Intelligence, che consenta alle bevarage company dei bottle cooler, che utilizzano i dispositivi di controllo termo-dinamico, di monitorarne il corretto funzionamento e l'utilizzo da parte degli esercizi commerciali che adoperano i bottle cooler delle aziende di bevande.

Le misure percepite dai dispositivi di controllo dei bottle cooler, sono principalmente di temperatura, pressione, umidità, ecc. I dispositivi presentano anche tipologie di sensori in grado di rilevare l'apertura e la chiusura delle porte dei bottle cooler. Queste misure, di per se, possono essere utilizzate per verificare l'andamento di funzionamento del dispositivo, ma allo stesso tempo non permettono di identificare come il bottle cooler viene utilizzato dagli esercizi commerciali. A tal fine, sono stati sviluppati (al di fuori del contesto dell'elaborato di tesi) e testati un insieme di algoritmi termodinamici, in grado di dedurre lo stato di utilizzo dei bottle cooler, a partire dalle misurazioni ottenute dai sensori. Lo studio degli andamenti termo-dinamici dei bottle cooler, affincato all'applicazione di un certo numero di soglie fisiche, ha così permesso di sviluppare algoritmi in grado di stabilire, per esempio, quante potenziali vendite sono state svolte nell'arco di una determinata ora, oppure il numero di *refill* giornalieri a cui è stato sottoposto un determinato bottle cooler.

In generale, si tratta di algoritmi in fase di sperimentazione e, come tali, devono essere sottoposto a un processo di validazione, che permetterà di verificare l'attendibilità dei risultati ottenuti dall'elaborazione delle misurazioni termo-dinamiche. Alla luce di questi elementi, il progetto ruota attorno all'obiettivo di sviluppare una piattaforma, che integri gli algoritmi già sviluppati, al fine di elaborazione le misurazioni provenienti dai singoli dispositivi di raffreddamento.

Il progetto, nel suo complesso, è stato suddiviso in due fasi:

- una prima fase di sviluppo di una tradizionale piattaforma di Business Intelligence, in grado di elaborare, tramite gli algoritmi, le misurazione raccolte e organizzare i risultati ottenuti, per le analisi sul monitoraggio dei bottle cooler;

- una seconda fase riguarda la progettazione, lo sviluppo e la valutazione di una piattaforma Big Data per l'elaborazione delle misurazioni raccolte dai sensori.

La prima fase è stata realizzata appositamente per lo svolgimento dello studio di fattibilità del progetto. Questa infatti ha permesso di testare e verificare l'attendibilità dei risultati prodotti da algoritmi, sviluppati su una base teorica, su un contesto reale, con misurazioni e verifiche reali.

La seconda fase del progetto, data dalla necessità di sviluppare una seconda architettura, in grado di elaborare le misurazione raccolte, deriva principalmente dall'esigenza di dover gestire un numero crescente di bottle cooler. La crescita del numero di bottle cooler provocherebbe, di conseguenza, una crescita esponenziale delle misurazioni raccolte. Per questo motivo emerge la necessità di progettare e valutare, una possibile soluzione alternativa ai tradizionali sistemi di BI, in grado di elaborare la mole di dati prodotta dai dispositivi, in tempi ragionevoli e in sintonia con le tempistiche di analisi.

Nei paragrafi successivi verranno descritte le soluzione e le architetture adottate in ognuna delle due fase, soffermandosi sui dettagli tecnici e quantitativi, non forniti fino ad ora. Inoltre verranno valutati pregi e difetti di entrambe le soluzione, alla luce delle perfomance e delle tempistiche di elaborazione.

3.2 Analisi del Dominio

Prima di procedere alla descrizione delle soluzioni implementate, è opportuno analizzare il dominio di contesto. Lo scopo dell'analisi del dominio è quello di comprendere a fondo i concetti, le dinamiche, le regole generali che definiscono il dominio applicativo, all'interno del quale la soluzione dovrà essere integrata.

I bottle cooler, su cui attualmente sono installati i dispositivi di controllo termodinamico, sono all'incirca 30. Ogni dispositivo è collocato all'interno di un esercizio commerciale (come ad esempio bar, autogrill, ecc), detto anche Point Of Sale (POS). Ogni POS prevede una o più linee di collegamento, alle quali vengono collegati uno o più bootle cooler.

Ognuno dei dispositivi installati campiona, ogni 30 secondi, i valori di all'incirca 20 variabili termodinamiche. Non tutte le misurazioni vengono campionate ogni 30 secondi, ve ne sono alcune, come per esempio quelle relative all'apertura e alla chiusura delle porte dei bottle cooler, che vengono rilevate solo nel momento in cui l'evento si verifica. I flussi dati che andranno ad alimentare gli algoritmi dovranno tenere in considerazione tale aspetto, in modo da riorganizzare i dati per l'elaborazione. I dati relativi alle variabili, una volta raccolti, vengono inviati periodicamente a una sorgente detta Provider, che fungerà da sorgente informativa della piattaforma.

I vari algoritmi termodinamici sono stati già implementati in tecnologia Java, e a partire delle misurazioni raccolte dai sensori, permettono di ricavare:

- gli eventi di *Refill*
- il livello di Stock (*StockLevel*)
- gli eventi di Potenziale Vendita (*PotentialSale*)

Come questi algoritmi vengono implementati, non è determinante per la progettazione dell'architettura. Invece, risulta importante avere coscienza di quali sono le interfacce di input e output per ognuno degli algoritmi sopra citati. Tutti gli algoritmi forniti, ad ogni esecuzione, elaborano una serie di valori, relative ad un certo intervallo temporale, di un solo dispositivo per volta. Questo è un requisito importante al fine della progettazione della soluzione. Infatti, a seconda dell'algoritmo che deve essere lanciato

in esecuzione, la piattaforma avrà il compito di selezionare l'insieme delle variabili ed eventi necessari per quella particolare computazione.

3.3 Fase 1: Business Intelligence Tradizionale

I sistemi di Business Intelligence, rappresentano l'insieme dei sistemi informativi aziendali e delle tecnologie informatiche finalizzate a supportare e automatizzare processi di misurazione, controllo e analisi dei risultati e delle performance aziendali. Tali sistemi, nati a partire dagli anni Novanta, sono oramai una realtà nella maggior parte delle organizzazioni aziendali. Questi sistemi rappresentano un valido supporto per il management nel compiere scelte operative e strategiche adeguate e tempestive.

In generale la piattaforma sviluppata, si è cercato di progettarla seguendo le best practice di progettazione dei sistemi di Business Intelligence; tuttavia, l'integrazione degli algoritmi di elaborazione, ha introdotto, all'interno della soluzione, varianti che scostano l'architettura delle tradizionali tecniche di BI. Nei sistemi "classici" di Business Intelligence, i dati vengono ordinati, riorganizzati ed eventualmente aggregati, mai generati. In questo contesto, vi è sia una riorganizzazione dei dati, dovuta alla preparazione dei dati di input per l'elaborazione, ma vi è anche una generazione dovuta all'introduzione degli algoritmi termo-dinamici all'interno del flussi di Business Intelligence.

La piattaforma di Business Intelligence, per ogni bottle coolear, a partire dalla raccolta periodica dei valori delle grandezze fisiche e degli eventi registrati dai sensori installati, dovrà determinare, tramite gli algoritmi termodinamici sviluppati ad-hoc, una serie di fatti rilevanti che consentano di monitorare il corretto utilizzo dei bottle cooler. I fatti prodotti dalla computazione degli algoritmi, dovranno poi essere gestiti e riorganizzati con le tradizionali tecniche di Business Intelligence, al fino di creare uno strato di dati ad-hoc, dal quale poter estrarre informazioni utili al business.

Nel paragrafo successivo, è possibile osservare l'architettura di BI implementata appositamente per il dominio di contesto appena descritto.

3.3.1 Architettura Funzionale

Lo schema, presente in Figura3.2, mostra l'architettura, ad alto livello, sviluppata per la piattaforma di Business Intelligence dedicata al monitoraggio dei bottle cooler. Come è possibile notare, l'architettura presenta

Figura 3.2: Architettura Funzionale

diversi livelli:

- **Sorgente Provider**: rappresenta la sorgente informativa che si interfaccia con i dispositivi di controllo presenti sui bottle cooler. Questa sorgente periodicamente riceve le misurazioni raccolte sui diversi bottle cooler, posizionati all'interno dei diversi punti vendita. Le informazioni principali, che questa sorgente raccoglie, sono di 2 tipologie:
  - *Valori delle variabili* fisiche e digitali, come per esempio la temperatura e la pressione.
  - *Eventi* di Apertura/Chiusura porta.

  Queste vengono memorizzate su una base dati che sfrutta la tecnologia di Microsoft SQL Server. La sorgente, per motivi di permessi di varia natura, non sono accedibili direttamente tramite le tecniche SQL

tradizionali, ma le estrazione dei dati avvengono solamente mediante l'utilizzo di *store procedure* ad-hoc

```
exec SensorsValue @deviceId = {device_id}
                  @dateFrom = {date_from}
                  @dateTo = {date_to}
```

Ad ogni richiesta di estrazione, deve essere specificato il dispositivo di riferimento (*device_id*) e l'arco temporale (*date_from* e *date_to*) relativo ai valori delle variabili e degli eventi che si desidera ottenere. Questa limitazione obbliga la piattaforma a gestire l'importazione dei dati un dispositivo per volta.

- **Staging Area (SA) e Data Warehouse (DWH)**: rappresentano il cuore della piattaforma di Business Intelligence, ovvero l'area in cui i dati, importati dalla sorgente informativa, vengono riorganizzati, ripuliti e preparati per l'elaborazione e l'analisi. Nel contesto seguente i dati vengono importati all'interno della Staging Area, dove vengono ripuliti e organizzati per essere utilizzati come input per gli algoritmi. Questi infatti, al termine della fase di import dalla Sorgente Provider, elaborano le misurazioni di un dispositivo per volta, e riportano i risultati di output ottenuti dalla computazione, sul Data Warehouse. All'interno di questa base di dati consistente e integrata, vengono mantenuti i dati di input e di output degli algoritmi.

- **Algoritmo**: rappresenta l'algoritmo di interesse che viene lanciato su un determinato insieme di valori, compresi in un certo arco temporale, per un particolare dispositivo. A seconda del fatto che si vuole monitorare, viene lanciata l'esecuzione di un algoritmo piuttosto che un'altro. I principale fatti che si intende monitorare, per i quali sono stati sviluppati i relativi algoritmi, sono:

  - l'evento di **refill**, ovvero il momento in cui avviene il rifornimento del bottle cooler;
  - la **potenziale vendita**, cioè l'azione di vendita registrata su un bottle cooler;
  - il **livello di stock** rilevato sul bottle cooler in un dato istante temporale.

Gli algoritmi, tutti sviluppati in Java, sono indipendenti l'uno dall'altro, sia per la struttura, che per l'esecuzione. L'insieme dei dati passati come input, necessario per la computazione, varia da un algoritmo all'altro. In particolare, l'algoritmo di Refill sfrutta sia le variabili fisiche e dinamiche, che gli eventi di apertura e chiusura porta; invece, gli algoritmi di Potential Sale e Stock Level sfruttano, rispettivamente gli eventi di apertura/chiusura porta e le misure delle variabili fisiche e dinamiche. Tale organizzazione dell'input viene gestita, nella maniera più appropriata, all'interno della Staging Area, al fine di poter poi memorizzare tali dati all'interno del Data Warehouse. Dualmente, l'output prodotto dagli algoritmi, viene mantenuto anch'esso all'interno del Data Warehouse; l'insieme degli output prodotti, su tutti i dispositivi, andrà direttamente ad alimentare i fatti all'interno del Data Mart.

- **Data Mart**: rappresenta una porzione del Data Warehouse, che ruota attorno ad un particolare fatto di interesse per il business. A partire dai dati grezzi e dagli eventi calcolati dagli algoritmi termodinamici, viene popolato il rispettivo Data Mart, in cui i dati sono strutturati secondo un modello ottimizzato per l'analisi, chiamato Modello Multidimensionale; in questo modello le informazioni sono organizzate in fatti e dimensioni. I fatti rappresentano tipicamente gli eventi quantitativi di interesse, mentre le dimensioni sono le informazioni correlate ad un fatto, che ne permettono la contestualizzazione. Solitamente all'interno di un Data Mart vengono collocati i dati che descrivono un solo fatto di interesse, per cui, l'architettura verrà sviluppata in modo tale da implementare un Data Mart per i fatti di Refill, uno per quelli di Potential Sale e un'altro ancora per i fatti di Stock Level.

  Sia i Data Mart che il Data Warehouse sono stati implementati con tecnologia PostgreSQL.

- **Reportistica**: insieme di sistemi e tecniche in grado, a partire dai dati presenti sul Data Mart, di fornire una documentazione analitica sui fatti di rilievo oggetto dell'analisi da parte del business. L'aspetto della reportistica, nonostante presente con tecnologia Jaspersoft, non viene trattato perché non materia del presente elaborato di tesi.

Descritti i macro livelli che caratterizzano l'architettura della piattaforma di interesse, si può procedere ad analizzare e osservare i singoli flussi di caricamento che regolano i vari livelli della piattaforma.

3.3.2 Flussi di Caricamento

I macro componenti/livelli che caratterizzano l'architettura della piattaforma di Business Intelligence, implementati per il monitoraggio dei bottle cooler, vengono gestiti mediante un **processo di ETL**.

In generale, l'ETL non è altro che un processo di estrazione, trasformazione e caricamento dei dati all'interno di una base di dati integrata, normalmente un Data Warehouse o un Data Mart. I dati, come nel caso seguente, vengono estratti da sorgenti OLTP (ovvero transazionali) e poi sottoposti ad un processo di trasformazione, che consiste per esempio nel selezionare solo i dati di interesse per il sistema, eliminare eventuali duplicati, derivare nuovi dati calcolati, ecc. Tale trasformazione ha lo scopo di consolidare i dati, ovvero rendere omogenei dati provenienti da sorgenti diverse, e fare in modo che siano più aderenti possibili alla logica di business del sistema di analisi per cui viene sviluppato.

La gestione di tutti i flussi, che governano la piattaforma di Business Intelligence implementata, sono stati sviluppati con **Pentaho Kettle**. Pentaho Data Integration (PDI, chiamato anche Kettle) è uno dei tanti strumenti di ETL utilizzati per l'integrazione dei dati. A seconda del tipo di manipolazione che occorre applicare ai dati, Kettle applica componenti e strumenti differenti, come per esempio:

- la pulizia dei dati;
- la migrazione dei dati tra applicazioni o database distinti;
- l'esportazione di dati presenti su file a database, e viceversa;
- la gestione dei flussi big data;
- e molto altro.

Le *trasformations* e i *jobs* sono i concetti che stanno alla base di un flusso ETL in Kettle. Le trasformations gestiscono la manipolazione diretta dei dati o delle righe estratte dalle sorgenti; si compongono di una o più fasi,

che svolgono le attività di base del processo di ETL, come la lettura da file, la pulizia dei dati, il filtraggio, o il caricamento dei dati sulle basi di dati. Le attività, che vengono definite all'interno di un trasformazione, vengono eseguite in parallelo. Diversamente accade all'interno dei jobs, entità composte ad uno o più attività (trasformazioni o altri job), che vengono eseguite sequenzialmente, seguendo un determinato ordine, stabilito da precedenze definite dalle connessioni fra le diverse attività.

Figura 3.3: Job Principale di Caricamento

La Figura3.3 mostra il Job principale che gestisce il caricamento dell'intero flusso dei dati, provenienti dai sensori presenti sui dispositivi. Il flusso è costituito da tre sotto-Job principali:

1. Caricamento del Data Warehouse
2. Esecuzione degli Algoritmi Termodinamici
3. Caricamento del/dei Data Mart

Le tre attività appena citate, vengono eseguite sequenzialmente, in base all'ordine delle connessioni definite. La necessità di svolgere tali attività in maniera sequenziale deriva dal fatto che sono dipendenti l'una dall'altra. Infatti, da un lato, l'esecuzione degli algoritmi, non può essere eseguita prima che i dati di input, provenienti dai sensori, siano organizzati e caricati all'interno del Data Warehouse. Dall'altro lato, il caricamento dei Data Mart deve attendere che l'esecuzione degli algoritmi sia terminata, per potersi alimentare dai dati di output prodotti.

I flussi che gestiscono il caricamento dell'intera piattaforma sono stati implementati in *full-refresh*, ovvero non viene mantenuto uno storico dei dati già presenti sulla base di dati, e ad ogni esecuzione del caricamento, i dati vengono sovrascritti e sostituiti. Questo avviene per tutti i dati della

piattaforma, ad esclusione dei dati di input e output presenti sul Data Warehouse, questi infatti vengono gestiti in maniera incrementale, per consentire lo svolgimento dello studio di fattibilità sul processo.

All'avvio del Job principale viene specificato l'intervallo temporale di riferimento $\{date\_from, date\_to\}$. Questo intervallo viene utilizzato per specificare e selezionare le misurazioni che si sono verificate all'interno dell'arco temporale definito. In particolare, durante la fase di import, dal Server Provider vengono esportate tutte le misurazioni, relative ad un particolare insieme di dispositivi, che si sono verificate durante tale intervallo temporale.

Di seguito verranno descritte nel dettaglio le tre fasi principale del Job per il caricamento globale.

Data Warehouse

La fase di caricamento del Data Warehouse è la base, su cui si appoggia l'intera piattaforma di BI per il monitoraggio dei bottle cooler. Come viene mostrato in Figura3.4, il flusso adibito al caricamento del Data Warehouse è composto da due macro job:

- **Integration Layer**: flusso adibito all'importazione dei dati che risiedono sulla Sorgente Provider.
- **Staging Area**: flusso che riorganizza i dati importati e li prepara per essere utilizzati come imput per gli algoritmi di elaborazione.

La suddivisione del caricamento nei due flussi appena citati, permette, da un lato, di mantenere all'interno del Data Warehouse una copia speculare dei dati presenti sulla sorgente, dall'altro lato permette di "disaccoppiare" la fase di estrazione dei dati, da quella di trasformazione. 03_LOAD_DWH essendo il primo flusso ad essere eseguito, provvede al salvataggio dell'intervallo temporale ("Set dates") specificato durante l'avvio del Job principale, utilizzato anche dai flussi successivi.

Il flusso di **Integration Layer** si occupa principalmente di:

1. *Importare i dati provenienti dalla Sorgente Provider.*
   Per ogni bottle cooler disponibile, preleva dalla sorgente informativa, tutte le misurazioni delle variabili e degli eventi che si sono verificati nell'intervallo temporale specificato in input. I dati provenienti dalla sorgente Provider hanno la seguente struttura:

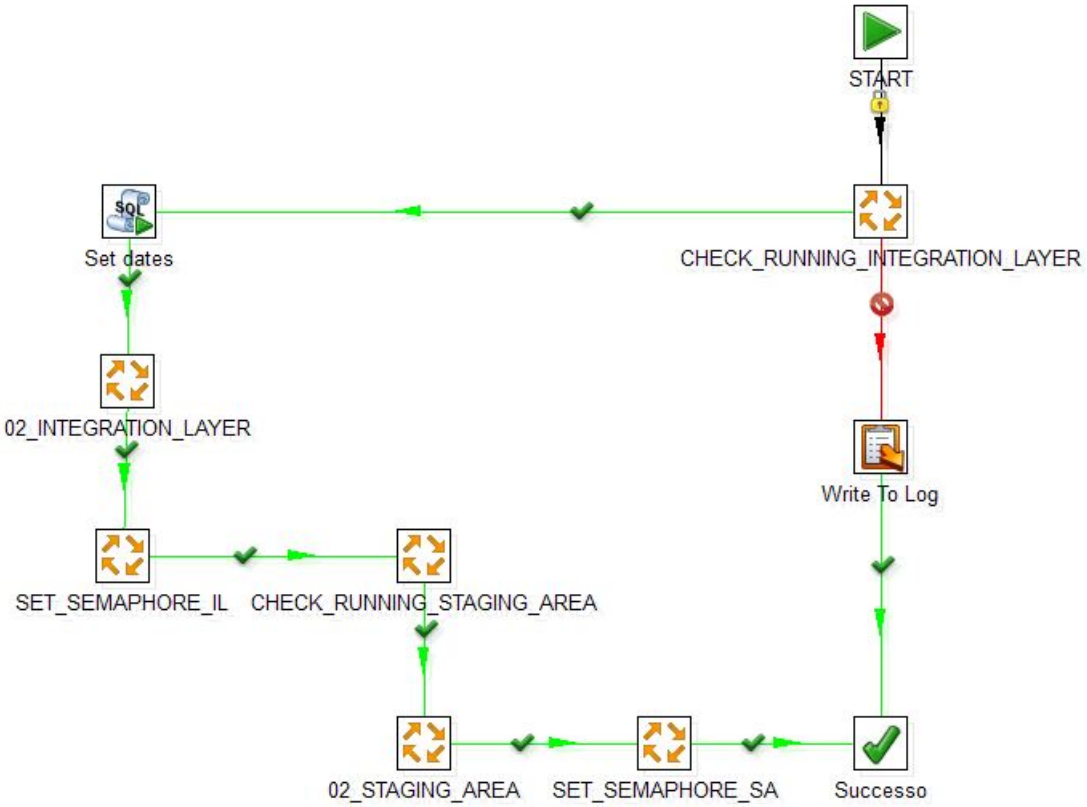


Figura 3.4: Caricamento del Data Warehouse

Device	Variable	VarValue	VarTime
182	134	21.4	2014-12-02 05:47:00.000
182	136	22.0	2014-12-02 21:09:30.000
183	140	0.00	2014-12-02 22:55:22.000
183	140	1.00	2014-12-02 22:57:49.000
182	136	21.0	2014-12-02 23:09:30.000
183	140	0.00	2014-12-02 23:55:22.000
...	...	...	...

Per ogni dispositivo (*Device*) installato all'interno dei bottle cooler, e per ogni variabile disponibile (*Variable*), viene registrato il valore associato (*VarValue*), verificatosi in un certo istante temporale (*VarTime*).

2. *Ri-campionare i valori relativi agli eventi di apertura e chiusura porta.* Gli algoritmi termodinamici, sviluppati per l'elaborazione dei dati provenienti dai sensori, necessitano di valori campionati ogni 30 secondi. Questo non crea problemi se si considerano solamente le variabili fisiche e dinamiche, ma non è così per le apertura e le chiusura porta, che vengono rilevate solamente quando l'evento si verifica. Queste, diversamente delle variabili fisiche e dinamiche, non sono campionate ogni 30 secondi, per cui l'Integration Layer ha il compito di ri-campionare tali valori, in modo tale da predisporre l'input esattamente secondo le specifiche degli algoritmi. Qui di seguito viene mostrato un esempio di ri-campionamento da parte dell'Integration Layer.
Si supponga che per un determinato dispositivo siano disponibili le seguenti rilevazioni relative all'apertura e alla chiusura della porta:

Device	Variable	VarValue	VarTime
183	140	0.00	2014-12-02 22:55:22.000
183	140	1.00	2014-12-02 22:57:49.000

Queste due rilevazioni, soggette al ri-campionamento, portano alla creazione dei seguenti campioni:

Device	Variable	VarValue	VarTime
183	140	0.00	2014-12-02 22:55:00.000
183	140	0.00	2014-12-02 22:55:30.000
183	140	0.00	2014-12-02 22:56:00.000
183	140	0.00	2014-12-02 22:56:30.000
183	140	0.00	2014-12-02 22:57:00.000
183	140	1.00	2014-12-02 23:57:30.000

Questa attività di ri-campionamento degli eventi viene svolta fra l'estrazione delle misurazioni di un dispositivo e l'altro. In questo modo, al termine della fase di import, non solo i dati sono presenti all'interno del Data Warehouse, ma una prima trasformazione è già stata applicata ai dati, che dovranno poi essere passati come input agli algoritmi. Non solo durante la computazione degli algoritmi si assiste alla generazione di dati, anche a questo livello dell'architettura i dati vengono prodotti per essere poi correttamente elaborati dagli algoritmi.

Il flusso di **Staging Area** viene attivato non appena il flusso di Integration Layer termina le proprie attività. Questo in generale predispone l'input che verrà fornito agli algoritmi termodinamici. L'input passato ai vari algoritmi può essere di due tipologie: variabili fisiche e dinamiche da un lato, e apertura/chiusura porta dall'altro. In fase di analisi del dominino, si sono stabilite le interfacce di comunicazioni fra gli algoritmi e i dati di input. Questi ultimi infatti, vengono posizionati in determinate tabelle del Data Warehouse, caratterizzate da una struttura ben definita:

- la tabella della base di dati che accoglie le Variabili fisiche e dinamiche ha la seguente struttura:

Device	VarTime	Var1Value	Var2Value	...	VarNValue
183	2014-12-02 22:55:00.000	10.0	22.1	...	20.0
183	2014-12-02 22:55:30.000	11.0	21.4	...	21.0
183	2014-12-02 22:56:00.000	9.5	21.2	...	21.5
182	2014-12-02 22:56:00.000	13.00	22.2	...	22.5
...	...	...	...	...	...

In generale, c'è stata una sorta di "Pivoting" dai dati importati dalla fase di Integration Layer. Ogni record rappresenta l'insieme dei valori delle N variabili (*Var1Value...VarNValue*), in un particolare istante temporale (*VarTime*), per un determinato device (*Device*).

- le apertura/chiusura porta presentano la seguente struttura:

Device	StartTime	EndTime	OpenSeconds
182	2014-12-02 20:56:00.000	2014-12-02 20:56:25.000	25.00
182	2014-12-02 20:58:10.000	2014-12-02 21:00:13.000	143.00
...	...	...	...

Ogni record rappresenta l'apertura e la rispettiva chiusura (*StartTime* e *EndTime*), per un certo numero di secondi (*OpenSeconds*), della porta di un determinato bottle cooler.

Avendo a che fare con all'incirca 30 bottle cooler, considerando che per ognuno di questi si campionano all'incirca 20 variabili, se si considera il caricamento che copre l'arco di una settimana, il numero di record totale da elaborare sarà all'incirca: $30*10*7*(2*60*24) = 12.096.000$. Le query SQL, che stanno alla base delle trasformazioni che si occupano di predisporre i dati di input per gli algoritmo, considerando la mole di dati che devono elaborare, sono estremante "costose" dal punto di vista computazionale e le perfomance diminuiscono con l'aumentare di dati.

Esecuzione degli Algoritmi

Il secondo "step" del flusso principale di caricamento è caratterizzato dal job che si occupa di lanciare l'esecuzione degli algoritmi. Come è possibile notare dalla Figura3.5, le esecuzioni dei tre algoritmi vengono lanciate in parallelo. Questo perché, da un lato, accedono in sola lettura allo stesso insieme di input, e dall'altro, l'output prodotto viene memorizzato in strutture distinte in base all'algoritmo che li ha prodotti. I tre algoritmi quindi, in generale sono indipendenti l'uno dall'altro, condividendo però i dati di input. A seconda dell'algoritmo che viene lanciato il flusso fornisce entrambe le tipologie di input, oppure una sola delle due. In particolare:

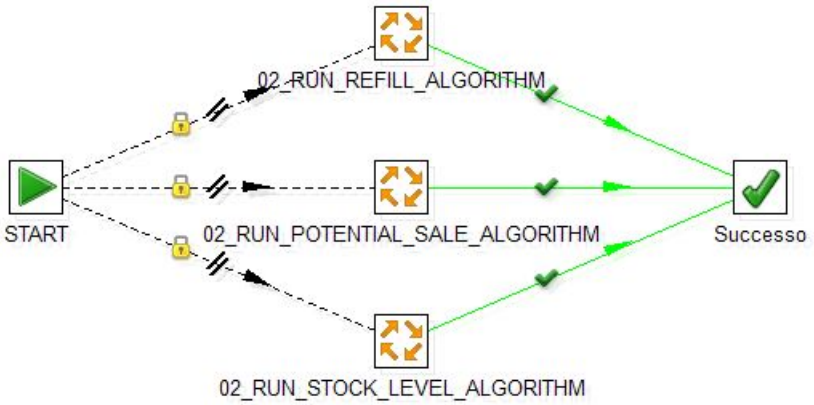


Figura 3.5: Esecuzione degli Algoritmi Termodinamici

- le variabili fisiche e dinamiche sono di interesse per gli algoritmi di Refill e di Stock Level.

- gli eventi di apertura/chiusura porta sono di interesse per gli algoritmi di Refill e di Potential Sale.

Un'altra caratteristica che accomuna tutti e tre gli algoritmi è la modalità di computazione che svolgono, ovvero all'interno della stesso ciclo di computazione non vengono elaborati tutti i dati di input, relativi all'intervallo temporale specificato, di tutti i dispositivi considerati. Ogni algoritmo elabora ciclicamente un dispositivo per volta, considerando tutti i dati di input appartenenti a quel particolare dispositivo. Questo è un vincolo progettuale dovuto all'architettura dei singoli algoritmi.

I dati di output prodotti dalla computazione dei singoli algoritmi vengono memorizzati separatamente sul Data Warehouse, secondo le seguenti strutture:

- $Refill(device, startTime, endTime, openSeconds, refillValue)$

- $PotentialSale(device, startTime, endTime, openSeconds, PotentialSaleValue)$

- $StockLevel(device, time, StockLevelValue)$

Caricamento dei Data Mart

Infine, l'ultimo job del flusso principale provvede al caricamento dei fatti e delle dimensioni che andranno poi a popolare i Data Mart della piattaforma. Le informazioni che contestualizzano il dominio dei bottle cooler, e che andranno a popolarne le dimensioni sono:

- Beverage company
- POS (Point Of Sales)
- Lines
- Device

Tali informazioni, allo stato dell'arte del progetto, vengono gestite manualmente mediante apposite tabelle $m\_$. Questo perché non vi è un sistema informativo vero e proprio dal quale queste informazioni possono essere prelevate. Sviluppi futuri prevedono meccanismi automatici di gestione di tali informazioni.

Attualmente i Data Mart implementati sono tre:

- PotentialSale
- Refill
- Temp

Il Data Mart relativo ai fatti di StockLevel non è stato implementato perché lo studio di fattibilità sui risultati, prodotti dal rispettivo algoritmo, non è stato soddisfacente a tal punto da implementarvi potrà il rispettivo Data Mart.

Nel paragrafo successivo verrà mostrato più nel dettaglio il modello concettuale e logico che sta alla base della progettazione dei Data Mart appena citati.

3.3.3 Modello Concettuale/Logico

Di seguito viene presentato lo schema concettuale delle dimensioni e dei fatti ipotizzati per la soluzione di Business Intelligence, che consentiranno

alle diverse beverage company di monitorare il corretto utilizzo dei bottle cooler da parte degli esercenti.

Nel dettaglio sono state definite:

- Le **dimensioni conformi**
  - **Device**: attributi e gerarchie che caratterizzano i bottle cooler;
  - **Tempo**: attributi e gerarchie relative alla dimensione temporale.
- I **fatti**
  - **Refill**: eventi di rifornimento - refill - che avvengono su ogni bottle cooler;
  - **Temp**: temperature rilevate su ogni bottle cooler;
  - **Potential Sale**: potenziali vendite - potential sale - che avvengono su ogni bottle cooler.

Nello specifico, utilizzando il formalismo del DFM (Dimensional Fact Model), per ogni dimensione e fatto verranno descritte le informazioni di riferimento e le misure che le caratterizzano.

Dimensione Device

La dimensione *device* (vedi Figura3.6) consente di raggruppare e analizzare i device aggregandoli secondo gli attributi analitici che li caratterizzano. Ogni device istallato su uno specifico bottle cooler è identificabile mediante un codice univoco. Come è possibile notare, per la dimensione device risultano essere presenti diversi attributi dimensionali:

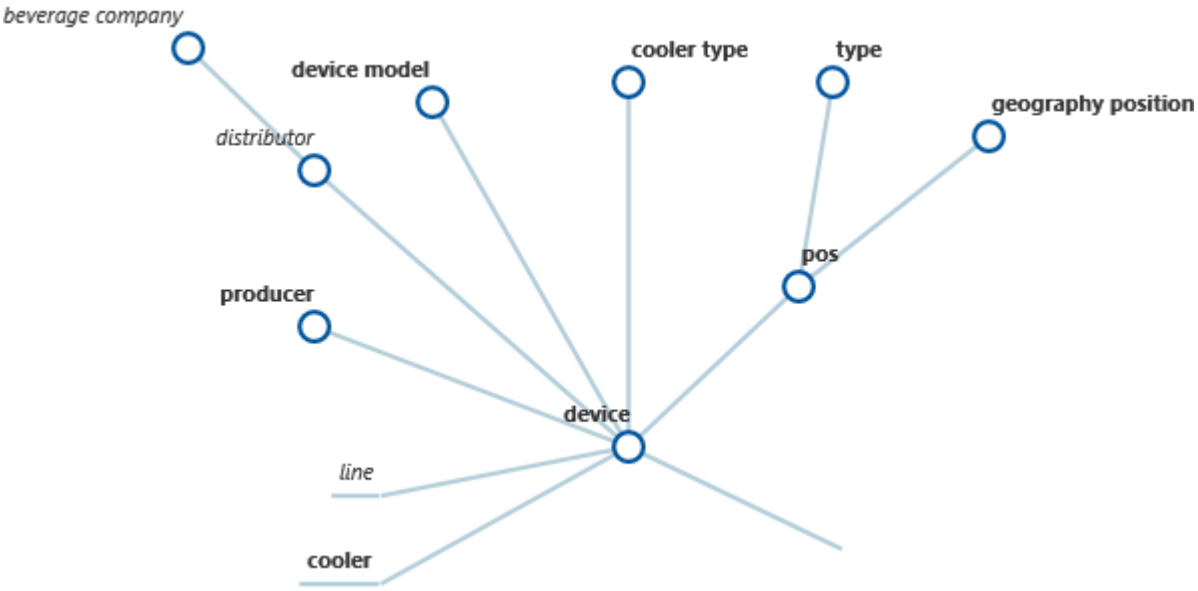


Figura 3.6: Dimensione Device

Nome	Descrizione
device	Singolo dispositivo di controllo installato su uno specifico bottle cooler
device model	Modello del device utilizzato come driver di configurazione dei parametri degli algoritmi termodinamici
cooler type	Clusterizzazione dei bottle cooler sulla base della tipologia dei prodotti contenuti
producer	Produttore del bottle cooler
distributor	Compagnia distributrice della beverage company
beverage company	Compagnia produttrice delle bevande
pos (Point Of Sale)	Luogo fisico in cui è installato il cooler
type	tipologia di POS
geography position	Posizione geografica del POS
line	Linea di trasmissione dati
cooler	Bottle cooler su cui è installato il device

Dimensione Time

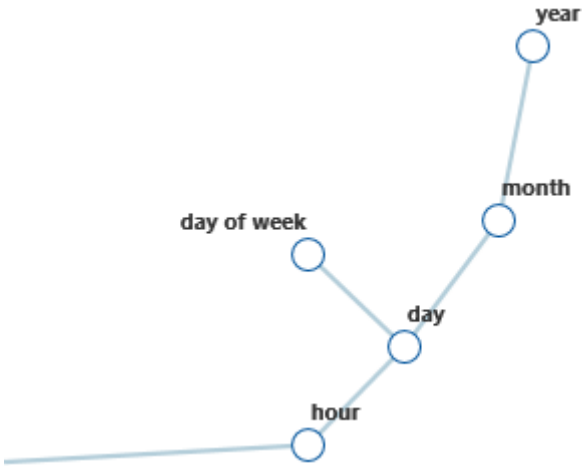


Figura 3.7: Dimensione Time

La dimensione conforme "Tempo" descrive tutti gli attributi e le gerarchie relative alla dimensione temporale. Essa risulta essere fondamentale per tutte le analisi, dal momento che nella maggior parte dei casi analizzati (e in generale in tutti i sistemi di BI), la reportistica ufficiale ha sempre un riferimento temporale volto a definire il periodo di riferimento per i dati esposti.

I dati importati dalla sorgente Provider presentano una granularità temporale dell'ordine dei 30 secondi. Per quanto riguarda l'analisi, questi dati vengono aggregati, in quanto un livello di dettaglio così basso non porterebbe alcun beneficio analitico. Si può notare, in Figura3.7, come sia possibile aggregare il singolo giorno in due modalità diverse: una risalita consente di aggregare fino al livello mensile/annuale, mentre una risalita permette di aggregare i giorni a livello settimanale. Di seguito andremo a dettagliare i vari attributi:

Nome	Descrizione
hour	Valore intero che identifica l'ora
day	Valore intero che identifica il giorno
month	Descrizione del Mese
year	Valore intero che identifica l'anno
day of week	Descrizione de identifica il giorno della settimana

Fatto Refill

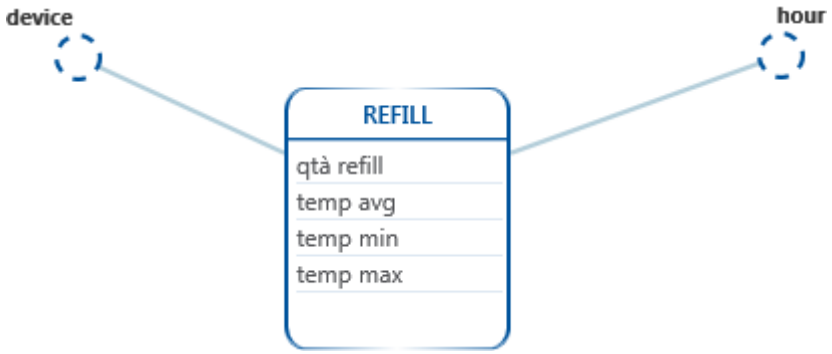


Figura 3.8: Fatto Refill

Il fatto Refill descrive l'occorrenza degli eventi di rifornimento (refill) che avvengono su ogni bottle cooler. Tali eventi sono determinati da un algoritmo termodinamico sulla base dei dati rilevati dai sensori, ovvero sulla base delle variabili fisiche/dinamiche e delle apertura/chiusura porta. Il fatto Refill è collegato alla dimensione conforme device e alla dimensione tempo a livello di ora.

Nome	Descrizione	Aggregazione
qtà refill	Numero di refill orari per bottle cooler	Somma
temp avg	Temperatura media orario	Media
temp min	Temperatura minima oraria	Minimo
temp max	Temperatura massima oraria	Massimo

Fatto Potential Sale

Il fatto Potential Sale rappresenta le potenziali vendite associate ad ogni bottle cooler. Il fatto Potential Sale è collegato alla dimensione conforme device e alla dimensione tempo a livello di ora. La dimensione descrittiva *temp at sale* rappresenta la temperatura del bottle cooler al momento della singola vendita.

Nome	Descrizione	Aggregazione
qtà sale	Potenziali numero di vendite all'ora per bottle cooler	Somma
qtà sale lower threshold	Potenaziale numero di vendite avvenute nel momento in cui il bottle cooler era al di sotto di una determinata soglia	Somma
qtà sale upper threshold	Potenziale numero di vendite avvenute nel momento in cui il bottle cooler era al si sopra di una determinata vendita	Somma
seconds from last refill	Numero di secondi trascorsi dall'ultimo refiil precedente alla potenziale vendita	-
temp at sale	Temperatura del bottle cooler nell'istante della potenziale vendita	-

Fatto Temp

Il fatto Temp rappresenta la temperatura che caratterizza un determinato bottle cooler ad un particolare ora del giorno. Il fatto Temp è collegato alla dimensione conforme device e alla dimensione tempo a livello di ora.

Qui di seguito è possibile identificare le misure che caratterizzano il fatto Temp.

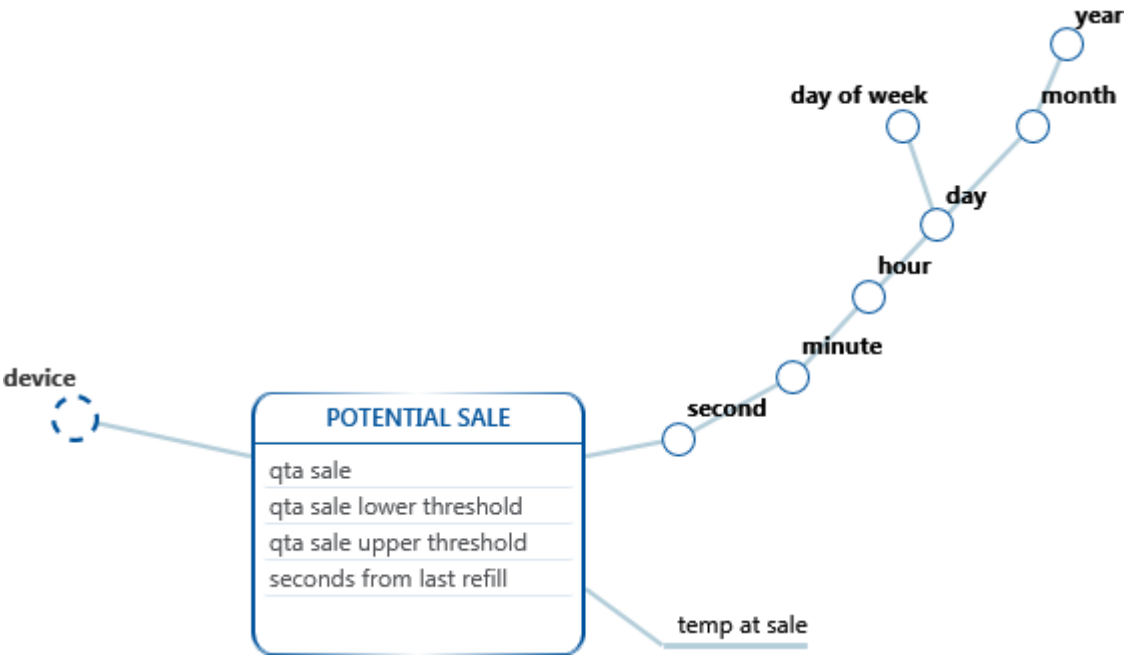


Figura 3.9: Fatto Potential Sale

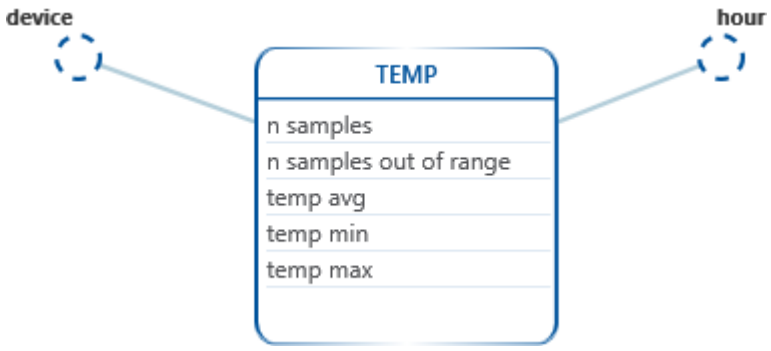


Figura 3.10: Fatto Temp

Nome	Descrizione	Aggregazione
n samples	Numero di campioni di temperatura rilevati all'ora	Count
n samples out of range	Numero di campioni di temperatura al di fuori dei range definiti dalla beverage company	Count
temp avg	Temperatura media oraria del bottle cooler	Media
temp min	Temperatura minima oraria del bottle cooler	Minimo
temp max	Temperatura massima oraria del bottle cooler	Massimo

Modello Logico

La struttura multidimensionale dei dati all'interno dei singoli Data Mart, può essere rappresentata utilizzando due distinti modelli logici: quello multidimensionale (MOLAP) e quello relazionale (ROLAP). I sistemi MOLAP memorizzano i dati utilizzando strutture intrinsecamente multidimensionali, invece i sistemi ROLAP utilizzano il modello relazione. Nel seguente contesto si è scelto di utilizzate un sistema ROLAP basato sul cosiddetto Schema a Stella (Star Schema). Uno schema a stella in generale è composto da un insieme di relazioni DT che rappresentano le Dimension Table, ovvero le dimensioni che permettono di contestualizzare i fatti, e una relazione FT chiamata Fact Table, per ogni fatto di interesse che si vuole analizzare.

Nel contesto del monitoraggio dei Bottle Cooler, essendo ancora un progetto in fase di valutazione, non presenta tutte le informazioni che sono state descritte nel modello DFM qui sopra descritto, e non rispetta fedelmente tutte le regole definite secondo lo schema a stella. I modelli DFM presentati mostrano ciò che ci si aspetta, in un ipotetico futuro, quando il progetto verrà posto in produzione.

Attualmente il modello logico che è stato sviluppato per i Data Mart, è il seguente:

- $DT\_BEVERAGE\_COMPANY(\underline{ids\_beverage\_company}, id\_beverage\_company, beverage\_company\_name, company\_code,$

upper_temp_threshold, *lower_temp_threshold*, *provider*)

- *DT_DEVICE*(*ids_device*, *id_deviceinteger*, *device_name*, *provider*, *ids_line:DT_LINE*, *ids_beverage_company:DT_BEVERAGE_COMPANY*)
- *DT_LINE*(*ids_line*, *id_line*, *line_name*, *timezone*, *provider*, *ids_pos:DT_POS*)
- *DT_POS*(*ids_pos*, *id_pos*, *pos_name*, *id_pos_type*, *pos_type*, *id_geography_position*, *geography_position_latitude*, *geography_position_longitude*, *provider*)
- *FT_POTENTIAL_SALE*(*ids_device*, *time_stamp*, *temp_at_sale_avg*, *qta_sale*, *qta_sale_lower_threshold*, *qta_sale_upper_threshold*, *seconds_from_last_refill*)
- *FT_REFILL*(*ids_device*, *hour_refill*, *qta_refillinteger*, *temp_avg*, *temp_mindoubleprecision*, *temp_max*
- *FT_TEMP*(*ids_device*, *hour_temp*, *n_samples*, *n_samples_out_of_range*, *temp_avg*, *temp_min*, *temp_max*)

3.4 Fase 2: Big Data

La prima fase del progetto per il monitoraggio dei bottle cooler, si è rivelata particolarmente utile per lo svolgimento dello studio di fattibilità, volto a stabilire l'affidabilità del progetto. Questo è stato svolto su un campione di all'incirca 30 bottle cooler, sui quali fisicamente si è riscontrata l'attendibilità dei risultati prodotti dagli algoritmi.

Questo studio è stato fatto con la speranza che, alla luce di risultati soddisfacenti, il progetto potesse essere applicato ad un numero molto maggiore di bottle cooler. All'avvio del progetto i numeri stimati, si aggiravano dai 50.000 ai 100.000 bottle cooler. Se si considera una situazione a pieno regime, considerando di voler mantenere due anni di dati in linea e tutti gli aspetti legati alla raccolta e al campionamento delle misurazioni descritte nella prima fase del progetto, si può notare una sostanziale crescita della quantità di dati da memorizzare ed elaborare, che non può essere trascurata e gestita con le tradizionali tecniche di BI.

Se si considera che, per ogni bottle cooler:

- Vengono raccolte le misurazioni di all'incirca 20 sensori;
- Ogni giorno vengono svolte 2.880 $(2 * 60 * 24)$ rilevazioni per sensore;
- Ogni record contenente un campione rilevato occupa al più 20 byte;

la stima giornaliera dei dati raccolti su di un bottle cooler è di circa 1.2 MB. Se però questo numero viene moltiplicato per il numero totale di bottle cooler stimati, risulterà essere all'incirca 120 GB al giorno.

Alla luce di tutto ciò che è stato discusso nell'elaborato, si può supporre di essere di fronte a una fonte Big Data, e quindi non più gestibile, in tempo ragionevoli, con i tradizionali sistemi di BI.

La necessità di poter fornire giornalmente informazioni, il più aggiornate possibile, alle rispettive beverage company, è un requisito che non può essere assolutamente trascurato, al fine della buona riuscita del servizio.

Nei successivi paragrafi verrà presentata la soluzione Hadoop progettata e sviluppata appositamente per il monitoraggio dei bottle cooler.

3.4.1 Architettura Funzionale

La necessità di gestire una quantità di dati molto maggiore, evitando che i tempo di elaborazione crescano esponenzialmente, porta allo progettazione di una soluzione in grado di massimizzare e parallelizzare l'elaborazione dei dati, ma soprattutto, come nel seguente contesto, in grado di eliminare i punti, all'interno del flusso di elaborazione dei dati che, al crescere dei dati, potrebbero risultare critici.

La soluzione è stata progettata e implementata sulla base della piattaforma Hadoop. Questa, come si è potuto notare, è:

- una delle tecnologie più utilizzate in ambito Big Data, quasi da diventare uno standard;
- utilizzata come core sulle piattaforme dei principali vendor, come ad esempio Cloudera, Amazon, Hortonwork, ecc;
- utilizzata sia in contesti cloud che contesti in-house;
- ecc;

Questi, e molti altri, sono fra i principali motivi per i quale si è scelto di progettare e sviluppare una soluzione sfruttando la tecnologia di Hadoop, analizzandone le caratteristiche e le logiche.

La soluzione implementata durante la prima fase del progetto prevedeva lo svolgimento di 4 macro attività:

1. Import dalla Sorgente Provider dei dati raccolti dai sensori;
2. Campionamento e Organizzazione dei dati per l'input degli algoritmi;
3. L'esecuzione degli algoritmi;
4. Caricamento dell'output, ottenuto dagli algoritmi, sul Data Mart.

All'aumentare della mole di dati, vi sono alcune di queste attività che ne risentono, in termini di performance, andando così ad impattare negativamente su tutta la soluzione prodotta. Le attività in questione riguardano il campionamento e l'organizzazione dei dati da un lato, e l'esecuzione degli algoritmi dall'altro. Il vincolo posto dagli algoritmi di elaborare un dispositivo per volta non può essere violato, per via delle logiche fisico-matematiche applicate per il recupero dei fatti, verificatosi sui bottle cooler. Inoltre, man mano che i dati crescono, le operazioni di join, applicate per la manipolazione e l'organizzazione dell'input per gli algoritmi, diventano sempre meno performanti.

L'architettura, che verrà mostrata nei paragrafi successivi, è stata progettata in modo tale da "rimappare" le attività critiche dell'architettura precedente, sulla logica MapReduce di Hadoop, in modo tale da sfruttare il parallelismo e la capacità di calcolo offerte dal cluster Hadoop.

Nella Figura3.11, è possibile osservare l'architettura di massima progettata per lo sviluppo della soluzione Hadoop. L'intera piattaforma ruota attorno al framework di calcolo di MapReduce, in particolare:

- La logica di Map ha il compito di sostituire l'attività di predisposizione dei dati di input necessari agli algoritmi. Durante tale fase, la logica della Map ha il compito di "riorganizzare" i dati in modo tale da raggruppare, sotto la stessa chiave, le misurazioni che dovranno essere elaborate all'interno della stesso ciclo di computazione dell'algoritmo.

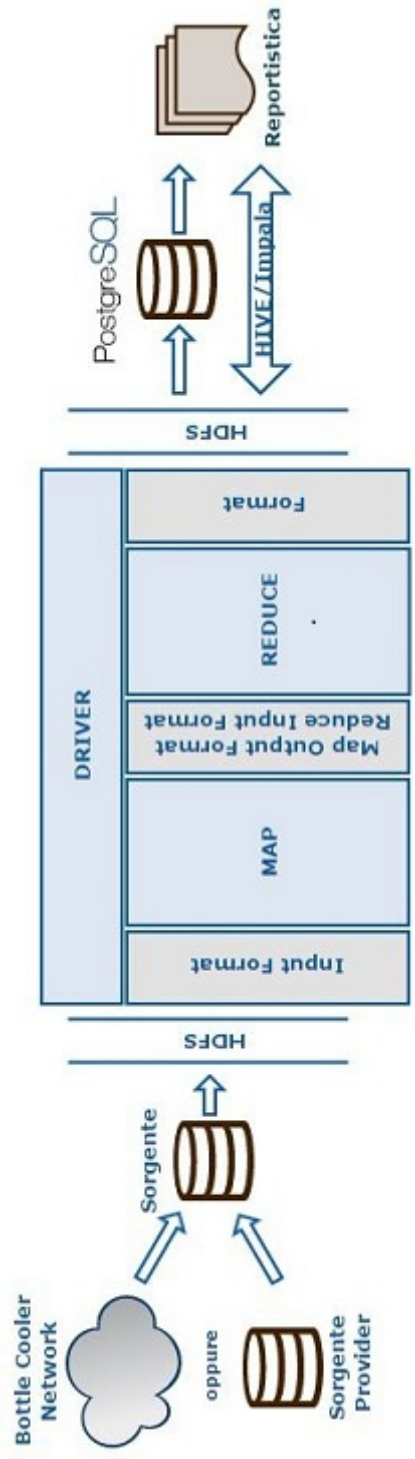


Figura 3.11: Architettura Funzionale della Soluzione Hadoop

- All'interno della logica di reduce, invece, vengono integrati gli algoritmi termodinamici, che a partire dai dati forniti dalla fase di map, ricavano i fatti verificatosi, con determinate probabilità, sui diversi bottle cooler.

In generale, è stata progettata e sviluppata una singola logica MapReduce in grado di generalizzare l'esecuzione di uno qualsiasi degli algoritmi sviluppati. A seconda dell'algoritmo che si vuole lanciare in esecuzione:

- la fase di map passa, alla fase di reduce, solamente le misurazione di interesse per l'algoritmo scelto per la computazione;

- dualmente, la fase di reduce, applica ed esegue uno solo dei tre algoritmi sviluppati.

Le logiche di Map e di Reduce, vanno così a sostituire le due macro attività, svolte all'interno del flusso tradizionale di ETL, che all'aumentare dei dati potrebbero risultare critiche.

La soluzione MapReduce che sarà sviluppata non va ad influire sulle attività di "Import dalla sorgente Provider" e "Caricamento dell'output sul Data Mart". Come e con quali strumenti i dati vengono posizionati o prelevati da HDFS, non è stato studiato e considerato nel seguente elaborato. Nonostante ciò, alcune ipotesi relativamente all'input e all'output sono state svolte. In particolare, si potrebbe pensare di utilizzare come Sorgente, da cui HDFS si alimenta, un database basato su tecnologie in grado di gestire e memorizzare i Big Data. Una possibile scelta potrebbe essere quella di Amazon S3, file system distribuito sul cloud di Amazon, in grado di scalare all'aumentare dei dati. Questo potrebbe consentire di adottare un soluzione in grado di mantenere la totalità dei dati grezzi raccolti sui diversi bottle cooler. Alla luce di ciò, si potrebbe pensare, da un lato, di mantenere la Sorgente Provider come fonte alimentante di tutta l'architettura, oppure, si potrebbe pensare di "dirottare" i dati grezzi raccolti sui diversi bottle cooler, direttamente sulla Sorgente.

Per quanto riguarda l'output, la quantità di fatti prodotti dalla computazione degli algoritmi si è dimostrata molto minore rispetto alla quantità di dati grezzi iniziali, per questo motivo si è pensato di mantenere il Data Mart al di fuori da HDFS, continuando ad utilizzare la tecnologia PostgreSQL per la memorizzazione dei Data Mart. L'utilizzo di tecnologie come

Hive e Impala può essere considerato nel momento in cui, i fatti che vengono rappresentati all'interno del Data Mart, provengono direttamente dai dati raccolti sui dispositivi. Un esempio di ciò potrebbe essere riscontrato nel Data Mart che raccoglie i fatti relativi alle temperature. Questi dati infatti non subiscono una fase di elaborazione da parte degli algoritmo, e per questo possono essere reperibili direttamente dai dati grezzi raccolti.

Nel paragrafo successivo verranno forniti i dettagli delle logiche di Map e di Reduce implementate.

3.4.2 Formalizzazione Logica di MapReduce

Avendo a che fare con algoritmi termodinamici, che computano sulla base di valori appartenenti a un particolare intervallo temporale, sono state svolte tutta una serie di valutazioni, al fine di sviluppare la logica MapReduce.

Dato un particolare intervallo temporale $[\mathbf{T_{inizio}}, \mathbf{T_{fine}}]$, si suppone che gli algoritmi termodinamici sviluppati, per riuscire a identificare gli eventi di interesse (ad esempio di *refill*), svolgono un certo numero di **valutazioni**, che si ripetono secondo una determinata **cadenza**. Per ogni cadenza di valutazione, l'algoritmo svolge la propria computazione sulla base delle misurazioni che si sono verificate all'interno dell'intervallo $[\delta_1, \delta_2]$ della cadenza (vedi Figura3.12).

Figura 3.12: Logica di Valutazione

Alla luce delle considerazioni appena fatte, è possibile ipotizzare due diverse logiche MapReduce, che si differenziano principalmente per il livello di dettaglio di esecuzione degli algoritmi:

- Mantenendo la logica utilizzata nella soluzione precedente, per ogni dispositivo viene lanciata l'esecuzione dell'algoritmo termodinamico, a cui vengono passati tutte le misurazioni dell'intervallo temporale $[T_{inizio}, T_{fine}]$;

- Si potrebbe alleggerire il carico computazionale, eseguendo per ogni cadenza di valutazione di ogni dispositivo, l'algoritmo termodinamico che provvederà all'elaborazione dei dati che appartengono all'intervallo $[\delta_1, \delta_2]$.

Entrambe le logiche possono essere applicate, ma ognuna delle due presenta vantaggi e svantaggi che occorre valutare al fine di poter fare una scelta accurata.

Nella prima ipotesi, l'esecuzione degli algoritmi, per ogni dispositivo, sull'intero arco temporale, non richiede la replicazione dei dati, perché vi sarà sempre e solo una istanza dell'algoritmo in esecuzione sullo stesso dispositivo. Questo da un lato evita il sovraccarico della rete, dall'altro però limita la scalabilità, perché al massimo vi possono essere tanti algoritmi in parallelo pari al numero di dispositivi da monitorare, ovvero $\#Dispositivi$.

La seconda ipotesi invece, da un lato tende a sovraccaricare la rete del cluster, per via della replicazione dei dati, necessari alle varie istanze dell'algoritmo che, sullo stesso dispositivo, elaborano cadenze di valutazione distinte, ma che possono presentare intervalli di valutazione sovrapposti. Dall'altro lato invece, la possibilità di avere un'istanza dell'algoritmo per ogni cadenza di valutazione, consente una scalabilità molto maggiore, rispetto all'ipotesi precedente, che risulta pari al $\#Dispositivi * \#Cadenze$.

Fra le due ipotesi si è scelto di implementare la logica di MapReduce privilegiando la scalabilità. Questo perché la possibilità di scalare, non solo al crescere dei dispositivi, ma anche all'aumentare del numero dei giorni, che costituiscono l'intervallo totale di elaborazione, è un aspetto importante per il progetto di monitoraggio dei bottle cooler. Inoltre, se la soluzione viene progettata in maniera tale da poter parametrizzare, non solo l'intervallo totale, ma anche il numero di cadenze e l'intervallo $[\delta_1, \delta_2]$, ci si rende conto che, è possibile ricreare le condizioni per cui, su un dispositivo, viene mandata in esecuzione una sola istanza dell'algoritmo, che considera l'intero intervallo temporale. In questo modo si riproduce la prima ipotesi, per cui viene lanciata una sola istanza dell'algoritmo per ogni dispositivo.

Descritta in linea di massima della logica che si è scelto di implementare, è possibile analizzare singolarmente le logiche delle fasi di map e di reduce.

Mapper

La fase di Map, ha come obiettivo la riorganizzazione e la replicazione dei dati necessari alle singole istanze dell'algoritmo. Affinché tale fase svolga effettivamente l'attività necessaria alla soluzione, occorre progettare nei minimi dettagli la logica, che andrà poi implementata all'interno del Mapper di Hadoop.

All'avvio di una qualsiasi applicazione MapReduce, il Job provvede a suddividere l'input in split, per ognuno dei quali viene allocata un'istanza del Mapper di Hadoop. Il Mapper in questo contesto, è stato progettato in maniera tale da elaborare ogni singola riga dello split assegnato. Per ognuna delle righe dello split, viene automaticamente invocata la funzione di *map* che, implementata all'interno del Mapper, ne ingloba la logica.

L'**Algoritmo 1** mostra, in pseudocodice, la logica che sta alla base del Mapper, implementata mediante la funzione *map*. Come è possibile osservare, la map riceve in input una delle righe dello split, come ad esempio:

dev; var; time; value
182; 136; 2014-12-02 22:56:00.000; 21.2

caratterizzata rispettivamente dall'identificativo del device, dall'identificativo della variabile campionata, dall'istante di campionamento e dal valore che la variabile ha assunto in quel determinato momento. Queste informazioni sono separata dal carattere di separazione ";".

La funzione di *map*, per ogni intervallo di valutazione, verifica se il valore campionato, per quella particolare variabile, è compreso o meno nell'intervallo. Nel momento in cui viene verificato che tale valore campionato è di competenza di un particolare dell'intervallo di valutazione, la funzione di map provvede a genera una coppia chiave/valore per qual campionamento, e lo predispone in output per la successiva fase di reduce.

Affinché venga lanciata una istanza dell'algoritmo per ogni intervallo di valutazione di ogni dispositivo, la chiave generata dalla funzione di *map* è stata pensata come la composizione tra l'identificativo del device d e l'i-esima valutazione. In questo modo, durante la fase di reduce, per ogni

input : $(dev, var, time, value)$ dove:
dev $\rightarrow$ device su cui è stato eseguito il campionamento
var $\rightarrow$ variabile campionata
time $\rightarrow$ tempo di campionamento
value $\rightarrow$ valore campionato

$[\mathbf{T_{inizio}}, \mathbf{T_{fine}}]$ $\rightarrow$ intervallo temporale di riferimento per l'algoritmo
cadenza $\rightarrow$ cadenza di valutazione dell'algoritmo
$\delta_1, \delta_2 \rightarrow$ intervalli temporale da considerare prima e dopo ogni cadenza
$[\mathbf{N_{from}}, \mathbf{N_{to}}]$ $\rightarrow$ intervallo contenente le N_{to}-N_{from}+1 valutazioni da considerare

output: (dev, i)
$(time, var, value)$

funzione map begin
if *process*$((dev, var, time, value))$ **then**
  for $i \leftarrow N_{\text{from}}$ **to** N_{to} **do**
    $t_{\delta 1} \leftarrow T_{\text{inizio}} + i * cadenza - \delta_1$;
    $t_{\delta 2} \leftarrow T_{\text{inizio}} + i * cadenza + \delta_2$;
    if $t_{\delta 1} \leq time \leq t_{\delta 2}$ **then**
      `return(key(`*dev,i*`),value(`*time,var,value*`))`;
    end
  end
end
end map;

Algorithm 1: Logica del Mapper

coppia chiave/valore distinta, passata in output dalla fase di map, verrà lanciata in esecuzione un'istanza l'algoritmo termodinamico.

Come è possibile osservare dal modello, le attività appena descritte vengono svolte dal Mapper, se e solo se, l'input $(dev, var, time, value)$ supera la procedura di *process*. Questa funge da eventuale meccanismo di filtraggio, validazione, trasformazione dell'input. Come è stato detto nei paragrafi precedenti, alcuni algoritmi termodinamici non utilizzano tutte le misurazioni raccolte dai sensori, per questo motivo, un meccanismo in grado di filtrare i campioni delle misurazioni non necessari evita il passaggio agli algoritmi di input non necessari alla computazione.

I parametri $[\mathbf{T_{inizio}}, \mathbf{T_{fine}}]$, **cadenza**, δ_1, δ_2, $[\mathbf{N_{from}}, \mathbf{N_{to}}]$ vengono passati al Mapper, dal Job principale dell'applicazione MapReduce, in fase di avvio dell'applicazione.

Reducer

L'**Algoritmo 2** mostra la logica che sta alla base del Reduce. Al

input : (dev, i) $set(time, var, value)$

$[\mathbf{T_{inizio}}, \mathbf{T_{fine}}]$ → intervallo temporale di riferimento per l'algoritmo
cadenza → cadenza di valutazione dell'algoritmo
δ_1, δ_2 → intervalli temporale da considerare prima e dopo ogni cadenza
$[\mathbf{N_{from}}, \mathbf{N_{to}}]$ → intervallo contenente le N_{to}-N_{from}+1 valutazioni da considerare

output: (dev, i) $(outTime, outVar, outValue)$

funzione reduce begin
$(outTime, outVar, outValue) \leftarrow$ `elabora(`*set(time,var,value)*`)` ;
`return(key(`*dev,i*`),value(`*outTime,outVar,outValue*`))`;
end reduce;

Algorithm 2: Logica del Reduce

termine della computazione della fase di map, i singoli reduce provvedono al recupero dell'insieme delle coppia chiave/valore prodotte. Le elaborazioni sulle coppie chiave/valore vengono distribuite su diversi reducer, i quali

provvedono ad elaborare tutti i valori di output che sono caratterizzati dalla stessa chiave. Assegnando la stessa chiave ai valori che dovranno essere elaborati assieme, si ha la sicurezza che questi vengano elaborati tutti dallo stesso reducer.

Nel caso della soluzione prodotta, ogni reducer in possesso di un sotto-insieme di dati da elaborare, per ogni singola chiave distinta, provvede a richiamare la funzione *reduce*. Questa si occupa di lanciare l'esecuzione dell'algoritmo sull'insieme di valori campionati che, condividendo la stessa chiave, appartengono allo stesso intervallo di valutazione del medesimo dispositivo. Il risultato prodotto dall'elaborazione dell'algoritmo viene poi salvato su HDFS.

3.4.3 Modello dei Dati

Ogni volta che si vuole sviluppare una soluzione MapReduce occorre ragionare sulla scelta dei formati in Input e di Output della Map e della Reduce. Questo perché il framework di Hadoop, per agevolare la trasmissione dei dati tra i nodi del cluster, impone che i dati in Input e Output del Mapper e del Reduce implementino l'interfaccia *org.apache.hadoop.io.Writable*. Sia la Map che la Reduce necessitano entrambi di una coppia di formati in input e in output che andranno a specificare il tipo delle coppie chiave/valore utilizzate in entrambe le fasi.

Figura 3.13: Formati di Input e Output

La Figura3.13 mostra nel dettaglio le coppie dei formati di input e di output definiti nella soluzione. In particolare, la coppia di formati:

- "$LongWritable, Text$", per l'input della Map, identifica la singola riga del file di input del Job. Nel dettaglio, $LongWritable$ rappresenta l'indice di riga, invece $Text$ viene utilizzato per rappresentare il contenuto della riga;

- "$SingleValutationOfDevice, VariableValueTime$", per l'input/output intermedio fra la Map e la Reduce, è stato sviluppato appositamente per la corrente soluzione, a partire dall'interfaccia Writable definita da Hadoop. Nel dettaglio, $SingleValutationOfDevice$ rappresenta la chiave, costituita dall'identificativo del dispositivo e l'i-esima valutazione, invece $VariableValueTime$ rappresenta il valore associato alla chiave, ed è costituito dall'identificativo della variabile e dal valore assunto da questa in uno specifico istante temporale. La Figura3.14 mostra il modello utilizzato per il formato delle coppie chiave/valore intermedie alla Map e alla Reduce.

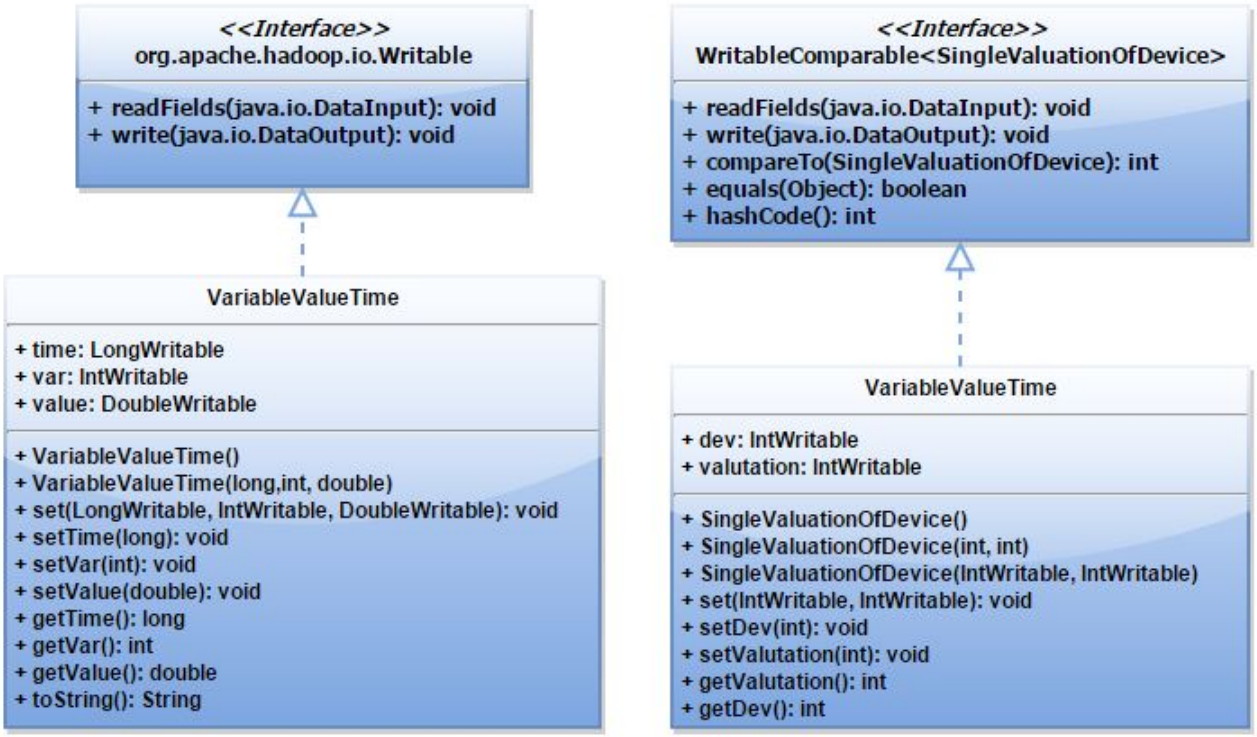


Figura 3.14: *thesis.etlmr.types.io*

- "$Text, Text$", per l'output della Reduce, identifica la coppia chiave/-valore restituita in output a seguito dell'elaborazione dell'algoritmo.

A seconda dell'algoritmo lanciato in esecuzione, l'output assumerà significati diversi (Es. Rifill, PotentialSale, ecc..).

Definita la logica di map e di reduce, e i rispettivi formati di input e di output, è necessario definire il modello dei dati che sono alla base della logica. Uno dei concetti su cui ruota tutta la logica della Map è quello di **singolo campione**. Questo rappresenta il dato di input elaborato da ogni singola funzione di *map*. La Figura3.15 mostra come tale concetto è stato modellato. In particolare, tramite *Sample* è possibile rappresentare il singolo campione, associato ad una particolare variabile *var*, che è stato raccolto sul dispositivo *dev* in un particolare istante temporale *TimeSample*.

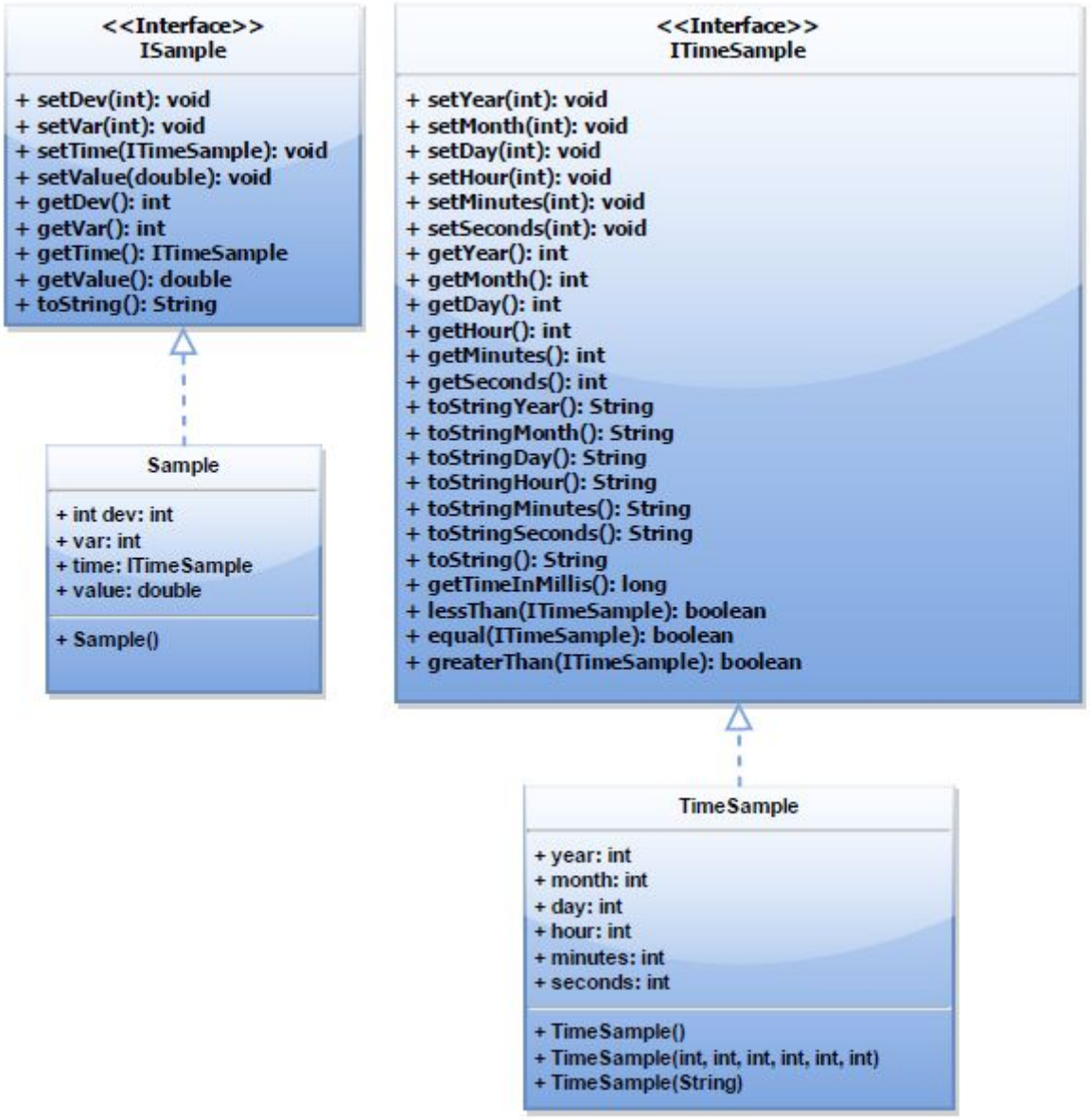


Figura 3.15: *thesis.etlmr.types*

3.4.4 Implementazione

Nel seguente paragrafo vengono mostrate le principali porzioni di codice, che hanno il compito di implementare la logica di MapReduce progettata. Per ognuna verranno evidenziati le particolarità e i punti fondamentali.

Driver

Ogni applicazione MapReduce è caratterizzata da un *Driver*, che funge da *entry point* per l'applicazione. In generale ogni Driver deve predisporre il Job, i parametri necessari ai Mapper e ai Reduce, gestire il contesto di riferimento, specificando le classi e i formati di input e output.

Nella soluzione, il Drive *ComputeAlgorithm* implementato svolge le seguenti attività:

- *Recupera gli argomenti di input necessari al Job*
  Affinché possa essere applicata la logica MapReduce sui singoli intervalli di valutazione dei dispositivi, è stata definita una serie di parametri che l'utente, in fase di avvio dell'applicazione, deve fornire affinché la computazione possa essere eseguita. I parametri in questione sono:

  - il tempo di inizio e di fine che caratterizzano l'intervallo [$\mathbf{T_{inizio}}$, $\mathbf{T_{fine}}$];
  - la cadenza di valutazione **cadenza**;
  - i delta δ_1, δ_2 che identificano l'intervallo di ogni valutazione;
  - la classe di "**preprocess**" da applicare ad ogni singolo campionamento;
  - infine, la classe dell'algoritmo **alg** che deve essere utilizzata per l'elaborazione delle misurazioni.

  La gestione da parte dell'utente dei parametri appena citati, ha permesso di realizzare un'applicazione in grado di fungere da framework per un numero maggiore di algoritmi.

- *Predispone le proprietà di configurazione per il Mapper e il Reducer*
  I parametri passati come argomenti di input all'applicazione, sono necessari alla computazione sia del Mapper che del Reducer; per questo motivo vengono definite delle proprietà di configurazioni, all'interno

delle quali vengono riposti i valori degli argomenti di input. Le proprietà di configurazione, essendo parte del contesto di esecuzione del Job, sono visibili sia al Mapper che al Reducer.

- *Predispone il Job*
  Durante tale predisposizione vengono definite le specifiche necessarie all'esecuzione dello stesso Job MapReduce. In particolare, vengono specificate le classi che implementano la logica del Mapper e del Reducer (*setMapperClass*() e *setReducerClass*()), i formati di input e di output definiti per il Mapper e il Reduce, infine i percorsi di input e di output dei dati, dove rispettivamente vengono prelevati i dati di input per l'elaborazione e posizionati i risultati ottenuti dagli algoritmi.

- *Avvio del Job*
  Al termina della predisposizione del Contesto e del Job, il Driver provvede ad avviare l'applicazione MapReduce implementata.

Qui di seguito è possibile osservare l'implementazione del Driver.

```
   /* Driver */
public class ComputeAlgorithm {
  public static void main(String[] args){
    Configuration conf = new Configuration();
      /* Recupero Argomenti di Input */
    String[] otherArgs = new
        GenericOptionsParser(conf,args).getRemainingArgs();
    CommandLine cmd = parseArgsInInput(otherArgs);
    String alg = cmd.getOptionValue("alg");
    String tinizio = cmd.getOptionValue("tinizio");
    String tfine = cmd.getOptionValue("tfine");
    String preprocess =
        cmd.getOptionValue("preprocess","DefaultPreprocess");
    long cad = Long.parseLong(cmd.getOptionValue("cadenza"));
    long d1 = Long.parseLong(cmd.getOptionValue("delta1"));
    long d2 = Long.parseLong(cmd.getOptionValue("delta2"));
    long n = numOfValutation(new TimeSample(tinizio),new
        TimeSample(tfine),cad,d1,d2);
    long nFrom = numFrom(cad,d1);
    long nTo = numTo(cad,d2,n);
      /* Set delle Configuration Property per il Mapper e il
         Reducer */
    conf.setStrings("Tinizio", tinizio);
    conf.setStrings("Tfine", tfine);
    conf.setLong("cadenza", cad);
    conf.setLong("delta1", d1);
    conf.setLong("delta2", d2);
```

```
    conf.setStrings("alg", alg);
    conf.setLong("nFrom", nFrom);
    conf.setLong("nTo", nTo);
    conf.setStrings("preprocess", preprocess);
      /* Prediposizione del Job */
    Job job = new Job(conf, "ComputeAlgorithm");
    job.setJarByClass(ComputeAlgorithm.class);
    job.setMapperClass(SetInputDataMapper.class);
    job.setReducerClass(ExecAlgorithmReducer.class);
    job.setMapOutputKeyClass(SingleValuationOfDevice.class);
    job.setMapOutputValueClass(VariableValueTime.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new
        Path(cmd.getOptionValue("in")));
    FileOutputFormat.setOutputPath(job, new
        Path(cmd.getOptionValue("out")));

    if(job.waitForCompletion(true)){
      System.exit(0);
    }else{
      System.exit(1);
    }
  }
}
```

Mapper

All'avvio del Mapper, secondo le specifiche delle librerie di Hadoop, viene eseguito il *setup*. Qui dopo aver prelevato dalle proprietà di configurazione i parametri necessari al Mapper, si provvede ad istanziare, mediante *setPreprocessClass*, l'oggetto "*preprocess*" che implementa la classe astratta *AbstractPreprocess*. Attraverso l'oggetto *preprocess* è possibile applicare ad ogni singolo campione, elaborato dalla *map*, le operazioni di filtraggio, trasformazione e manipolazione. Tali operazioni dovranno essere definite all'interno della classe che estende la classe astratta *AbstractPreprocess*, e che viene fornita come parametro in ingresso all'applicazione MapReduce.

La Figura3.16 mostra la struttura della classe *AbstractPreprocess*. La classe $DefaultPreprocess$ viene utilizzata come "preprocess" nel caso in cui non si voglia applicare nessun tipo di filtro o trasformazione al singolo campione. Se all'applicazione MapReduce non viene fornita in input nessuna classe di "preprocess" viene automaticamente utilizzata la classe $DefaultPreprocess$ e contestualmente non viene applicata nessuna logica ai campioni gestiti dalla *map*.

Applicati eventuali filtri e trasformazioni, il singolo campione viene poi sottoposto alla logica di replicazione e organizzazione definita per la preparazione dell'input per le singole istanze l'algoritmo.

Qui di seguito è possibile osservare l'implementazione della logica del Mapper.

```
   /* Mapper */
public class SetInputDataMapper extends
Mapper<LongWritable,Text,SingleValuationOfDevice,VariableValueTime>{

  private SingleValuationOfDevice keyOut = new
      SingleValuationOfDevice();
  private VariableValueTime valueOut = new VariableValueTime();
  private ISample sample = new Sample();
  private long nFrom;
  private long nTo;
  private long tinizio;
  private long delta1;
  private long delta2;
  private long cadenza;
  private AbstractPreprocess preprocess;

  @Override
  protected void map(LongWritable key,Text value,Context context){
      /* Processa il singolo campione */
    sample = preprocess.process(value.toString());
    if(sample != null){
     /* Implementazione della logica di Map */
     long tDelta1;
     long tDelta2;
     long time = sample.getTime().getTimeInMillis();
     for(long i = nFrom; i <nTo ; i++){
       tDelta1 = tinizio + (i*cadenza*1000) - (delta1*1000);
       tDelta2 = tinizio + (i*cadenza*1000) + (delta2*1000);
       if(time >= tDelta1 && time <= tDelta2){
         keyOut.setValutation((int) i);
         keyOut.setDev(sample.getDev());
         valueOut.setTime(time);
         valueOut.setVar(sample.getVar());
         valueOut.setValue(sample.getValue());
         context.write(keyOut, valueOut);
       }
     }
    }
  }

  @Override
  protected void setup(Context context){
    super.setup(context);
      /* Get delle Configuration Property */
    Configuration conf = context.getConfiguration();
    this.nFrom = conf.getLong("nFrom", -1);
```

```
    this.nTo = conf.getLong("nTo", -1);
    this.cadenza = conf.getLong("cadenza", -1);
    this.delta1 = conf.getLong("delta1", -1);
    this.delta2 = conf.getLong("delta2", -1);
    String tinizio = conf.getStrings("Tinizio","")[0];
    this.tinizio = new TimeSample(tinizio).getTimeInMillis();
      /* Istanzia la classe "preprocess" */
    setPreprocessClass(conf.getStrings("preprocess")[0]);
  }

  protected void setPreprocessClass(String classPreprocess) {
    if(classPreprocess.equals("DefaultPreprocess")){
      preprocess = new DefaultPreprocess();
    }else{
      Class<?> preprocessClass = Class.forName(classPreprocess);
      preprocess = (AbstractPreprocess)
          preprocessClass.getConstructor().newInstance();
    }
  }
} /*Fine Mapper*/
```

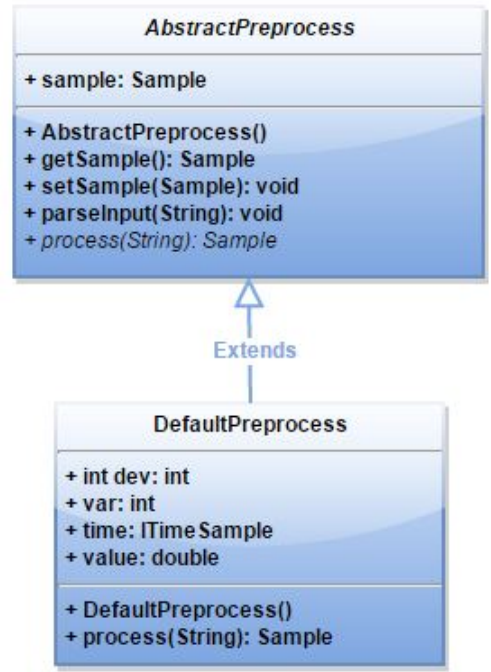


Figura 3.16: *thesis.etlmr.mapper.preprocess*

Reducer

L'ultima fase dell'applicazione è quella del Reduce. Anche in questo caso, le librerie di Hadoop impongono l'esecuzione della procedura di *setup* all'avvio di ogni istanza di Reducer. Durante la procedura di setup, si provvede al recupero delle proprietà di configurazione impostate dal Driver. Fra queste proprietà, vi è la stringa che identifica la classe di riferimento per l'algoritmo che si intende eseguire sui dati. Tale classe viene istanziata in fase di setup e ad ogni ciclo di reduce, viene utilizzata per l'elaborazione dei dati di output della fase di map.

La Figura3.17 mostra la classe astratta *AbstractAlgorithm*, dalla quale si estende per implementare la classe, che dovrà incapsulare la logica specifica per il determinato algoritmo che si vuole realizzare. Il metodo che ingloba la logica dell'algoritmo che si vuole eseguire è *elabora*. Questo utilizza *InputDataset* come struttura dati in input e *OutputDataset* come struttura di output. La possibilità di avere una classe astratta *AbstractAlgorithm*, che definisce l'interfaccia che deve essere rispettata per l'implementazione dell'algoritmo, consente di poter utilizzare la soluzione MapReduce sviluppata con un numero molto maggiori di algoritmi, rispetto a quelli già realizzati. Questi dovranno però rispettare i vincoli implementativi dati dalla classe astratta *AbstractAlgorithm*.

Qui di seguito è possibile osservare l'implementazione della logica del Reducer.

```
public class ExecAlgorithmReducer extends
Reducer<SingleValuationOfDevice,VariableValueTime,Text,Text>{

  private Text keyOut = new Text();
  private Text result = new Text();
  private Configuration conf;
  private String nameAlg;
  private Class<?> algClass = null;
  private AbstractAlgorithm algIstance = null;
  private long cadenza;
  private String tinizio;
  private int valutation;
  private int codiceDevice;
  private long t;
  private long tStart;
  private TimeSample tempoValutazione;
  private Iterator<VariableValueTime> itr;
  private InputDataset inputSet;
  private OutputDataset outputSet;

  @Override
```

```
  protected void reduce(SingleValuationOfDevice key,
      Iterable<VariableValueTime> values,
      Reducer<SingleValuationOfDevice, VariableValueTime, Text,
      Text>.Context context){

    valutation = key.getValutation();
    codiceDevice = key.getDev();
    t = tStart + (valutation * cadenza*1000);
    tempoValutazione.setTimeSample(t);
      /* Predisposto l'input dell'algoritmo */
    inputSet.setInputDataset(values);
    algIstance.set(conf, tempoValutazione, codiceDevice);
      /* Esecuzione dell'algoritmo */
    outputSet = algIstance.elabora(inputSet);

    itr = outputSet.getOutputDataset().iterator();
    while(itr.hasNext()){
      keyOut.set(valutation+";"+codiceDevice);
      result.set(itr.next().toString());
      context.write(keyOut, result);
    }
  }

  @Override
  protected void setup(Reducer<SingleValuationOfDevice,
      VariableValueTime, Text, Text>.Context context){
    super.setup(context);
    tempoValutazione = new TimeSample();
    inputSet = new InputDataset();
      /* Get delle Configuration Property */
    conf = context.getConfiguration();
    nameAlg = conf.getStrings("alg","")[0];
    cadenza = conf.getLong("cadenza", -1);
    tinizio = conf.getStrings("Tinizio","")[0];
    tStart = new TimeSample(tinizio).getTimeInMillis();
      /* Istanzia la classe "algorithm" */
    algClass = Class.forName(nameAlg);
    algIstance = (AbstractAlgorithm)
        algClass.getConstructor().newInstance();
  }
}
```

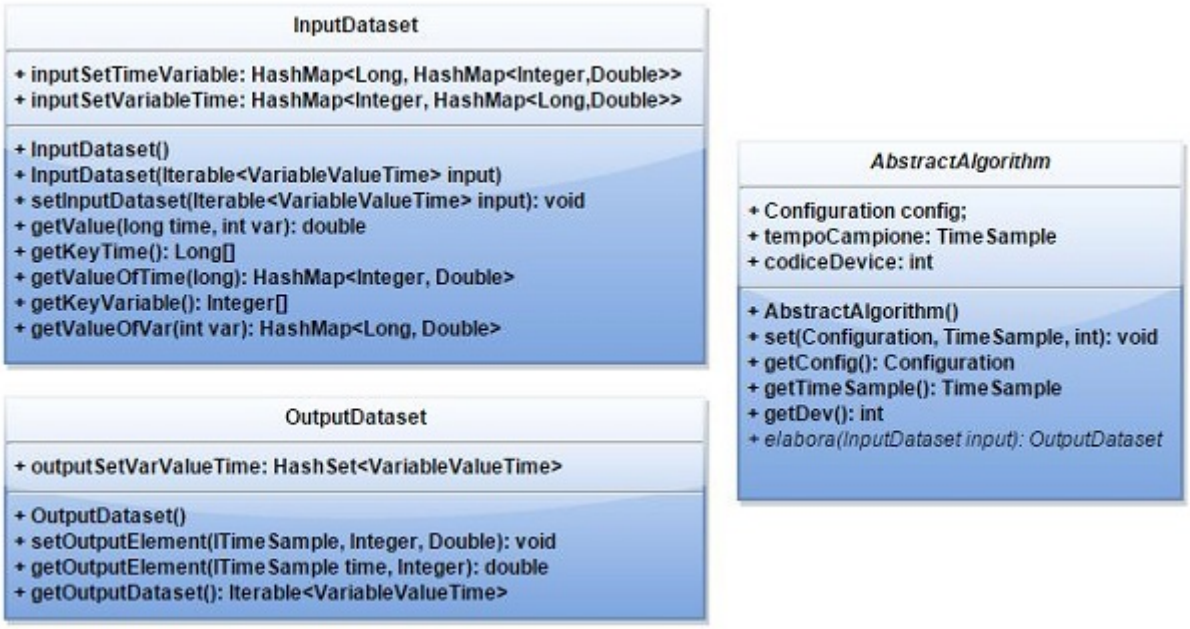


Figura 3.17: *thesis.etlmr.types.algorithms* e *thesis.etlmr.algorithms*

3.4.5 Test e Analisi delle Performance

Dopo aver terminato lo sviluppo sia della Fase 1, relativo al flusso tradizionale di ETL, sia della Fase 2, relativo alla soluzione Big Data, si è passati allo svolgimento di alcuni test, per verificare quali siano gli impatti sulle performance, all'aumentare dei dati, per ognuna delle due soluzioni.

Inizialmente sono stati generati 5 set di dati, ognuno dei quali contiene le misurazioni delle variabili raccolte sui diversi bottle cooler, nell'arco di una giornata (24 ore). I cinque file si differenziano per il numero di Bottle Cooler per i quali sono stati raccolti i dati. In generali, la quantità di bottle cooler che caratterizza i cinque set di dati è:

- 10 Bottle Cooler ($\approx$ 12MB);
- 100 Bottle Cooler ($\approx$ 120MB);
- 1.000 Bottle Cooler ($\approx 1,2$GB);
- 10.000 Bottle Cooler ($\approx$ 12GB);
- 100.000 Bottle Cooler ($\approx$ 120GB);

Questi data set di misurazioni sono stati utilizzati sia per i test che riguardano il flusso tradizionale di ETL, sia per quelli svolti con la soluzione MapReduce. In totale sono stati svolti 8 esperimenti:

- 5 di questi riguardano l'applicazione della soluzione MapReduce sviluppata sui cinque data set prodotti;
- i restanti 3 riguardano l'applicazione del flusso tradizionale di ETL ai data set che contengono le misurazioni di 10, 100 e 1.000 bottle booler. Visti i tempi impiegati dal flusso tradizionale di ETL non si è ritenuto importante procedere ai test sui data set contenenti rispettivamente le misurazioni di 10.000 e 100.000 bottle cooler.

Applicando i primi 3 data set (10,100,10.000) al flusso tradizionale di ETL, si sono ottenuti i seguenti risultati:

Num.Device	Tempo ETL (ms)	Tempo ETL (m)
10	196.800	3,28
100	700.200	11,67
1.000	26.520.000	442,0

Come è possibile notare, applicando il flusso tradizionale di ETL, moltiplicando per un fattore 100x il numero di bottle cooler, i tempi necessari per il completamento del flusso, risultano passare da poco più di 3 minuti ad all'incirca 7 ore e mezzo, con un fattore di moltiplicazione di circa 135x. All'aumentare dei dati, i tempi di calcolo aumentano esponenzialmente a causa delle operazioni di join applicate sui dati in fase di organizzazione dell'input e alla sequenzializzazione della computazione degli algoritmi, dispositivo per dispositivo. Occorre tenere in considerazione che parte dell'inefficienza riscontrata, è dovuta all'utilizzo di un macchina caratterizzata da poca memoria ram e pochi core a disposizione.

I test sulla soluzione MapReduce sono stati svolti su di un cluster di 7 nodi sui quali è stata installata la distribuzione Cloudera CDH5. L'elaborazione dei 5 data set, mediante la soluzione MapReduce, ha portato i seguenti risultati:

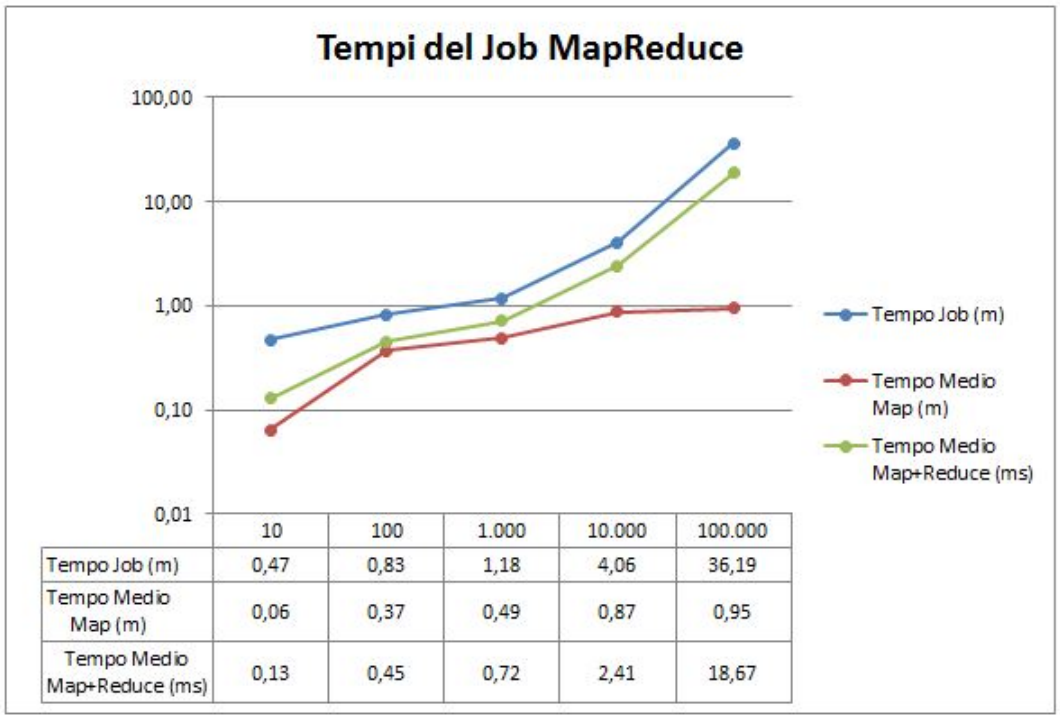
			Millisecondi			
Num. Device	Tempo Job	T.Medio Map	Num. Map	T.Medio Reduce	Num Reduce	T.Medio Setup
10	28443	3820	1	3943	28	20680
100	49868	22312	1	4941	28	22614
1.000	70858	29691	8	13428	28	27739
10.000	243706	52083	78	92359	28	99264
100.000	2171652	56957	906	1063455	28	1051240
			Minuti			
Num. Device	Tempo Job	T.Medio Map	Num. Map	T.Medio Reduce	Num Reduce	T.Medio Setup
10	0,47	0,06	1	0,07	28	0,34
100	0,83	0,37	1	0,08	28	0,38
1.000	1,18	0,49	8	0,22	28	0,46
10.000	4,06	0,87	78	1,54	28	1,65
100.000	36,19	0,95	906	17,72	28	17,52

Diversamente dalla soluzione precedente, in questo caso è stato possibile ricavare i tempi medi legati alle fasi dell'elaborazione, ovvero i tempi medi relativi alle fasi di map e di reduce. A partire da queste è stato poi possibile

ottenere il tempo medio di setup dell'ambiente su cui sono state eseguite le singole applicazioni MapReduce.

$$T.MedioSetup = TempoJob - T.MedioMap - T.MedioReduce$$

Il tempo di setup ci permette di capire, per ogni singola esecuzione dell'applicazione MapReduce, quanto tempo viene utilizzato per la computazione e quanto invece, per l'organizzazione dell'ambiente di esecuzione. Questo è importante per poter confrontare i tempi ottenuto dalle due soluzioni. La Figura3.18 mostra il grafico ottenuto dai risultati delle esecuzioni della



	10	100	1.000	10.000	100.000
Tempo Job (m)	0,47	0,83	1,18	4,06	36,19
Tempo Medio Map (m)	0,06	0,37	0,49	0,87	0,95
Tempo Medio Map+Reduce (ms)	0,13	0,45	0,72	2,41	18,67

Figura 3.18: Tempi del Job MapReduce

soluzione MapReduce. Come è possibile osservare, all'aumentare dei dati i tempi crescono in maniera lineare; inoltre è possibile notare che al di sotto dei 1.000 dispositivi, i tempi di computazione risultano minori dei tempi di setup necessari all'ambiente per la configurazione. Questo porta a concludere che, per una quantità di dati al di sotto di una certa soglia, i tempi di

latenza dovuti al cluster superano i tempi di computazione. Invece, all'aumentare del volume dei dati, da un lato il tempo di setup tende a diminuire, dall'altro il tempo di computazione di MapReduce, rispetto al tempo totale del Job, tende ad aumentare.

Ora è possibile confrontare i tempo ottenuti dalle diverse soluzioni.

Num. Device	Tempo ETL(m)	Tempo Job MapReduce(m)	Tempo Medio Map e Reduce(m)
10	3,28	0,47	0,13
100	11,67	0,83	0,45
1000	442,00	1,18	0,27

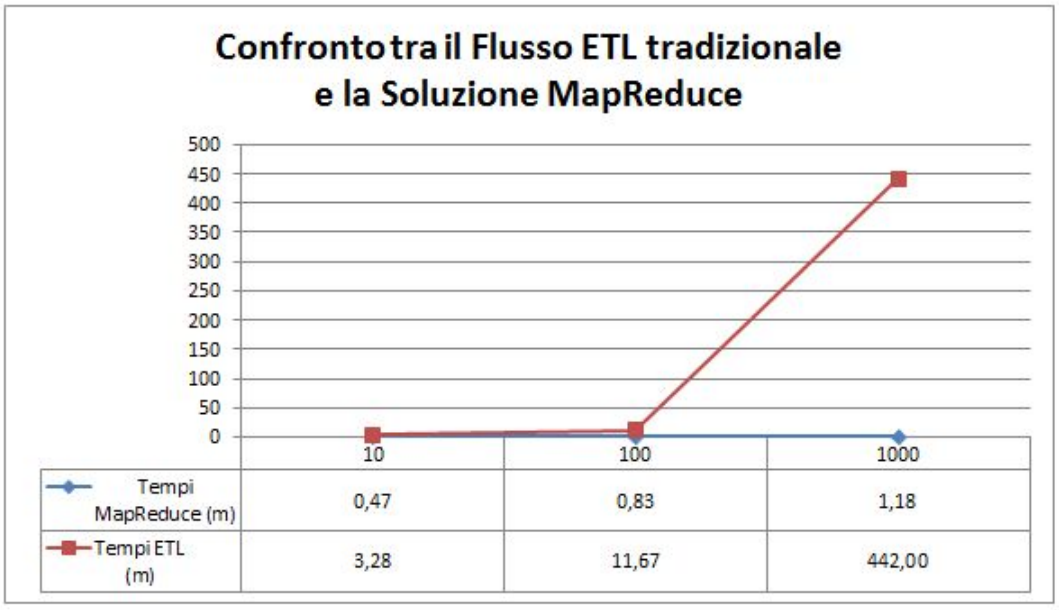


Figura 3.19: Confronto tra ETL Tradizionale e Job MapReduce

Come è possibile notare, all'aumentare dei dispositivi, e quindi contetualmente alla quantità di dati, i tempi crescono in maniera esponenziale. Tale differenza è data principalmente da due aspetti. Il primo riguarda l'eliminazione di tutte quelle operazioni di Join che appesantiscono la computazione nei tradizionali flussi di ETL. Il secondo aspetto è dato dalla possibilità di sfruttare il parallelismo, offerto dalla piattaforma di Hadoop, sia per la

riorganizzazione degli input per gli algoritmi, che per la relativa esecuzione. Anche considerando solo i tempo reali di map e reduce, escludendo i tempi di setup, le perfomance di computazione date dalla soluzione MapReduce sono indiscutibilmente migliori.

Conclusioni

L'avvento del fenomeno dei Big Data, ha portato con se un innumerevole quantità di aspetti e implicazioni che si sono percepite in diverse aree di business. Sia chi deve trarre informazioni da questa tipologia di dati, sia chi li deve gestire e manipolare, si trova si fronte a un mondo che, nonostante sia molto discusso e noto a tutti, presenta molti aspetti sconosciuti, anche per molti degli addetti ai lavori. Fra chi comprende tali aspetti invece, c'è chi concorda con essi e chi li considera solamente un fenomeno mediatico. Molte sono le domande che ci si pongono: "Cosa sono questi Big Data?", "Come li si può sfruttare?", "Si possono realmente utilizzare per un fine di più alto livello?". Una risposta definitiva per tutte queste domande ancora non c'è. Le molte aziende che, fino ad oggi, hanno fatto del "dato" il loro business, si trovano di fronte a un grosso cambio generazionale. Le sole tecniche tradizionale risultano non essere più sufficienti a soddisfare i requisiti di elaborazione e gestione dei Biga Data. Le tecnologie legate a questo fenomeno sono molte e molte altre nasceranno. Le nuove tecnologie Big Data non sono state sviluppare per sostituire le tecniche fino ad oggi utilizzate, ma per affiancare gli strumenti già in produzione al fine di estrapolare valore da questa nuova tipologia di dati. I "due mondi" dovranno coesistere e cooperare, cercando di sfruttare e fare emergere il meglio da ognuna. Non vi è un unico strumento al quale affidarsi, non vi sono regole o standard particolari che permettono di gestire questa moltitudine di tecnologie. Le aziende che percepiscono il valore legato ai Big Data, molto spesso non sanno quali strumenti adottare e quale soluzione instaurare per sfruttare la varietà e la quantità di dati che si presentano. Da tutto questo insieme, dal contesto emerge Hadoop, piattaforma di calcolo distribuita basata sul framework MapReduce. Hadoop, negli ultimi tempi, viene visto come lo strumento per i Big Data, utilizzato come base da molti vendors. Molto spesso, e in maniera errata, Hadoop viene confuso con i concetti di

Big Data, quando invece funge da strumento per la loro elaborazione e gestione. Hadoop non rappresenta uno standard da poter applicare con un certo metodo a progetti reali, inoltre il solo utilizzo di Hadoop potrebbe non essere sufficiente. Molto spesso vi è la necessità di ricorrere alle molte componenti che caratterizzano, il cosiddetto, ecosistema di Hadoop.

La necessità di sviluppare una soluzione in grado di gestire l'elaborazione degli algoritimi nell'ambito del monitoraggio dei bottle cooler, all'aumentare dei dispositivi dai quali vengono raccolti i dati, ha messo in luca la necessità di adottare strumenti e tecnologie Big Data a supporto delle presenti tecniche di BI. Il contesto progettuale che si è presentato, si è prestato molto bene alla progettazione e allo sviluppo di un'applicazione MapReduce. Non tutti i progetti Big Data si prestano ad essere applicati al contesto di MapReduce di Hadoop, in alcuni casi sono necessari strumenti di più alto livello, come ha esempio Hive e Impala, e molti altri. La soluzione sviluppata nel presente elaborato ha permesso di valutare quali sono gli impatti relativi allo sviluppo di una reale soluzione MapReduce, utilizzando direttamente le librerie messe a disposizione da Hadoop. Lo "scoglio" computazionale derivante dalla comprensione e alla progettazione della logica di MapReduce non è da sottovalutare. I dettagli tecnici legati ai meccanismi architetturali di MapReduce devono essere compresi prima di sviluppare una soluzione in grado di fornire le perfomance desiderate. Inoltre la velocità con cui tali strumenti cambiano e vengono aggiornati, nel corso delle settimane, potrebbe creare problematiche di comprensione e compatibilità. Componenti come Hive e Impala aggirano la problematica di sviluppare una soluzione direttamente in MapReduce, ma non è scontato che siano gli strumenti ideali per gli obiettivi che ci si è preposti. La soluzione Hadoop sviluppata inoltre, ci ha permesso di valutare come strumenti Big Data, come Hadoop e MapReduce, possano andare a svolgere le attività critiche che si possono presentare nei tradizionali flussi di ETL, all'aumentare dei dati. La soluzione sviluppata, all'aumentare dei dati, si è dimostrata essere più efficiente e performante rispetto alla soluzione già implementata, mediante gli strumenti tradizionali di ETL. All'aumentare dei dati, i flussi tradizionali di ETL si sono rilevati estremamente inefficienti, impiegando tempi di elaborazione non accettabili al fine degli obiettivi del progetto di monitoraggio dei bottle cooler. I risultati operativi derivanti dai test, hanno mostrato che l'utilizzo delle capacità computazionali offerte dal cluster in-house di Hadoop a disposizione, ha permette di ridurre i tempi di elaborazione e di computazione. La possibi-

lità inoltre, in sviluppi futuri, di sfruttare strumenti in cloud, come cluster e storage, consentirebbe di gestire dinamicamente, all'occorenza, l'aumento e la diminuzione di capacità di calcolo e storage. Questo permetterebbe di svincolarsi dagli aspetti tecnici ed economici dati dai cluster e piattaforme in-house. Occore considerare inoltre, che la scelta della piattaforma, sulla quale vengono svolti le soluzioni Big Data, è un fattore fondamentale, che impatta sui risultati delle perfomance.

Alla luce di tutto ciò, si può concludere che in presenza di dati considerati Big, i tradizionali strumenti devono essere affiancati da tecnologie in grado di gestire la varietà, la velocità e quantità che caratterizzano i dati. Strumenti Big Data, come Hadoop, come è stato dimostrato dal presente elaborato, permettono realmente di superare i limiti dati dai tradizionali sistemi. La mancanza però di uno standard e di metodologie di progettazione e sviluppo in ambito Big Data, crea problematiche che tendono a rallentare il processo di sviluppo di soluzioni Big e quindi, di conseguenza l'evoluzione del fenomeno dei Big Data. L'adozione di strumenti Big Data inoltre, come si è visto, richiede investimenti dal punto di vista infrastrutturale, come ad esempio l'acquisto di macchine per cluster oppure l'acquisto di servizi in cloud. Nonostante tutto ciò, come è possibile notare all'architettura progettata, le due tipologie di strumenti (Big Data e Tradizionali) continuano a coesistere e collaborare. Questo aspetto continuerà a presentarsi anche in altri contesti progettuali.

Bibliografia

[1] Eaton C., Deroos D., Deutsch T., Lapis G., Zikopoulos P., *Understanding Big Data - Analytics for Enterprise Class Hadoop and Streaming Data*, IBM, 2012

[2] Eaton C., Deroos D., Deutsch T., Lapis G., Zikopoulos P., *Understanding Big Data - Analytics for Enterprise Class Hadoop and Streaming Data*, IBM, 2012

[3] Rezzani A., *Big Data - Architettura, tecnologie e metodi per l'utilizzo di grandi basi di dati*, Maggioli Editore, 2013

[4] Apache Flume^{TM}, `http://flume.apache.org/`

[5] Big Data: riconoscerli, gestirli, analizzarli, *Dedagroup ICT Network*, `http://www.dedagroup.it/`

[6] Bigtable: A Distributed Storage System for Structured Data, *Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber*, Google Inc., 2006

[7] NoSql-Database, `http://nosql-database.org/`

[8] Apache Cassandra, `http://cassandra.apache.org/`

[9] MongoDB Manual, `http://docs.mongodb.org/manual/`

[10] Impala, `http://impala.io/index.html`

[11] R.L.Saltzer, I.Szegedi, P.De Schacht `Impala in Action - Querying and mining big data`, Manning Publications, 2014

[12] PrestoDB, http://prestodb.io/

[13] Osservatori.Net Digital Innovation, Big Data: Un Mercato in cerca d'autore http://www.osservatori.net/home, 10/12/2014

[14] Il sole 24 Ore, Big data in azienda? Cresce il mercato in Italia ma solo il 13% delle imprese ha uno scienziato dei dati http://www.infodata.ilsole24ore.com/

[15] Che cosa pensereste se vi dicessero che in Italia i BigData non esistono? , Luca Debiase http://blog.debiase.com/

[16] Big Data Live: casi di eccellenza, P.Pasini, A.Perego http://www.sdabocconi.it/, Osservatorio Business Intelligence, 2013

[17] Big Data Challenge: sfida a colpi di Big Data http://www.telecomitalia.com/

[18] Next-Generation Analytics and Platforms, TDWI Best Practices Report http://tdwi.org/Home.aspx, Dicembre 2014

[19] Chicago: City of Big Data, http://bigdata.architecture.org/

[20] How to Reconcile Big Data and Privacy, L. Greenemeier http://blogs.scientificamerican.com/ Marzo 2014

[21] Apache™ Hadoop®!, http://hadoop.apache.org/

[22] Hadoop: The Definitive Guide, Fourth Edition Tom white, Early Release April 2015

[23] Verifica Sperimentale di un Modello per MapReduce, Tesi di Laurea Magistrale di Paolo Rodeghiero, Marzo 2012

[24] The Hadoop Distributed File System, R.Chansler, H.Kuang, S.Radia, K.Shvachko, S.Srinivas

[25] Cloudera, http://www.cloudera.com/

[26] Hortonworks®!, http://hortonworks.com/

[27] Amazon Elastic MapReduce (AWS), `http://aws.amazon.com/elasticmapreduce/`

[28] MapR Technologies, `https://www.mapr.com/`

Ringraziamenti

Innanzitutto desidero ringraziare di cuori i miei genitori, Mauro e Flavia, per la persona che sono oggi, senza il loro aiuto e il loro amore non sarei diventata quella che sono. Mi hanno sempre sostenuto e sopportato, per questo e per altri infiniti motivi non finirò mai di ringraziarli ed amarli. Un grazie va al mio amore Francesco che mi ha sopportato in questi lungi mesi, sempre al mio fianco, con tanto amore. Il suo sostegno e il suo amore in moltissime circostanze mi hanno dato la forza di andare avanti. Un grazie al mio ometto Samuele, che nonostante stia crescendo a vista d'occhio rimarrà per sempre il mio piccolo Lele. Un grazie va alla mia nonna Giovanna che c'è sempre stata e sempre ci sarà, senza il suo grande aiuto questi ultimi mesi sarebbero stati diversi. Un grazie ai miei nonni Oreste e Laura che mi hanno sempre sostenuto. Ringrazio la migliore amica di sempre Valentina, che durante le lunghe telefonate, ha sempre sopportato le mie lamentele e mi ha sempre dato la forza di affrontare gli esami uno dopo l'altro. Un grazie va ai miei amici di corso Angelo e Pietro, che hanno reso speciali questi anni di università e mi hanno fatto conoscere Silvia e Martina due amiche altrettanto stupende e uniche. Desidero ringraziare il mio relatore, il Prof. Matteo Golfarelli, che mi ha guidato nella realizzazione della tesi con molta professionalità e pazienza. Un grazie all'amica e alla tutor migliore di sempre Daniela, che negli ultimi mesi mi ha guidato e dato forza nell'intraprendere questo nuovo percorso lavorativo. Ringrazio Alessandro per essere stato paziente e per avermi guidato durante tutto lo svolgimento del progetto. Un grazie va al mio co-relatore Paolo che mi ha sopportato e seguito durante tutto il periodo della tesi mostrandosi sempre collaborativo e sempre pronto a darmi suggerimenti. Desidero infine ringraziare tutti coloro che direttamente o indirettamente hanno sempre creduto in me e mi hanno sostenuto i questi anni di università.