ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

Progettazione e prototipazione di un sistema di Data Stream Processing basato su Apache Storm

Tesi in
Laboratorio di Basi di Dati

Relatore:
Prof. MATTEO GOLFARELLI

Co-relatore:
Dott. LORENZO BALDACCI

Presentata da:
ALESSIO ADDIMANDO

SESSIONE DI LAUREA I
ANNO ACCADEMICO 2015-2016

A chi ha avuto il coraggio di aiutarmi a crescere.

Sommario

Introduzione

Con l'avvento di Internet, il numero di utenti con un effettivo accesso alla rete e la possibilità di condividere informazioni con tutto il mondo è, negli anni, in continua crescita. Con l'introduzione dei social media, in aggiunta, gli utenti sono portati a trasferire sul web una grande quantità di informazioni personali mettendoli a disposizione delle varie aziende. Inoltre, il mondo dell'Internet Of Things, grazie al quale i sensori e le macchine risultano essere agenti sulla rete, permette di avere, per ogni utente, un numero maggiore di dispositivi, direttamente collegati tra loro e alla rete globale. Proporzionalmente a questi fattori anche la mole di dati che vengono generati e immagazzinati sta aumentando in maniera vertiginosa dando luogo alla nascita di un nuovo concetto: i *Big Data*. Un aumento così vertiginoso, tuttavia, ha da subito messo in luce un deficit nella capacità di storage che le tradizionali Basi di Dati relazionali offrono. Per far fronte a questa esigenza contesti aziendali ricorrono alle recentissime soluzioni NoSQL.

All'interno dei giganteschi dataset, inoltre, possono essere presenti informazioni che per le aziende rappresentano un vero e proprio strumento non solo per migliorare il rapporto col cliente o la sua gestione interna ma anche per avere un vantaggio verso aziende avversarie. Questi dati, elaborati correttamente, permettono quindi di ottenere delle informazioni di valore strategico che aiutano il processo di analisi decisionale aziendale a qualsiasi livello, dalla produzione di un prodotto fino all'azione di marketing che gli concerne.

Nasce, di conseguenza, la necessità di far ricorso a nuovi strumenti che possano sfruttare la potenza di calcolo oggi offerta dalle architetture più complesse che comprendono, sotto un unico sistema, un insieme di host utili per l'analisi (i.e. Cluster). Nel panorama più recente, sono stati introdotti, a tal fine, numerosissimi framework, proprietari e open-source, che agevolano l'intero processo. Tra i più utilizzati è importante mettere in risalto Hadoop, in generale considerato principale piattaforma modulare nel panorama dei Big Data, che fornisce la possibilità di elaborare grandi dataset sfruttando la potenza di calcolo di cluster di commodity hardware.

Lo stack di Hadoop è vario ed offre, oltre alla struttura di base, numerosi moduli utili per le diverse esigenze di analisi. A tal merito, una quantità di dati così vasta, routine se si parla di Big Data, aggiunta ad una velocità di trasmissione e trasferimento altrettanto alta, rende la memorizzazione completa dei dati malagevole, tantomeno se le tecniche di storage risultano essere i tradizionali DBMS.

Una soluzione relazionale classica, infatti, permetterebbe di processare dati solo su richiesta, producendo ritardi, significative latenze e inevitabile perdita di frazioni di dataset.
Occorre, perciò, far ricorso a nuove tecnologie e strumenti consoni a esigenze diverse dalla classica analisi batch che le procedure Hadoop offrono.
In particolare, è stato preso in considerazione, come argomento di questa tesi, il *Data Stream Processing.*
La grande quantità di dati, infatti, non sempre necessita di obbligatoria storicizzazione. Ai fini di analisi è possibile elaborare flussi continui di dati, elaborati direttamente in memoria, raccolti da sorgenti (e.g. sensori, web, architetture di rete) geograficamente distribuite che li producono con data rate non predicibili, al fine di ottenere risposte in modo tempestivo.
A tal fine, l'ecosistema Apache mette a disposizione uno dei migliori sistemi per il data stream processing: Apache Storm; utilizzato per la parte sperimentale, progettando e, successivamente, sviluppando un prototipo di un'applicazione riferita al caso di studio.
Come prima introdotto, la sorgente dei dati, in questa particolare branca d'analisi ha principalmente natura real-time, ossia la generazione avviene ad alta frequenza ininterrottamente.
In accordo con questa fondamentale caratteristica, si è scelto come caso di studio una delle sorgenti effettivamente "inesauribile": la rete Internet.
Il progetto, pertanto, affronta l'analisi quantitativa (i.e. statistiche) e qualitativa (i.e. individuazione di informazioni utili) dei netflow di dati scambiati dalla rete del campus universitario di Forlì-Cesena tra terminali interni, e verso l'esterno, individuandone sessioni di comunicazione. Quest'ultima, in aggiunta, deve essere intesa in chiave di *cybersecurity*, in quanto, in questa particolare applicazione, il processo di Analytics si impone, come primo scopo, l'individuazione, tra le sessioni, di eventuali situazioni "sospette" (i.e. intromissioni di malware, attacchi hacker).
La tesi è strutturata in 4 capitoli. Nel primo si fornisce un'introduzione al mondo dei Big Data, ai database *NoSQL*, fondamentali nella gestione e nella memorizzazione dei dati a livello di cluster, e ad Hadoop in generale. Nel secondo, viene offerto un approfondimento sul data stream processing, le sue specifiche applicazioni, le caratteristiche cardine e alcuni esempi in merito. Nel terzo capitolo, invece, si analizza la struttura di Apache Storm sia a livello di moduli da cui è costituito sia in termini di architettura run-time che permette l'esecuzione delle sue applicazioni. Nel quarto, e ultimo, capitolo è presente una descrizione della realizzazione del progetto, partendo dal design fino all'effettiva creazione del prototipo, le attività di test e i risultati ottenuti.

1. Evoluzione del data management: i Big Data

Se si dovesse cominciare a parlare di data management, la ricostruzione storica del concetto di dato ci riporta negli anni Sessanta, quando quest'ultimo non si discostava, in termini tecnici, da semplici serie di caratteri e cifre, memorizzati in forma statica. Una delle sfide maggiori, infatti, era cercare di creare una "relazione" tra i dati, inizialmente provenienti da schede perforate e, successivamente, da flat file, ovvero file non strutturati memorizzati su strutture, sostanzialmente hardware-oriented. Le uniche tecnologie disponibili, pertanto, permettevano di immagazzinare dati grazie all'ausilio di supporti magnetici che le aziende e le banche, uniche realtà dove era logico poter parlare di data storage, utilizzavano per un'analisi sommaria limitata alla loro semplice estrazione.
Qualche anno dopo, con l'introduzione dei primi *DBMS*, il data management comincia la sua evoluzione sempre più veloce che conserva come milestone più significativa il progetto di prima *Base di Dati relazionale* della *Relational Software Inc.* (oggi *Oracle Corp.*) [1]. Con l'avvento dei database relazionali e del linguaggio *SQL* (Structured Query Language), negli anni Ottanta, l'analisi dei dati assume una certa dinamicità: l'SQL infatti 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 operazioni di tipo transazionale (inserimento, cancellazione e modifica).
La maggior parte dei sistemi OLTP si contraddistinguono per una limitata possibilità di storicizzazione dei dati. Infatti, l'avvento della rete Internet e della distribuzione su vasta scala dei PC comportano, nel giro di poco, l'insorgenza di nuovi bisogni aziendali.
Aumentano vertiginosamente i contesti in cui sono presenti numerose applicazioni che non condividono la stessa sorgente e i cui dati sono replicati su macchine diverse; non garantendo quindi uniformità tra le informazioni. Per far fronte a questa differenza il primitivo concetto di Database Relazionale si evolve trasversalmente nel moderno *Data Warehouse*. La differenza sostanziale rispetto ai primi si basa sull'assunzione che i database sono progettati in maniera tale da avere un'informazione estremamente strutturata e priva di ridondanze dovuta a più processi di normalizzazione.
I Data Warehouse, al contrario, sono caratterizzati da un tipo di informazione meno strutturata e sintetica, con un notevole livello di dettaglio minore. Inoltre, sono ammesse duplicazioni e ridondanze, tant'è vero che per la loro definizione iniziale si fa ricorso ad un processo di "denormalizzazione" del classico database. Mentre un database relazionale contiene dati relativi ai processi operativi aziendali in una

struttura rigida, fornendo funzionalità semplici come letture, inserimenti, modifiche e cancellazioni, il Data Warehouse è progettato in modo da fornire funzionalità per individuare relazioni tra i dati che possono essere utilizzate per definire delle tecniche strategiche.
L'immensa quantità di dati oggi prodotti e le necessità sempre più emergenti di analizzarli implica, a partire dal nuovo millennio, la nascita di un nuovo fenomeno oggi diffusissimo nel panorama della Business Intelligence, e in generale della Computer Science: i *Big Data*.

1.1 Definizione, principali data source e applicazioni

Con il termine Big Data si intende una grande quantità di dati e informazioni in formati semi strutturati, o addirittura destrutturati, generati ad alta velocità con risorse informative di vasta varietà. I *dataset*, inoltre, richiedono tecniche e tecnologie avanzate di elaborazione delle informazioni per consentirne:

- Cattura.
- Stoccaggio.
- Distribuzione.
- Gestione e analisi.

Il processo analitico avviene, principalmente, in contesti aziendali che sfruttano i risultati ottenuti per l'assistenza alla realtà decisionale che il business richiede sempre più insistentemente. Dalla definizione standard, inoltre, vien facile estrapolare tre concetti cardine:

- Volume
- Varietà
- Velocità

Le tre chiavi di definizione, anche dette comunamente *The Three V's* [2] riescono a schematizzare perfettamente la "struttura" con la quale i Big Data si identificano.

Il **Volume**, naturalmente, si riferisce alla grandezze dei dati. Insita anche nella denominazione, l'enorme quantità di memoria che possono occupare parte dall'ordine di grandezza del *Terabyte* (10^{12} byte) sfociando senza difficoltà nel *Petabyte* (10^{15} byte) e così via. E' riportato, ad esempio, che Facebook riesca ad elaborare fino a un milione di fotografie per secondo e poiché un petabyte equivale a 1024 terabyte, le stime suggeriscono che il social network detiene dataset da più

di 260 miliardi di foto occupando uno spazio di oltre 20 petabyte. Twitter, analogamente, si difende con i suoi 7 terabyte giornalieri [3].
Nel corso degli ultimi anni, con l'avvento del Web 2.0, dei sensori nella ormai nota Internet of Things e la grande diffusione del sistemi gestionali, la crescita è oltremodo esponenziale.
In più, è da considerare che uno dei principi chiave per operare con i Big Data è proprio la memorizzazione di tutti i dati "grezzi", indipendentemente dal loro immediato utilizzo, poiché ogni scarto potrebbe portare all'eliminazione di informazioni utili in futuro. Altro motivo che ne avvantaggia la crescita.

La **Varietà** si riferisce alla eterogeneità strutturale in un insieme di dati. I progressi tecnologici permettono alle imprese di utilizzare vari tipi di dati in forma strutturata, semi strutturata o anche non strutturata.
I *dati strutturati*, che costituiscono solo il 5% di tutti i dati esistenti, sono i dati tabulari più comuni, presenti solitamente in fogli di calcolo o database tradizionali.
Un linguaggio testuale per lo scambio di dati Web come può essere l'*XML* (*eXtensible Markup Language*) è invece, per esempio, un tipico esempio di *dati semi-strutturati*. I documenti XML contengono *tag* definiti dall'utente che rendono il dato *machine-readable*, ossia comprendibile dai calcolatori e quindi facilmente analizzabile.
Testi, immagini, audio e video sono invece esempi di *dati non strutturati*, ossia posseggono un'organizzazione strutturale non compatibile al calcolatore e non utilizzabile immediatamente ai fini di analisi.
Naturalmente è importante far notare che un elevato livello di varietà di dati non è poi una gran novità nel panorama tecnologico-aziendale. Le varie organizzazioni hanno a che fare con dati non strutturati da fonti interne (i.e. sensor data) ed esterne (i.e. social network) da tempo.
L'aspetto innovativo risiede nell'urgenza di definire nuovi sistemi per il *data management* e processi di *analytics* necessari alle organizzazioni per sfruttare i dati nei loro processi di business (i.e. *marketing target strategies*).
La varietà dei dati, dunque, va vista come un'ulteriore vantaggio al processo di "digitalizzazione" per le aziende che, in origine semplici *brick-and-mortar*[1], mostrano la necessità di definire i profili dei loro clienti e tutte le caratteristiche principali favorevoli per i processi di vendita in generale.

[1] (Econ.) riferito alle aziende che non utilizzano la rete Internet per i loro processi organizzativi e/o produttivi.

La **Velocità** è il terzo aspetto da considerare nella definizione. I Big Data, infatti, oltre ad avere elevato volume, vista anche la natura eterogenea delle varie sorgenti, sono generati con una rapidità tale da accendere la sfida tra le aziende nell'analizzare le informazioni con altrettanta rapidità, estrapolare le informazioni utili per il business e identificare un problema, una tendenza o un'opportunità di guadagno. Il tutto necessariamente rientrando, in ogni caso, in un intervallo di tempi di elaborazione minimizzato rispetto alla possibile concorrenza.
E' necessario, a tal fine, possedere tecnologie e logiche applicative in grado permettere il processo in un tempo, approssimativamente, reale.

Sebbene le "three V's" diano un'idea abbastanza chiara di cosa possa rappresentare il concetto di Big Data, IBM, SAS e Oracle [4], massimi esponenti nel panorama applicativo dell'argomento, hanno introdotto rispettivamente tre ulteriori V che, sinteticamente, necessitano di essere menzionate:

- Veridicità.
- Variabilità (e complessità).
- Valore.

La **Veridicità**, concetto abbastanza soggettivo, rappresenta l'inattendibilità di alcune sorgenti di dati. Ad esempio, il *sentiment* dei clienti nei social media sono ovviamente informazioni di natura incerta, in quanto basate su giudizio umano, fondamentalmente facilmente condizionabile.
Eppure, anche queste assumono la loro importanza; tanto da rendere l'analisi di dati imprecisi e incerti un altro aspetto Big Data rivolto allo sviluppo di strumenti e algoritmi per la gestione e l'estrazione di dati per ora, appunto, poco attendibili.

La **Variabilità** (e **complessità**) sono due ulteriori dimensioni di dati di grandi dimensioni. La **variabilità** si riferisce alla variazione dei tassi di flusso di dati. Spesso, la velocità Big Data non è coerente e presenta picchi periodici come anche grandi rallentamenti. La **complessità**, invece, si riferisce al fatto che i Big Data sono generati attraverso una miriade non definita di fonti.
Ciò impone l'esigenza di collegare, "purificare" e trasformare l'input ricevuto da fonti diverse, uniformando il tutto rispetto ad un unico comun denominatore: l'analisi.

Il **Valore**, infine, è un concetto che assume una dimensione prettamente economica. Basandosi sulla definizione di Oracle, i Big Data sono spesso caratterizzati da un "valore a bassa densità". Ciò significa che i dati ricevuti nella forma originale, di solito, hanno un valore basso rispetto al volume in entrata. Questo implica che,

proporzionalmente parlando, un valore elevato può essere ottenuto soltanto analizzando grandissimi volumi di tali dati.

Nonostante il fenomeno dei Big Data abbia suscitato l'interesse di innumerevoli aziende specializzate in sistemi informativi e business intelligence, nonché di numerosi centri di ricerca universitari, la loro definizione rimane comunque molto relativa e poco standardizzata.
E' riportato, a conclusione di questa prima parte introduttiva liberamente ispirata all'articolo in bibliografia, un grafico (figura 1.1) che rappresenta tutto ciò che il fenomeno dei Big Data, ad oggi, "può essere".

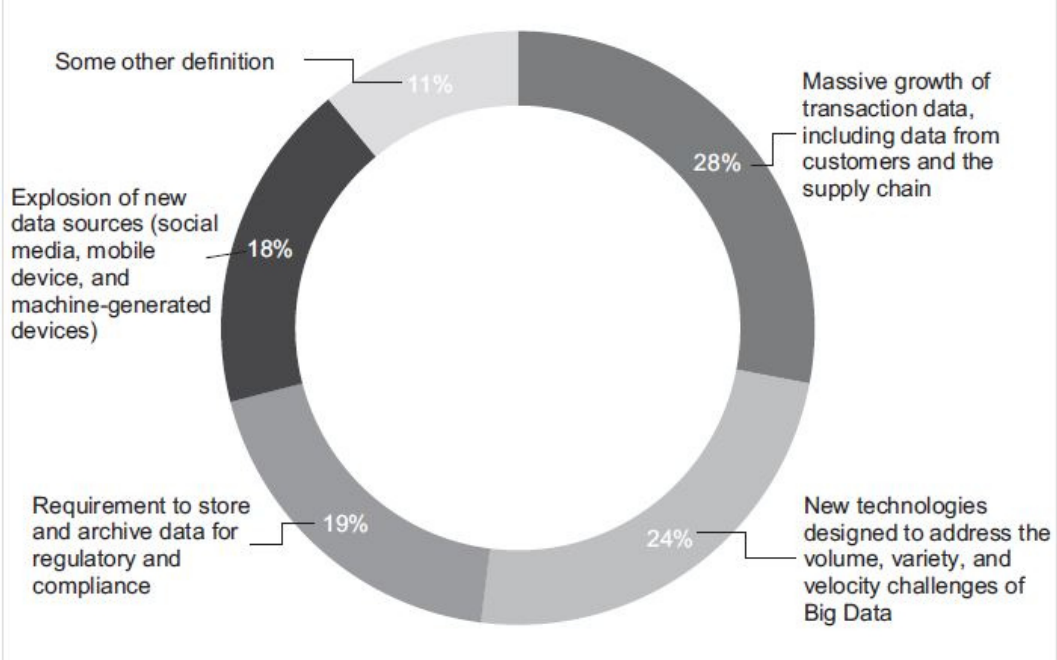


Figura 1.1 - Il grafico riporta tutte le possibili definizioni di Big Data con eventuali data source.

Affiancata alla definizione diviene logico fornire una, seppur breve, carrellata su quelli che attualmente sono i principali usi; non escludendo un'ulteriore evoluzione degli stessi. Come si è detto in precedenza, le necessità base operando con i Big Data si incentrano fondamentalmente sull'analisi di quest'ultimi, con o senza l'effettiva procedura di semplice *storage*.
Il Data Management, infatti, in questa sua particolare eccezione rappresenta solo parte integrante del processo applicativo Big Data.
Il loro "*ciclo di vita*", a tal proposito, parte da un copioso e rapido input con acquisizione iniziale technology-oriented fino al raggiungimento finale con un output decisamente più business-oriented.
In sostanza, l'input risulta inutile se fine a se stesso. Il suo valore potenziale è sbloccato soltanto quando viene effettivamente sfruttato per guidare il processo decisionale aziendale (i.e. azioni di marketing).
Per far ciò, le aziende urgono di elevate capacità di rapida trasformazione di grandi volumi di dati alla ricerca di intuizioni significative a fini di mercato. Il processo generale può essere suddiviso in cinque fasi, illustrate in figura 1.2, che convergono nei due principali sotto-processi: la gestione e l'analisi.
La prima coinvolge i processi e le tecnologie di supporto per acquisire e memorizzare gli elementi poi oggetti di recupero per l'analisi futura.
La seconda, nella parte destra, si riferisce alle tecniche utilizzate per analizzare e acquisire le tanto bramate informazioni utili.
A tal merito, l'analisi dei dati può essere perciò vista come un sotto-processo nel processo globale di "*monetizzazione*" dei Big Data.

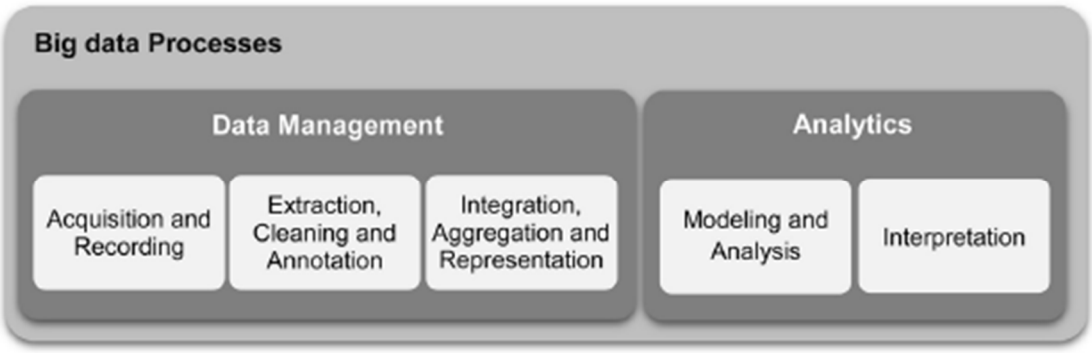


Figura 1.2 - Dualità tra il semplice Data Management e il post processo di Analytics.

Sempre dalla figura 1.2, in più, si evince l'unione concettuale ma non applicativa tra il tradizionale Data Management e un più avanzato stadio di analisi indipendente dotato di sue metriche, modelli e campi di azione.
Data la sua copiosità, un elenco esaustivo delle tecniche renderebbe il documento troppo lungo.

Ad approfondimento di quanto appena presentato, quindi, si riporta di seguito solo una possibile, ma non unica, divisione delle principali "branche" che l'analisi dei Big Data offre ad oggi:

- Multimedia Analytics.
- Social Media Analytics.
- Predictive Analytics.

La sorgente dei dati multimediali è facilmente intuibile; si tratta principalmente di analisi di documenti testuali, tracce audio, video e immagini,
Tutte posseggono stampo prettamente statistico. Il *text mining,* che avviene tramite logiche di *machine learning* e algoritmi di cosiddetta *Information extraction* (*IE),* consente alle aziende di convertire grandi volumi di testo *human-generated* (dati non strutturati) in una sintesi significativa che supporta il processo decisionale. Per esempio, analisi del testo può essere utilizzata per prevedere il mercato azionario sulla base delle informazioni estratte da notizie finanziarie.
Analogamente l'analisi audio è finalizzata all'estrazione delle informazioni dai dati audio naturalmente non strutturati. Un esempio può essere lo sp*eech mining,* utilizzato attualmente soprattutto dai call center per l'assistenza ai clienti. Complesse tecniche di analisi vengono lanciate su migliaia o addirittura milioni di ore di chiamate registrate (*LVCSR* system[2]).
Queste tecniche consentono di migliorare la *customer experience* di cui oggi si sente parlare molto, al fine di valutare l'agente di vendita/assistenza e le sue performance, aumentare le vendite e monitorare il comportamento dei propri clienti identificando eventuali problemi di prodotti o servizi.
Per i video, o le immagini, il discorso è analogo. Il processo Big Data permette di campionare grandi quantità di streaming di dati, di cui si parlerà in seguito in quanto argomento principale della tesi, provenienti da canali televisivi o online.
In più, con l'avvento dei social network "*photo-based*" (e.g. Instagram) risulta possibile comprendere i nuovi trend, la *social influence* di un particolare personaggio valutandone l'influenza e le connessioni in una rete sociale; o anche effettuare azioni di *community detection*, identificando quali sono le comunità che si formano nel corso del tempo, i loro interessi e in che percentuale si presentano. Così da indirizzare campagne di vendita mirate, pubblicità, etc.

A conclusione di questa carrellata sui principali ambiti che il Big Data process assume: la *Predictive Analytics*.

[2] Conosciuto come "large-vocabulary continuous speech recognition", è un algoritmo per il riconoscimento vocale automatico (ASR) che converte suoni in parole identificate sulla base di un dizionario predefinito.

Quest'ultima comprende una varietà di tecniche finalizzate a "prevedere", come è intuibile dalla denominazione, i risultati futuri sulla base di dati storici e attuali.
Nella pratica, il concetto può essere applicato a quasi tutte le discipline; dal prevedere la rottura di una lavatrice in base al flusso di dati provenienti dalle diverse migliaia di sensori (*prognostic maintenance*) alla previsione delle prossime mosse di acquisto dei clienti in base a quello che comprano (*suggestion system*[3]).
Al suo interno, le tecniche di analisi predittiva sono suddivise in due gruppi ben divisi a seconda dell'origine che il campione dei dati possiede.
Basandosi su vecchi modelli, dati storici certi e/o survey si cercano di scoprire nuovi approcci alle situazioni aziendali e non, prevedendo quindi errori e cali. Una sorta di "contabilità direzionale[4]" basandosi però su moli di dati semi strutturati.
Al contrario se l'origine dei dati appare completamente illogica, si cercano di scoprire nuovi pattern e relazioni che hanno alta probabilità di ripresentarsi in futuro. Muovendoci in ambito statistico è necessario sempre considerare un *margine di errore* che i nuovi modelli di analisi cercano sempre più di assottigliare.

1.2 Tecnologie di storage non convenzionali: NoSQL

Rispetto a quanto detto in precedenza, in relazione alla caratteristiche principali che individuano la presenza di Big Data (*The three V's)* emerge facilmente che la sfida di questa immensa quantità di dati abbia come punto d'inizio proprio lo *storage* e il come organizzare migliaia di informazioni di natura prettamente eterogenea senza provocare, in contraccolpo, un abbassamento delle prestazioni nell'intero processo.
Un aumento così vertiginoso, infatti, ha da subito presentato un "deficit" nella capacità di storage che le tradizionali Basi di Dati relazionali offrono.
A tal proposito, per far fronte a questa esigenza contesti aziendali ricorrono alle recentissime soluzioni **NoSQL.**
Il movimento NoSQL, da intendere non necessariamente come un ripudio categorico alla tradizionale Basi di Dati relazionale ma come semplice alternativa causata da una serie di scope-changes (*Not-Only* SQL), ha come obiettivi:

- Garantire velocità di esecuzione anche nell'elaborazione di numerosi terabyte di dati.
- Permettere la scalabilità orizzontale (i.e. eventuale aggiunta di nuovi server).

[3] Anche detti sistemi di *recommendation,* sono meccanismi utilizzati soprattutto nell'ambito dei noti e-commerce per permettere un processo di "guida agli acquisti" sulla base degli interessi personali dei clienti.
[4] (Econ.) processo proprio di aziende medio-grandi finalizzato all'analisi dello storico contabile per future decisioni interne e previsioni di mercato esterne.

- Fornire un elevato livello di *availability* ma, specialmente, la possibilità di ricevere dati non strutturati *senza dipendere da uno schema fisso e predefinito*.

I database NoSql, infatti, vengono anche detti *schemaless database*. La mancanza di schema, che negli RDBMS viene sviluppato a fronte di un processo di modellazione concettuale e successivamente logica, consente ai database NoSql di adattarsi alla varietà dei dati. Diversamente dai sistemi tradizionali, la tipologia e la quantità di fonti diverse da cui possono provenire i dati è perciò molteplice (*sensori, WWW, social media, etc*). A conseguenza di queste premesse la struttura NoSQL-based si presenta concettualmente molto diversa.
I dati sono conservati in *documenti*, non in tabelle, per cui la prima differenza sta nel fatto che le informazioni non sono distribuite in differenti strutture logiche, ma vengono aggregate per oggetto in documenti inizialmente soltanto di tipo *Key-Value* e poi anche *Document Store* basati su semantica *JSON*[5].
Ogni documento aggregato raccoglie tutti i dati associati a un'entità, in modo che qualsiasi applicazione possa trattare l'entità come oggetto e valutare tutte le informazioni a questa collegate.
Così facendo ogni singolo documento è *indipendente* dall'intera Basi di Dati, in quanto tutte le informazioni sono già raccolte in esso, permettendo l'introduzione di nuovi "tipi di dato" in ogni momento.
A questo punto viene spontaneo far notare la grande possibilità di sfociare nella *duplicazione delle informazioni*, considerata impensabile nel modello relazionale.
In realtà, visti i costi sempre più accessibili di questi nuovi sistemi di storage, spesso anche open-source, rendono questo inconveniente poco importante.
Poiché il nuovo paradigma di stoccaggio sta prendendo piede molto velocemente le tipologie di NoSQL Database risultano già essere numerose. Di seguito solo le più diffuse.

[5] JSON (JavaScript Object Notation) è un formato di testo completamente indipendente dal linguaggio di programmazione, ma utilizza convenzioni conosciute dai programmatori di linguaggi C, Java, Python e molti altri. Questa caratteristica fa di JSON un linguaggio ideale per lo scambio di dati.
JSON è basato su due strutture:

- Un insieme di coppie nome/valore. In diversi linguaggi, questo è realizzato come un oggetto, un record, un dizionario, una tabella hash, un elenco di chiavi o un array associativo.
- Un elenco ordinato di valori. Nella maggior parte dei linguaggi questo si realizza con un array, un vettore, un elenco o una sequenza.

Sono strutture di dati universali. Virtualmente tutti i linguaggi di programmazione moderni li supportano in entrambe le forme. E' ormai convenzionale che un formato di dato che sia interscambiabile tra linguaggi di programmazione debba essere basato su queste strutture. [5]

Key/value store

I Key/Value store rappresentano una tipologia di database NoSql che si basa sul concetto di *associative array*, implementati attraverso *Hash Map*: 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 transazionali di *add, remove, modify e find.*

I prodotti più famosi sono *Apache Cassandra, Berkley DB e Redis.*

Column-oriented

I column-oriented database, diversamente dai tradizionali RDBMS che memorizzano i dati per riga, sfruttano la memorizzazione dei dati per colonna

Appartengono a questo tipo *BigTable* di Google e *SimpleDB* di Amazon;

Document-oriented

I database document-oriented rappresentano una specializzazione dei key/value store. E' un insieme strutturato di coppie chiave/valore, spesso organizzati in formato JSON o XML che non pone vincoli allo schema dei documenti.

Garantiscono una grande flessibilità in situazioni in cui, per loro natura, i dati hanno una struttura variabile. I documenti sono gestiti nel loro insieme, evitando di suddividere tali documenti in base alla struttura delle coppie chiave/valore. Questa tipologia di database è particolarmente adatta alla memorizzazione di tipologie di dati complessi ed eterogenei.

I più noti e utilizzati document database sono MongoDB e CouchDB.

Graph

I graph database rappresentano una particolare categoria di database NoSql, in cui le "relazioni" vengono rappresentate come *grafi.*

Analogamente alla definizione matematica (insieme di nodi collegati tra loro da archi tale che $G = (V, E)$[6]) in ambito informatico il grafo rappresenta una struttura dati costituita da un insieme finito di coppie ordinate di oggetti.

Le struttura a grafo si prestano molto bene per la gestione di determinati dati non strutturati ma fortemente interconnessi (i.e. Social Network data).

Esempi sono *GraphDB Neo4J, Sones e InfinityGraph.*

[6] Dato un grafo G = (V, E) due vertici v, u ∈ V si dicono "connessi" se esiste un cammino con estremi v e u. Se tale cammino non esiste, v e u sono detti "sconnessi".

A conclusione del paragrafo è possibile quindi definire molto forte la relazione tra il Big Data process e il paradigma NoSQL.
Andando avanti la correlazione sembra essere un continua crescita tanto da portare sempre più aziende, tra cui anche *software house* organizzate con modello di sviluppo *Agile,* alla decisione di cominciare a farne uso.
Difatti, numerosi casi studio hanno chiaramente dimostrato i benefici di adozione NoSQL [6]. Una migliore comprensione della necessità del lavoro di squadra in termini di *Data Governance* nello sviluppo, può essere una conseguenza.
Naturalmente la creazione di strumenti che forniscono funzionalità di *SQL-like* all'interno di piattaforme NoSQL rende la piattaforma più facilmente assimilabile.
NoSQL, Big Data, l'Internet of Things sono un assortimento sempre crescente di nuove tendenze e tecnologie destinate a rivoluzionare il panorama della Computer Science (figura 1.3) [7]. Non sono più visti, difatti, come antagonisti dei sistemi relazionali ma come sistemi con speciali capacità uniche e soprattutto differenti. Esiste una convivenza pacifica in via di sviluppo, sebbene i sistemi relazionali siano ancora importanti poiché forniscono alcune operazioni che continueranno ad essere necessarie per il futuro del data management [8].

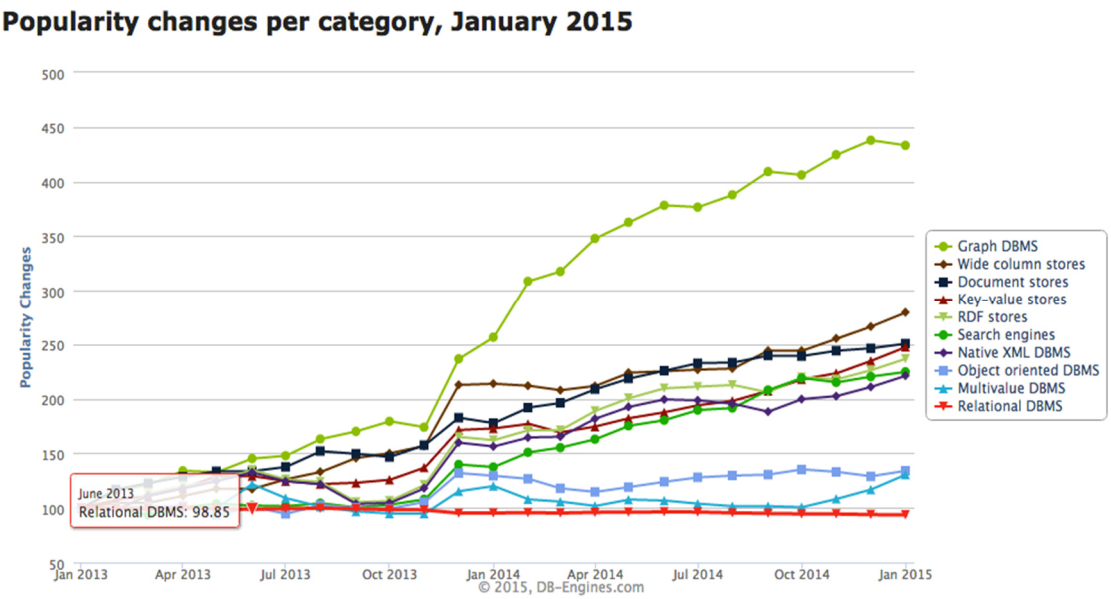


Figura 1.3 - Sviluppo delle tecnologie NoSQL tra il 2013 e il 2015 rispetto ai RDBMS.

1.3 Piattaforma di sviluppo: Apache Hadoop

Finora è stato affrontato un discorso generale sul fenomeno dei Big Data evidenziando caratteristiche, importanti applicazioni e tecnologie di storage dove è emersa la necessità di lavorare con database sempre più flessibili, ma soprattutto scalabili.
Naturalmente, le tecnologie tradizionali applicate al Big Data process mettono in luce una importante complicazione che, per la loro natura rigida e predefinita, non riescono a risolvere: la gestione e il calcolo di grandi moli di dati in tempo, approssimativamente, *reale* in presenza di dati non strutturati e spesso incomprensibili di primo acchito.
Tale esigenza ha portato allo sviluppo di nuovi modelli di gestione dei dati ma anche a nuove logiche architetturali necessarie per l'elaborazione di quest'ultimi in modo efficiente e *scalabile*.
A tal proposito, la più diffusa è la piattaforma **Apache™ Hadoop®** [9].
Hadoop è il più conosciuto framework *open-source* finalizzato alla memorizzazione e la gestione *distribuita* di grandi quantità di dati mediante cluster di macchine solitamente *commodity hardware*[7] che fungono da nodi di elaborazione.
La piattaforma presenta, a livello logico, una struttura prettamente modulare con alto fattore di *fault-tolerance*.
Tutti i moduli, infatti, sono progettati in modo tale che il software del framework gestisca automaticamente gli eventuali eventi di "down" dell'hardware, molto più frequenti in un sistema parallelo.
La struttura minimizzata presenta un "core" di memorizzazione noto come HDFS e un componente di elaborazione chiamato *MapReduce* (figura 1.4).
Il processo inizia con la divisione dei file in *blocks* (128Mb di default) distribuiti lungo i nodi del centro di calcolo (*cluster*).
Successivamente, per processare i dati, MapReduce procede con il *trasferimento del codice* nei nodi per poter processare in parallelo e con tempi ridotti (*Streaming Data Access*), sulla base dei dati che ogni nodo avrà il compito di elaborare.

[7] Commodity computing, o commodity cluster computing, è l'uso di un gran numero di già disponibili componenti (i.e. PC) per il calcolo parallelo, per ottenere una migliore quantità di calcolo ad un basso costo.

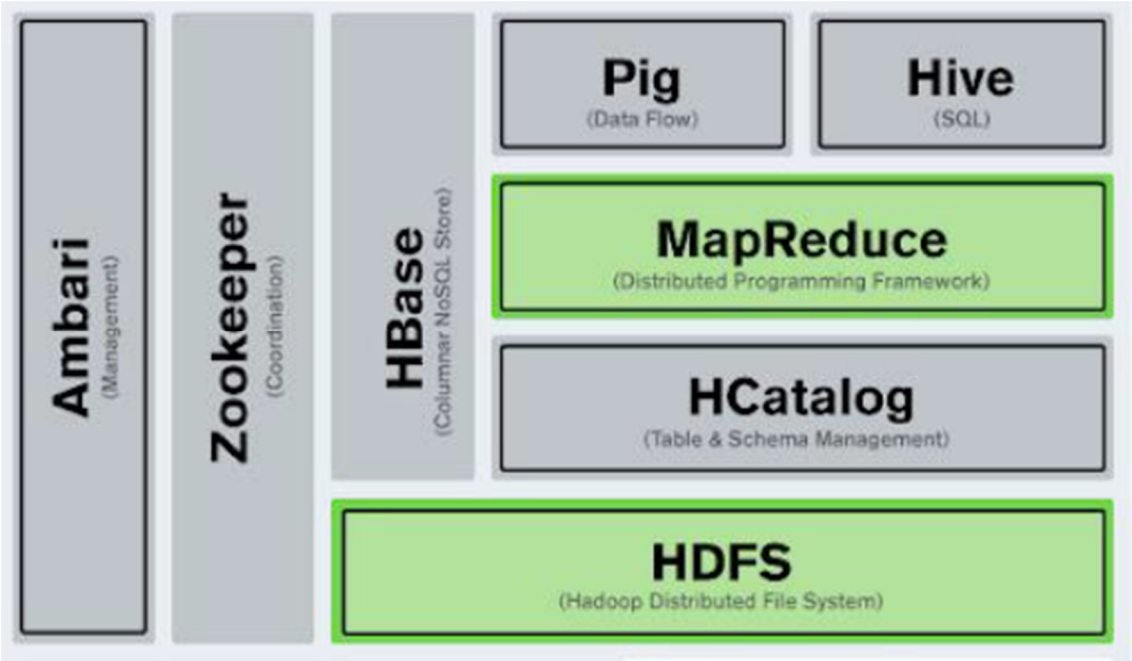


Figura 1.4 – In verde sono evidenziati i componenti del core dell'architettura Hadoop.

L'architettura base di Hadoop, si compone dei seguenti elementi:

- HDFS.
- Hadoop Common.
- MapReduce.
- Yarn.

HDFS (*Hadoop Distributed File System*), come riportato nella documentazione ufficiale e già accennato prima, è il filesystem distribuito di Hadoop, progettato appositamente per essere eseguito su commodity hardware.
Quando la mole di dati va oltre i canoni classici di stoccaggio di una singola macchina, diventa appunto necessario partizionare gli stessi su un certo numero di macchine separate e interconnesse da una rete, rendendo il filesystem distribuito più complesso rispetto ai tradizionali.

Hadoop Common rappresenta il livello software comune a tutto lo *stack* che fornisce le funzioni di supporto agli altri moduli.

MapReduce, in assoluto, è il responsabile del processo di calcolo. Operando secondo il principio *"divide et impera"* riesce a suddividere un problema complesso, assieme ai relativi dati, in piccole parti processate in modo autonomo procedendo, una volta concluse tutte le sotto-elaborazioni, al *merge* dei risultati parziali producendo *un unico risultato finale*.

Yarn, a tal proposito, si occupa della schedulazione dei task, ossia delle sotto-attività che compongono le procedure *map* e *reduce* eseguite nel processo di calcolo.

Si evince quindi che HDFS e MapReduce rappresentano il cuore del framework Hadoop. Affinché la computazione possa essere portata a termine, i due componenti devono collaborare fra loro.
In continuazione a queste premesse si delinea un sistema *altamente affidabile, scalabile* (i.e. aggiunta nodi al cluster) ma soprattutto *efficiente*.
E' logico adesso percorrere quelle che sono le differenziazioni interne relative ai nodi, elementi cardine dell'intera architettura, e alle procedure (figura 1.5).

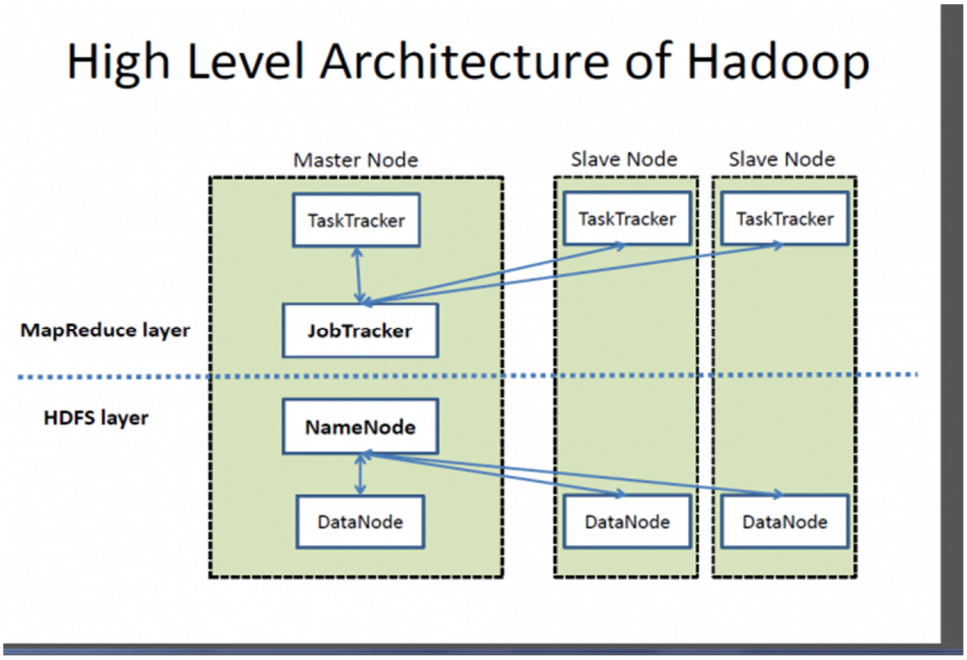


Figura 1.5 - Design e i vari tipi di nodo della piattaforma Hadoop.

In un cluster Hadoop, come è possibile vedere in figura 1.5, non tutti i nodi sono uguali, ma si dividono in due tipologie [10].
Il nodo *master*, che esegue i processi di coordinamento di HDFS e MapReduce e, analogamente, il nodo *slave*, utilizzato per la memorizzazione dei risultati e per l'esecuzione delle operazioni di calcolo.
Schematicamente la struttura è organizzata secondo i seguenti elementi:

- Il **Job Tracker** (master-side) è il servizio che prende in input il job da eseguire e lo suddivide tra i nodi *map* (*task tracker*), e i nodi *reduce* (*task tracker*), i quali svolgono le operazioni. La scelta avviene o tra i nodi più vicini o tra quelli che contengono i dati richiesti. Una volta che i task tracker hanno svolto il compito, il job tracker fornirà il risultato.

- I **Task Tracker** (*map*) sono i nodi scelti per l'operazione di map. Questi nodi si occupano di raccogliere i dati e di generare una *coppia chiave – valore* che poi verrà inviata ai task reduce.

- I **Task Tracker** *(reduce)* sono i nodi che ricevono le coppie chiave-valore e, dopo averle ordinate per chiave, le spediscono allo script di reducing che genera il risultato.

- Il **NameNode** (master-side) mantiene la struttura della directory di tutti i file presenti nel file system e tiene traccia della macchina in cui essi sono memorizzati. E' importante inoltre tenere in considerazione che questo memorizza **solo le strutture** e non i dati veri e propri.
  Le applicazioni client possono dialogare con il NameNode quando vogliono informazioni per individuare un file, o quando vogliono aggiungere/copiare /spostare/cancellare un file. Il NameNode risponde alle richieste restituendo un elenco di server DataNode dove si trovano memorizzati i dati richiesti. Quando il NameNode cessa la sua esecuzione ne risente l'intero file system. Per ovviare a questo inconveniente è presente un SecondaryNameNode *opzionale* che può essere ospitato su una macchina separata, il quale tuttavia non fornisce alcuna ridondanza reale.

- **DataNode** (*slave*): memorizza i dati nel file system che per essere funzionale ha solitamente più DataNode che contengono dati replicati.
  Le *applicazioni client* possono comunicare direttamente con un DataNode, una volta che il NameNode ha fornito la posizione dei dati. Allo stesso modo, le operazioni di MapReduce affidate alle istanze TaskTracker nei pressi di un DataNode, possono parlare direttamente ai DataNode per accedere ai

file. Istanze TaskTracker possono, anzi devono, essere "schierate" sugli stessi server delle istanze DataNode, in modo che le operazioni di MapReduce vengano eseguite vicino ai dati. Le istanze DataNode possono comunicare tra loro, specialmente quando si tratta di dati replicati.

A conclusione di questo terzo paragrafo si riporta un semplice schema su quanto finora spiegato riguardo l'architettura innovativa di Hadoop (figura 1.6).

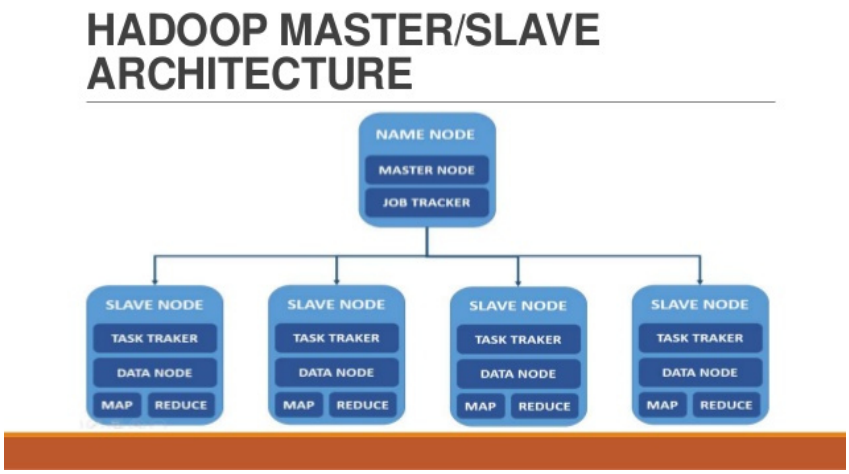


Figura 1.6 - Architettura Hadoop e relazione tra diversi tipi di nodo.

Ovviamente quella descritta, come già fatto notare, è soltanto l'architettura "entry-level" della piattaforma, dove è possibile, tuttavia, aggiungere tutti i componenti supplementari che fanno parte dello *stack* di Hadoop, innestati direttamente sulla parte core (e.g. Apache Pig, Apache Hive, Apache Hbase, Apache Spark, ***Apache Storm***). Per questo progetto di tesi è stato definito come caso di studio l'ultimo elencato. Un modulo di **Data stream processing**, per l'analisi di flussi di dati *real-time.*

2. Analisi di Data stream

Questo secondo capitolo, diversamente dal primo in cui si forniva una panoramica generale sull'argomento, è incentrato su una particolare accezione che il fenomeno Big Data assume: l'analisi e la gestione di *data stream.*
Innanzitutto è necessario inquadrare quello che può significare il concetto. Analizzare data stream rappresenta quella particola pratica che si occupa di elaborare flussi continui di dati, raccolti da sorgenti varie (e.g. sensori, web, social network) *geograficamente distribuite*, prodotti con *data rate* non predicibili, al fine di ottenere risposte in modo a sua volta tempestivo senza la necessaria storicizzazione [11].
Volendo fornire un esempio, la procedura in questione ha riscontro in soluzioni derivanti da applicazioni/esigenze anche nettamente diverse tra loro (e.g. mercato finanziario, rilevamenti bancari di frodi, **cyber attacchi a reti informatiche**). Naturalmente, la continuità che i flussi di dati assumono lascia intendere che in intervalli di tempo notevolmente ridotti la quantità di dati da analizzare cresca esponenzialmente. E' inutile dire che una quantità di dati così vasta, routine se si parla di Big Data, aggiunta ad una velocità di trasmissione e trasferimento altrettanto alta, rende impossibile la memorizzazione completa dei dati, tantomeno se le tecniche di storage risultano essere i tradizionali DBMS.
Una soluzione relazionale classica permetterebbe di processare dati solo su richiesta, producendo ritardi, significative latenze e inevitabile perdita di frazioni di dataset. Occorre, perciò, far ricorso a nuove tecnologie e strumenti consoni alle esigenze nell'immediato specificate. A tal merito, una prima possibile alternativa è presentata dall'evoluzione "customizzata" dei classici DBMS: il DSMS (*data stream management system*). Questa particolare soluzione fornisce una possibilità per gestire data stream in quanto presenta:

- Esecuzione continua delle query (*continuous query*, o anche *CQ*).
- Interrogazioni con sintassi SQL-like
- Elaborazione di flussi ma *non memorizzati.*

Di seguito, si elencano due ulteriori soluzioni esistenti che rendono possibile l'analisi di flussi secondo le regole prima definite:

DSP (*data stream processing*): è il modello che deriva dalla generalizzazione del DSMS (dati non persistenti, continuous query)

Permette l'elaborazione di stream di dati provenienti da sorgenti differenti, producendo, a sua volta, nuovi stream in uscita dove è possibile riapplicare il procedimento al fine di stratificare concettualmente l'analisi.

CEP (*complex event processing*): si sviluppa in parallelo al DSMS, come evoluzione del modello *publish-subscribe*[8] procede con l'elaborazione di notifiche di eventi e non dati generici provenienti da sorgenti diverse avvantaggiando l'identificazione di *pattern di eventi* (o eventi complessi) di interesse.

Poiché il progetto di tesi prende come caso di studio *Apache Storm,* e poiché quest'ultimo rappresenta uno dei framework più utilizzati per la prima tipologia di analisi (*DSP*), viene fornita, nelle prossime pagine, una overview focalizzata sul modello evidenziandone caratteristiche sommarie e, a completamento del discorso, le differenze più significative con gli altri sopra elencati.

Come prima presentazione, in figura 2.1, una sottodivisione concettuale propria dei sistemi di data stream processing.

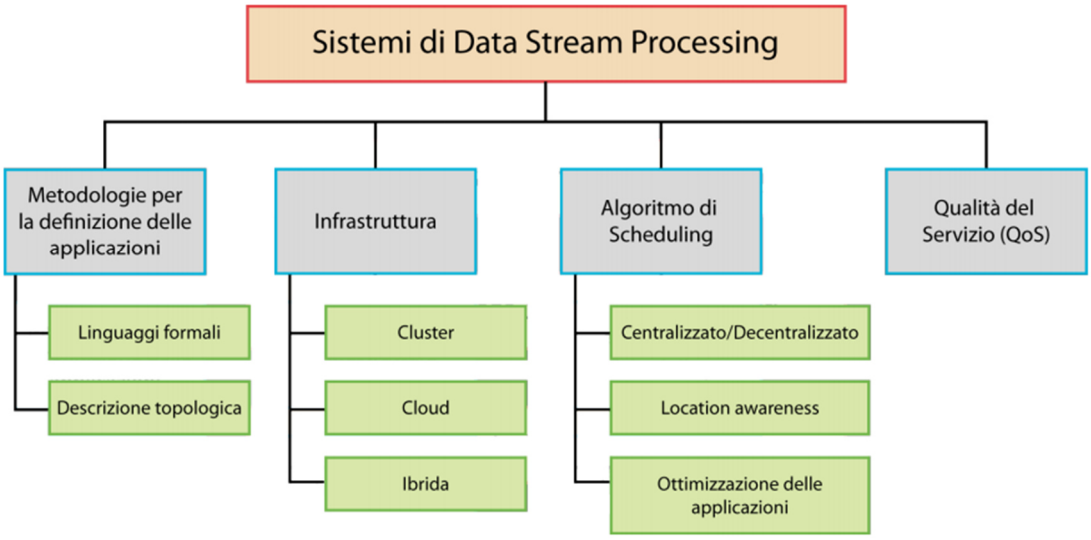


Figura 2.1 - Tutti gli aspetti di un sistema DSP canonico con le relative diverse alternative.

[8] Il modello ad oggi più utilizzato per la comunicazione asincrona ad eventi. Qualsiasi attore che effettua una subscribe ad un particolare "topic" (ID specifico per un canale di comunicazione) riceve gli eventi da chi, analogamente, pubblica un qualsivoglia contenuto con quell'ID.

Lo schema ha come primo elemento la *definizione delle applicazioni* che può avvenire mediante due diverse metodologie:

- I **Linguaggi formali**, dotati di maggiore rigorosità ma alta espressività possono essere *dichiarativi* o *imperativi,* e specificano la composizione degli operatori primitivi o il risultato vero e proprio (*SQL-like*).

- La **descrizione topologica**, utilizzata anche da Apache Storm, permette invece maggiore flessibilità. E' possibile, difatti, procedere ad una definizione esplicita dei componenti (*built-in o definiti dall'utente*) e delle interconnessioni per mezzo di un grafo diretto aciclico, chiamato, appunto, *topologia.* Vista la sua natura una topologia è ovviamente composta da *nodi* (operatori dell'applicazione) e *archi* (data stream scambiati tra gli operatori).
  Un esempio, anche utilizzato nel progetto di tesi come "primo approccio" ai sistemi DSP, può essere il *Word Counter*, in figura 2.2. In modo simile vengono individuati i trend su Twitter.

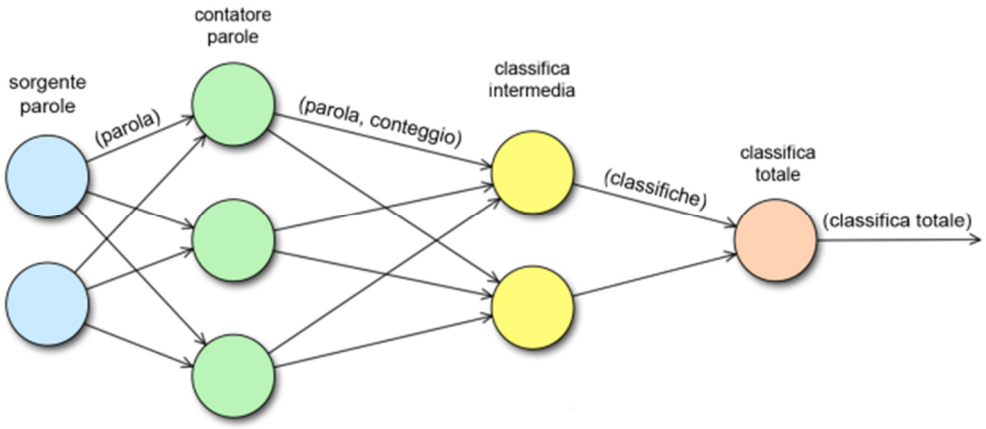


Figura 2.2 - Un particolare esempio di topologia DSP per il conteggio e classifica delle occorrenze di ogni parola.

Il secondo aspetto, invece, è *l'infrastruttura.* Dove, per l'appunto, risiedono le risorse computazionali. Le scelte possibili questa volta sono tre:

- Far uso di un **cluster dedicato** con un numero di nodi omogenei, "vicini" ed in numero staticamente definito.

- Dipendere totalmente dai recenti Cloud che permettono l'allocazione dinamica con conseguente migliore assorbimento delle fluttuazioni versatili

nell'arrivo dei dati. Avendo inoltre, nodi *geograficamente distribuiti* diviene nuovo interesse per il DSP, ma anche scatenatore di nuove problematiche riguardo latenza tra i nodi, scala etc.

- Mediare tra le due precedenti con delle soluzioni pressoché *ibride* considerando un insieme *statico* di nodi, estendibili con risorse *on-demand* nel cloud pur valutando bilanciamento del carico, *overheads* e costi generali.

Ad ogni modo, la prima soluzione è generalmente la scelta tradizionale perché più comoda, accessibile e soprattutto facilmente gestibile in termini di organizzazione delle risorse di calcolo. Per il progetto di tesi si è optato per la medesima soluzione, viste la possibilità di accesso all'utilizzo di un cluster, più avanti descritto.

Ogni sistema DSP, proseguendo, possiede uno **Scheduler**, componente fondamentale responsabile dell'assegnamento degli operatori delle applicazioni da eseguire alle risorse computazionali a disposizione. La sua efficienza è un aspetto critico che influenza fortemente le performance del sistema e delle applicazioni eseguite.
Un algoritmo di scheduling predefinito è sviluppato in relazione a diversi fattori ben individuati:

- La scelta ponderata tra un *algoritmo centralizzato* o *algoritmo distribuito* che portano ad un'obbligatoria conoscenza intera rete valutando eventuali problemi di scalabilità.
- La *valutazione delle metriche* da ottimizzare circa latenza, utilizzo della rete, importanza degli operatori e risorse.
- La *capacità adattativa* che una particolare architettura può offrire.
- Ove necessario, la *capacità di ottimizzare* il grafo applicativo generalmente con definizione di applicazioni con linguaggi formali (e.g., *merging*, *splitting* e load shedding)

Terminando, un buon sistema DSP garantisce una elevata ***QoS*** (*Quality of Service)* passando dalla gestione ottimizzata degli stream ad un elevato tasso di *fault-tolerance* nelle applicazioni.
Ne segue che ogni applicazione di tale stampo presenta quindi le seguenti caratteristiche di base:

- Le sorgenti sono emittenti di un flusso continuo (stream) dei dati.
- Lo stream **non** viene memorizzato in nessun dataset statico.

- Lo stream viene riversato su un insieme di operatori.
- Gli operatori sono progettati per lavorare in parallelo.
- Gli operatori interagiscono *solo per mezzo degli stream.*
- Gli operatori eseguono delle funzioni ben precise il cui risultato dipende solitamente *solo dal flusso in ingresso.*
- Gli operatori possono, a loro volta, *generare un nuovo stream.*

Per processare grossi flussi di dati, inoltre, è utile aumentare il grado di parallelismo con processi di:

- Pipeline: istruzioni complesse suddivise in una sequenza di passi.
- Task parallelism: stesso dato utilizzato da diversi operatori (successori nel grafo) in parallelo.
- Data parallelism: diversi dati appartenenti allo stesso flusso sono processati da diverse repliche dello stesso operatore.

O anche facendo ricorso alle tecniche di ottimizzazione più diffuse.
Lo *Sharding*, ad esempio, è un partizionamento orizzontale in cui i dati con caratteristiche simili sono indirizzati sempre alla stessa replica.
Quest'ultima, è determinata tramite soluzioni *hash-based* calcolate su sottoinsieme di attributi dei dati stessi. I batch vengono poi assemblati nuovamente alla fine del processo (*divide et impera*) con un'operazione di *merging*, figura 2.3.

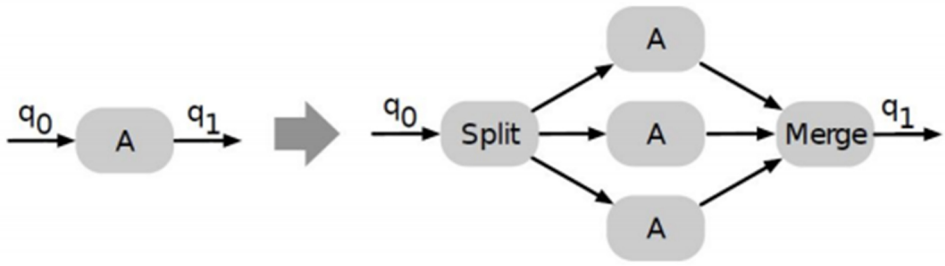


Figura 2.3 - Le differenze tra un approccio semplice di analisi a sinistra e, a destra, un'ottimizzazione mediante sharding.

Il *Load shedding* dove, sacrificando l'accuratezza dei risultati (random, secondo una probabilità o *priority-based*) in caso di sovraccarico del sistema si assume che sia comunque possibile continuare a regime ridotto l'elaborazione dello stream. Ovviamente ridurre il carico vicino alla sorgente fa sprecare meno lavoro, ma penalizza un numero maggiore di applicazioni). Schematizzazione in figura 2.4.

Figura 2.4 - Funzionamento generale di un processo di shedding.

A seguire, come precedentemente anticipato, si fornisce un approfondimento sulle differenze sostanziali del DSP rispetto al CEP.
Anzitutto, il tipo di dato. Se nel CEP questo non è identificato da altro se non come semplice "notifica di eventi" nel DSP teoricamente è possibile ricevere e ritrasmettere qualsiasi tipo di dato (primitivi, XML, **JSON**). Inoltre, se nel CEP il tempo di *ordinamento* dei dati risulta fondamentale nel DSP è praticamente superfluo vista la natura eterogenea dell'input. Il CEP, in conclusione, nel panorama applicativo è caratterizzato da linguaggi *pattern-based,* necessari per definire le azioni da intraprendere, le cosiddette *"*if this then that*"* in relazione a ripresentazioni periodiche di particolari input conosciuti.

In aggiunta, poiché l'analisi dati nasce con Hadoop e il batching, si riportano le differenze del DSP con la nota piattaforma e il suo paradigma Map-Reduce; per comprenderne al meglio l'evoluzione logica.
Sebbene diverse estensioni consentano di usare Hadoop per interrogare il dataset con un approccio SQL-like (Apache Hive) o con linguaggio procedurale (Apache Pig) la differenza principale tocca due aspetti ben precisi:

- Persistenza: Hadoop necessita della *memorizzazione* dei dati
- Batching: anche negli approcci per il *CQ* (*Hadoop Online*) si lavora considerando piccoli batch successivi da analizzare; questo introduce un ritardo proporzionale alla *dimensione del batch*.

In sostanza, sia il CEP che il Map-Reduce in generale, sono comunque associati ad esigenze di analisi e contesti applicativi nettamente differenti tra loro.
Oggigiorno realtà aziendali che fanno ricorso al Data Stream Processing sono ormai diverse. Se ne considerano due e i relativi *framework* utilizzati per tale processo su larga scala:

Amazon.com Inc-> *Amazon Kinesis.*
Twitter Inc -> ***Apache Storm***.

Il secondo elencato (Apache Storm) è stato assunto come strumento di lavoro nel progetto di tesi.

3. Apache Storm

Figura 3.1 - Logo di Apache Storm

Inizialmente pensato e prototipato da *Nathan Marz*, è entrato a far parte dell'ecosistema *Apache* dal Dicembre 2013, inserito all'interno dell'Apache *Incubator*. Diventa successivamente uno dei progetti principali nel Novembre 2015 con la sua ultima release Apache Storm 0.10.0 [12].
Storm è un sistema distribuito, affidabile e fault-tolerant utilizzato nella maggior parte dei casi per il data stream processing. Il lavoro è delegato a diversi tipi di componenti che sono ciascuno responsabile di uno specifico compito. Data la sua natura di sistema distribuito, il framework viene installato solitamente su piattaforme Cluster, dove ogni nodo è *fisicamente*, e successivamente anche *logicamente*, definito secondo il ruolo che occupa all'interno del sistema distribuito.

3.1 Struttura fisica e componenti

La struttura *fisica* (figura 3.2) [13], anzitutto, è organizzata secondo la logica standard dei sistemi distribuiti; è, infatti, presente un master node e un numero definito di nodi worker.
Il nodo master esegue senza termine un demone chiamato **Nimbus**, che è responsabile della distribuzione di codice in tutto il cluster, dell'assegnazione dei vari task a ciascun nodo worker e del monitoraggio di eventuali guasti.
I vari nodi worker, a loro volta, eseguono un demone denominato ***Supervisor***, che esegue una porzione di codice (*task*) dell'intera applicazione in esecuzione.
Storm, inoltre, per il management generale dei vari worker, che possono arrivare anche a decine di centinaia, fa ricorso ad un servizio distribuito per il coordinamento di applicazioni a loro volta distribuite: lo **ZooKeeper**. Quest'ultimo fornisce delle interfacce applicative che permettono di implementare rapidamente servizi di più alto livello per l'organizzazione delle sincronizzazioni, lo storage delle statistiche e di eventuali errori di esecuzione, o più in generale, delle procedure di *naming*. Il tutto completamente in automatico all'avvio del sistema.

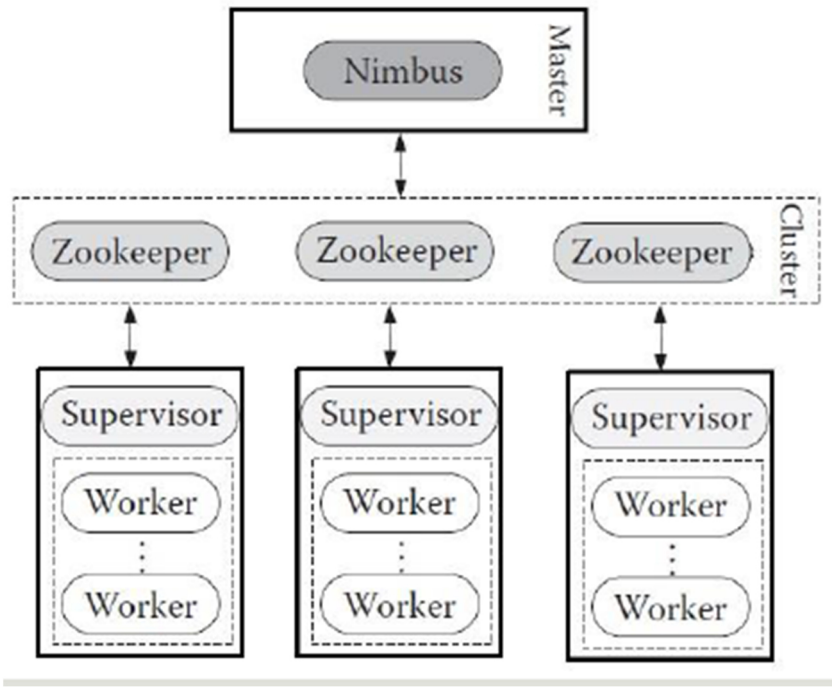


Figura 3.2 - I componenti di un Cluster Storm

E' possibile carpire dall'immagine sopra riportata, in aggiunta, che ogni supervisor, vista la sua particolare predisposizione al calcolo parallelo, ha la possibilità di gestire un maggior numero di worker; che quindi possono risiedere anche sulla stessa macchina. Un applicazione Storm, insomma, si identifica come nient'altro che un insieme di molti w*orker process* che vengono eseguiti su più macchine all'interno del Cluster. Ogni *worker process*, esegue una JVM (**Java Virtual Machine**) che esegue una porzione di codice dell'intera applicazione. Analogamente, ogni worker esegue al suo interno più ***executor*** (*thread*) che, a loro volta, comprendono *uno o più task* paralleli in esecuzione (figura 3.3). Dalla figura, per l'appunto, si estrapolano due diversi tipi di executor. Una prima tipologia con un rapporto diretto (1:1) con i propri task; una seconda, al contrario, comprendente più task nello stesso processo (1: N).
Il numero degli executor è importante; permette di dare, difatti, con l'aumento del numero di task, maggiore flessibilità. Oppure può essere utile in fase di test (i.e. controllare che un nuovo frammento di codice lavori come desiderato).

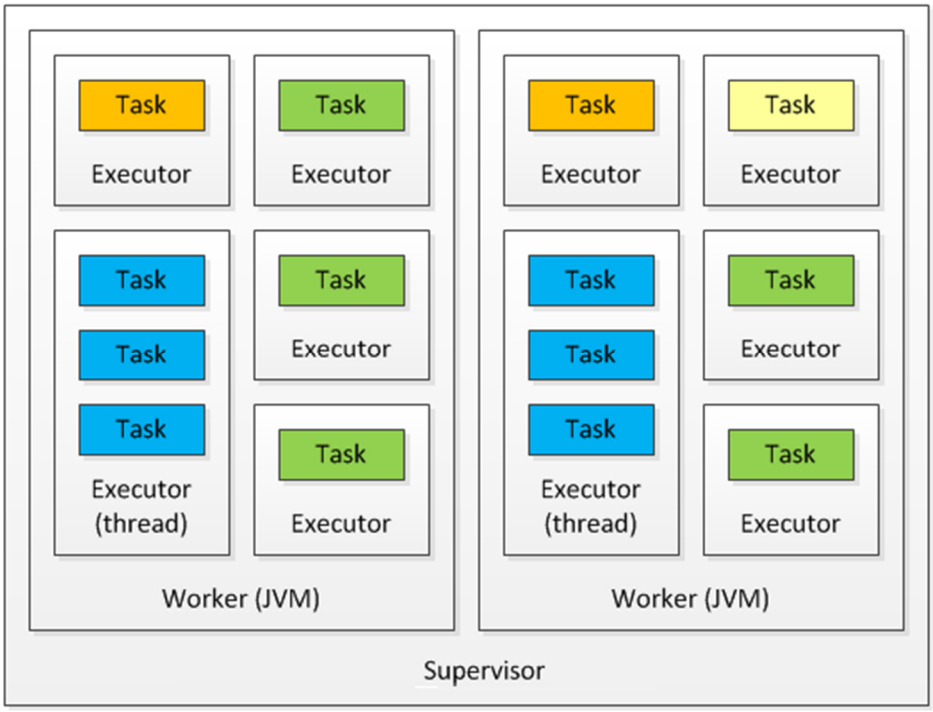


Figura 3.3 - Struttura di un nodo Supervisor all'interno di Apache Storm.

In conclusione a questa prima descrizione della struttura di Storm è proposto un breve approfondimento sulla comunicazione interna.
Per quella *node-to-node*, quindi "inter-worker", il sistema utilizza:

- *ZeroMq*, un *message-oriented middelware*[9] (*MOM*) installato nel Cluster.
- Netty, un framework di comunicazione event-driven asincrono per Java.

Per la comunicazione "intra-worker", e perciò *inter-thread*, invece, vengono segnalate le librerie ad alte performance ***LMAX Distruptor.***

Con questa serie di elementi, appena descritti, si riesce completamente a descrivere l'intera struttura "fisica" necessaria per sottomettere un qualsiasi progetto. Nel capito successivo verranno, invece, approfondite le logiche di elaborazione dei flussi di streaming introducendo la **topologia**, concetto cardine in un'applicazione Storm.

[9] Si intende un insieme di programmi informatici che fungono da intermediari tra diverse applicazioni e componenti software. Sono spesso utilizzati come supporto per sistemi distribuiti complessi.

3.2 Struttura logica, meccanismi di analisi e trasmissione

E' necessario, a questo punto, delineare le differenze a livello prettamente logico che distinguono un nodo dall'altro in base ai task assegnati e i vari elementi che ne derivano dal processo di design. Di seguito, per aiutare la comprensione, viene presentato un elenco dei "core concepts" [14] di Storm, descrivendone le caratteristiche e i ruoli che possono assumere in una determinata applicazione:

- **Nodo**: l'entità principale di tutto il sistema. Può assumere, come verrà approfondito dopo, il ruolo di **Spout** *(emettitore)* e/o di **Bolt** (esecutore).
- **Arco**: i nodi, naturalmente, sono collegati tra di loro. Questo collegamento rappresenta lo *stream di tuple* scambiate tra gli spout e i vari bolt.
- **Topologia**: la disposizione logica di tutti i componenti (spout e bolt) e le loro connessioni (stream). Rappresenta l'intero design dell'applicazione mediante, quindi, un *grafo orientato.*
- **Tuple**: una lista ordinata di valori ai quali è assegnato, per ognuno di questi, un nome.
- **Stream**: una sequenza di tuple scambiate tra uno *spout e un bolt* oppure *tra due soli bolt* identificata unicamente da una id.
- **Spout** (*emettitore*): la sorgente per definizione di uno stream. Una sorta di "canale live" di ricezione dati ritrasmessi ai vari bolt.
- **Bolt** (*esecutore*): accetta uno stream di tuple da uno spout o da un altro bolt. In genere l'esecuzione riguarda la computazione o la trasformazione delle tuple ricevute in input. In più, un bolt può opzionalmente emettere nuove tuple da lui appositamente rimodellate che fungono da stream input ad un altro bolt qualsiasi nella topologia.

Naturalmente, ogni spout e bolt hanno una o più istanze che sono eseguite completamente in parallelo. Nulla vieta ad un qualunque supervisor di eseguire sia task in veste di spout sia di bolt, anche contemporaneamente.
L'idea generale è quella di, dato uno stream in input di dati real-time, dividere le varie operazioni che compongono l'analisi tra più bolt in modo da analizzare più rapidamente ma in vari passaggi logici. Un esempio di topologia "Hello world", utilizzato anche come prima prova nel progetto di tesi, è il *Word Counter* (figura 3.2). Dato uno stream di dati, la topologia analizzerà le parole ricevute individuandone eventuali occorrenze.

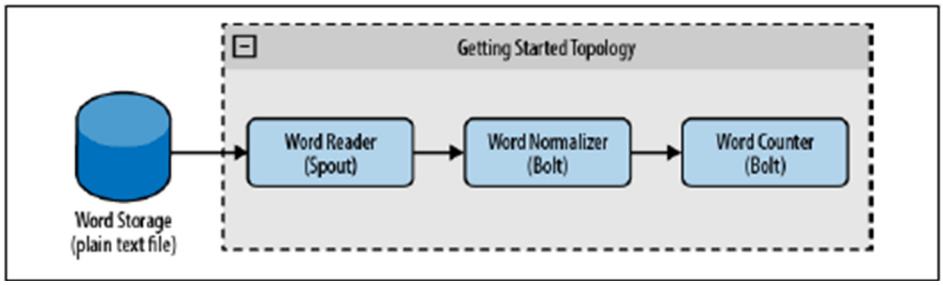


Figura 3.4 - Struttura Word Counter Topology

Come è possibile evincere dalla figura, la topologia è formata da uno nodo di emissione dati, quindi spout, (Word Reader) che effettua una lettura continua dallo *Word Storage*, in questo caso semplice flat file, emettendo le tuple create al primo nodo esecutore, quindi bolt (Word Normalizer). Quest'ultimo, si occupa solamente della normalizzazione delle varie tuple che, man mano che vengono analizzate, sono ritrasmesse con le relative modifiche al secondo bolt (Word Counter). A quel punto, l'ultimo bolt procede con il conteggio delle occorrenze.
Non è evidente nell'immagine ma l'ultimo bolt potrà, ovviamente, mostrare a video i risultati come anche inserirli in un database.
Da questo breve esempio diviene semplice intuire come, in un applicazione Storm, ciò che dà senso al sistema di esecuzione e analisi sono proprio gli stream definiti in fase di design e come le relative tuple vengono scambiate tra i componenti. In sostanza come i bolt elaborano i dati ricevuti.
In fase di progettazione di un sistema basato su Storm, per il precedente motivo, è necessario definire come le tuple debbano essere trasmesse per ottenere il risultato desiderato.
Ogni qual volta che uno spout o un bolt emettono una tupla questa viene posta, come già detto, in uno stream. Vari bolt possono sottoscrivere a un dato stream, dal quale ognuno di essi riceve una *esatta copia* di quella tupla.
Per ogni bolt possono esistere pertanto più stream in input. Nella gestione degli input, il comportamento predefinito è quello di smistare le tuple in modo da suddividere equamente il carico tra tutti i task di un bolt (in caso di parallelismo *1: N*). Per le architetture con un parallelismo in rapporto *1: N* (più task per ogni executor*)*, tuttavia, spesso, per particolari esigenze e requisiti di analisi, occorre inviare le tuple che condividono caratteristiche comuni ad un task specifico dei tanti presenti.
A tal fine Storm permette una vasta e completa scelta tra i meccanismi di stream *grouping* più vari. Uno stream grouping definisce, pertanto, come devono essere partizionate le tuple di un determinato data stream in input tra i vari task del bolt.

Di seguito i più utilizzati:

- Shuffle Grouping: il grouping predefinito, distribuisce equamente il carico tra tutti i task randomizzando il processo.
- Fields Grouping: le tuple sono suddivise in base al valore di uno o più campi. Tutte le tuple con una certa combinazione di campi vengono sempre inviate a un certo task. Ad esempio, se lo stream è raggruppato in relazione al campo "user-id", tuple con lo stesso "user-id" saranno sempre reindirizzate allo stesso task.
- Global Grouping: l'intero stream è inviato al task con id più basso.
- All Grouping: lo stream viene replicato in tutti i task del bolt.
- None Grouping*:* specifica che non ci si cura di come lo stream sia raggruppato. Attualmente è equivalente al normale Shuffle grouping in quanto la distribuzione, poiché senza particolari limiti, avviene random.

Nella pagina successiva è riportato uno schema grafico delle principali procedure di grouping mettendo in evidenza come possono essere gestiti gli stream in ingresso in un qualunque bolt.
Non bisogna, perciò, far confusione tra gli stream in input in un bolt generico e le tuple in input in un determinato task di quest'ultimo. La figura 3.5 sotto riportata spiega al meglio la differenza.

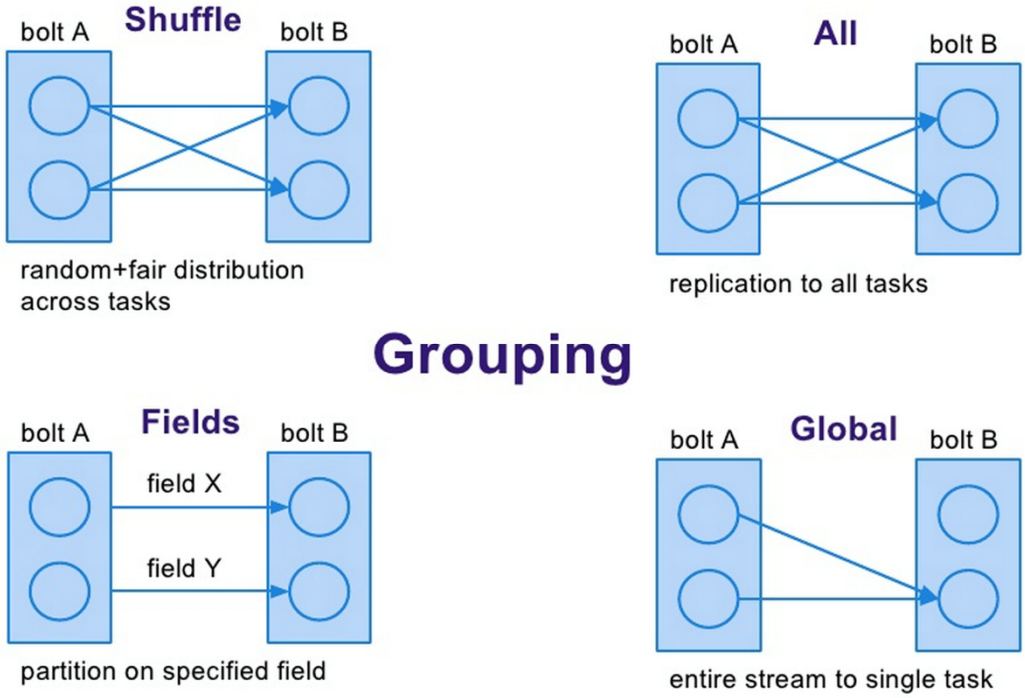


Figura 3.5 - Diverse tipologie di grouping in Apache Storm

La procedura di grouping, impostata in fase di creazione di una nuova istanza di un bolt, dipende soltanto da quel determinato bolt *e non interessa i precedenti da dove sono stato emettesse le tuple.* Per tanto, un bolt in fase di emissione si limita a creare gli stream inserendoci le tuple generate. Senza conoscere i suoi subscriber, in sostanza.
Naturalmente, per le architetture con parallelismo diretto, quindi con rapporto 1:1, il problema del grouping non sussiste. Un dato bolt che sottoscrive un particolare stream riceve in input tutte le tuple in arrivo sull'unico task disponibile.
Per il progetto di tesi, in fase di progettazione, viste le esigenze imposte dal dominio, si è deciso di adottare come principale processo lo Shuffle grouping, così da distribuire equamente il carico tra i componenti e i task annessi.
Segue, nella pagina successiva un breve approfondimento con un esempio esplicativo adeguato [15].
Già accennato in precedenza, lo Shuffle Grouping è un tipo di grouping che presenta un'emissione delle tuple completamente random. Per meglio comprendere la definizione sopra presentata, in figura 3.6, è riportato un esempio di applicazione concreta del meccanismo di smistamento.

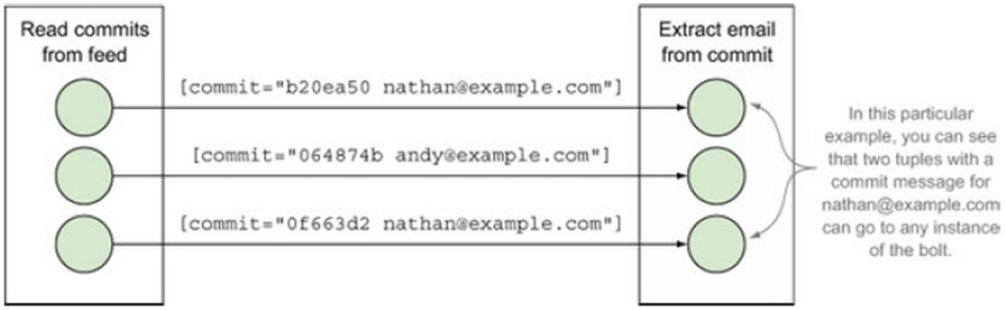


Figura 3.6 - Un esempio di Shuffle Grouping.

Come si evince dall'immagine, in questo particolare scenario non è infatti importante come le tuple siano inviate alle istanze dei bolt, in quanto sono tutti responsabili dello stesso compito di estrarre l'indirizzo e-mail.
Utilizzando un raggruppamento del genere verrà garantito un numero approssimativamente uguale di tuple ricevute da ogni task, distribuendo il carico su tutte le istanze disponibili. Questo raggruppamento è utile in molti casi di base in cui non si dispone di requisiti speciali su come i dati debbano effettivamente essere divisi.
Naturalmente in altri scenari, l'invio di tuple randomizzato non sarà sempre idoneo in base alle caratteristiche del dominio.

Difatti, estendendo l'esempio stesso con una serie di bolt in ascolto sui bolt di extracting, adibiti all'update delle mail in base al contenuto, alla tipologia di messaggio o a qualsivoglia fattore discriminante, l'efficienza dello *shuffle grouping* viene palesemente confutata facendo emergere il bisogno di un diverso tipo di *stream grouping* messo a disposizione dalla piattaforma.
Storm, per tal motivo, è un sistema di analisi completo; capace di soddisfare le richieste più particolari nel campo dell'analisi dei dati, garantendo sempre alte prestazioni. Nel paragrafo successivo, dopo questa carrellata sulle sue caratteristiche più tecniche, vengono infatti esposte le proprietà che lo rendono uno dei sistemi per il DSP più affidabili e performanti.

3.3 Proprietà di sistema e considerazioni

All'interno di tutti questi concetti di design e strutture complesse, ci sono alcune interessanti proprietà che rendono Storm unico nel suo genere. Di seguito le principali.

Affidabilità

L'affidabilità in un processo di streaming è fondamentale; la possibilità di perdita di dati, in questa realtà, infatti, è un fattore da considerare e può avvenire con una alta frequenza se non si gestisce al meglio il sistema.
Storm, per l'appunto, presenta due principali tipi di *guarantee:*

- *At-least-once,* affiancato da acknowledgment.
- *At-most-once,* **non** affiancato da acknowledgment.

Nella prima, ogni tupla è assegnata ad un id unico ed è tracciata in tutto il suo percorso che effettua nella topologia. Come è possibile notare in figura 3.7, per ogni tupla in output consegnata con successo nella topologia, viene utilizzato un meccanismo di *backflow* per verificare che effettivamente il trasferimento sia avvenuto.

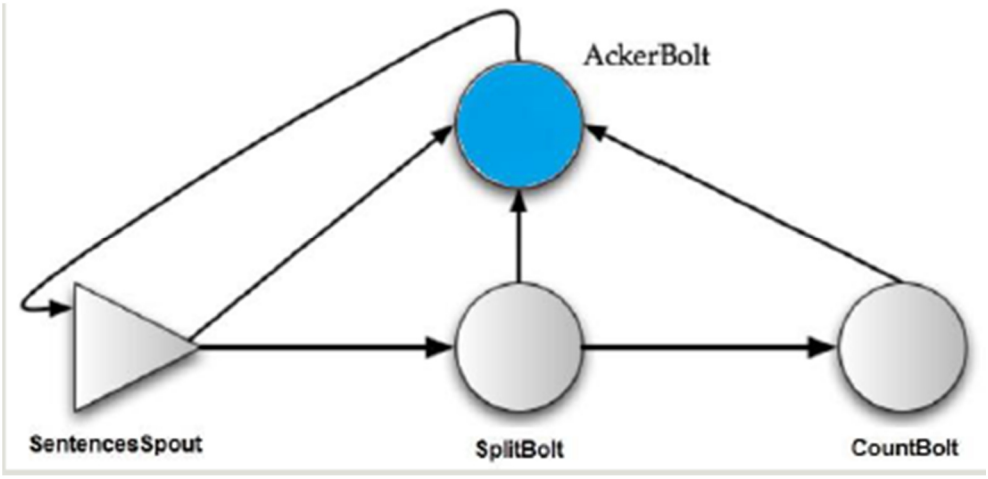


Figura 3.7 - Un meccanismo per la prevenzione alla perdita di dati adottata da Apache Storm.

Questo avviene emettendo la tupla in questione anche verso un *acker bolt* che si occupa di inviare acknowledge di conferma al *task dello spout.*
Se entro uno specifico *timeout,* regolabile anche dall'utente, la tupla non è stata notificata come correttamente ricevuta viene subito ritrasmessa.
Analogamente, quando il meccanismo di acknoledgement è disabilitato (ciò è possibile farlo tramite istruzioni da codice), Storm fornisce la semantica *at-most-once.* Con quest'ultima ogni tupla è eseguita solo una volta, ma persa in caso di fallimento.
Naturalmente per quanto appena scritto si tenderebbe a preferire palesemente la prima soluzione. Tuttavia, è facilmente intuibile che il meccanismo di acknowledgement visto il suo funzionamento ridondante e continuo comporta un uso maggiore delle risorse; disabilitandolo, perciò, le performance subiscono un miglioramento sostanziale, ed è questo il motivo per cui molte volte si ricorre alla seconda soluzione in relazione sempre a quelle che si presentano essere le esigenze del progetto. Fondamentalmente, tutti i messaggi sono garantiti nell'esecuzione *almeno una volta*. Se ci sono errori, i messaggi potrebbero sicuramente essere elaborati più di una volta, ma non verranno persi.

Semplicità nella programmazione

Il real-time processing risulta complesso, ci sono una serie numerosissima di fattori da dover considerare (i.e. latenza nella trasmissione, gestione delle code e dei buffer, perdita di dati, overhead). Con Storm, al contrario, la complessità si riduce drasticamente.

Supporto per più linguaggi di programmazione

Sicuramente appare più semplice sviluppare progetti in un linguaggio *JVM-based*, ma Storm supporta tutti i linguaggi di programmazione data la sua natura cross-platform.

Tolleranza agli errori (Fault-tolerant)

Un cluster Storm presenta un'alta soglia di tolleranza in caso di errore di esecuzione. A tal proposito il nimbus e i vari supervisor sono progettati per essere:

- *Fail-fast:* in caso di qualsiasi errore inaspettato i processi si arrestano immediatamente per dar la possibilità ad altri di essere riassegnati o loro stessi riavviati.
- *Stateless:* come detto in partenza, tutti gli status, i log di esecuzione etc sono mantenuti soltanto nello Zookeeper o su supporto disco.

Basti pensare che, nel caso in cui un worker cessi il suo funzionamento in modo improvviso e anomalo il processo supervisor ha la possibilità di, in primo luogo, riavviare subito il processo. In secondo luogo, in presenza di un persistente fallimento in esecuzione, quest'ultimo può riassegnare i task a nuovi worker in quel momento più liberi. Naturalmente, lo stesso succede a livelli gerarchici più alti tra il master node nimbus e i vari supervisor.

Scalabilità

In presenza di un aumento improvviso di dati che necessitano di un analisi che oltrepassa i limiti a massimo regime del sistema attualmente configurato, Storm permette processi di espansione delle risorse di calcolo anche noto come *scale up*[10], o di aumento del numero vero e proprio di nodi (macchine), ossia *scale out*.
Il sistema è per tal motivo dotato di una elevata scalabilità sia orizzontale che verticale.

Velocità

In fase di progettazione, visto lo scopo per cui Storm è stato inizialmente pensato, la velocità è una delle principali *key factor* del sistema di DSP.

[10] Si riferisce alla possibilità di aggiungere risorse, intese come CPU, memorie o dischi, al sistema per potenziare le performance quando il carico totale va oltre le capacità del sistema cluster stesso. Questo permette di aumentare le prestazioni con costi contenuti.

In conclusione, viste la serie di proprietà elencate e la struttura ben congeniata della piattaforma si può definire Storm una delle soluzioni più idonee per un progetto di *Big Data,* in quanto fornisce servizi di elaborazione ad alta velocità, con bassa latenza e massimo parallelismo. Il tutto in chiave di estrema affidabilità.
Per il progetto di tesi, come si spiegherà nel dettaglio nel prossimo capitolo, il dominio appartiene ad una realtà di stampo prettamente "real-time processing": **l'analisi dei netflow di dati scambiati dalla rete del campus universitario di Forlì-Cesena tra terminali interni e verso l'esterno.**

4. Applicazione del caso di studio

Nel corso del seguente capitolo verrà proposta una descrizione approfondita della parte sperimentale del progetto di tesi. Partendo dalla definizione del dominio applicativo fino ai risultati finali e la presentazione del primo prototipo dell'applicazione DSP, prodotta mediante lo strumento Apache introdotto nel capitolo precedente.

4.1 Descrizione del progetto

Il progetto si muove nel campo dell'analisi dei dati che, come più volte ripetuto, si contraddistingue per la sua vasta quantità di fonti. La sorgente di quest'ultimi, in questa particolare applicazione, è proprio la *rete Internet.* Il dominio applicativo, difatti, si concentra sull'analisi dei *netflow,* real-time streaming per l'appunto, che ogni giorno trasportano dati all'interno e all'esterno del campus universitario di Forlì-Cesena. Un traffico di dati potenzialmente campionabile e analizzabile al fine di estrapolare, in tempo mediamente reale, informazioni utili, statistiche ed eventuali *warning* annessi. Lo scopo del progetto, a tal proposito, fin dalla prima fase di definizione, possiede alla base tematiche di *cybersecurity.*
L'analisi continua dei vari flussi infatti, come è possibile intuire, oltre alle varie statistiche di routine, può effettivamente portare alla scoperta di situazioni "sospette" dal punto di vista della *sicurezza informatica* (i.e. intrusioni non identificate nei sistemi interni d'Ateneo a causa di *attacchi hacker*).
Il progetto, a tal scopo, è stato sviluppato, dopo una prima fase di setup e apprendimento degli strumenti da utilizzare, in collaborazione con il *centro di reti e sicurezza informatica* del Campus FC. Su gentile concessione di quest'ultimo, è stato reso possibile il reindirizzamento del traffico di rete verso il centro di elaborazione (*cluster*) configurato per l'occasione, i cui dettagli sono esplicati più avanti. Nel corso di tre mesi, è stata rilasciata una forma prototipale di *NetFlow Analyzer* che quotidianamente, rispetto alle richieste dell'ufficio di sicurezza, può fornire report informativi sullo status della rete, con eventuali segnalazioni dovute a particolari anomalie riscontrate nel traffico dati campionato.

4.2 Architettura utilizzata

Per lo sviluppo dell'applicazione è stato necessario anzitutto configurare un centro di elaborazione dati. Tutto il progetto, pertanto, è stato sviluppato su un cluster da 7 nodi di *commodity hardware* messo a disposizione dall'Ateneo, in particolare dal *Business Intelligence Lab*, che si occupa giornalmente di tali tematiche. Di seguito sono riportate le caratteristiche tecniche di ogni nodo:

- CPU: Intel i7-4790, 4 Core, 8 threads (*Hyper-Threading*), 3.6Ghz
- RAM: 32 GB
- HARD-Drive: 2x2TB HDD
- Ethernet: Gigabit
- Operative System: CentOS 6.6 (Linux)

Sul cluster è stato installato il modulo di analisi *Apache Storm* versione *0.10.0*. Dei sette nodi disponibili il numero 4 funge da *master-node*, ossia da *Nimbus*, e i restanti 6 da *worker-node,* ossia da *Supervisor.*
All'indirizzo IP locale del master, sulla *porta 8080*, in aggiunta, per facilitare la gestione e la supervisione di tutti i nodi durante l'esecuzione delle varie applicazioni e la visualizzazione delle varie applicazioni, Storm mette a disposizione una UI (*User Interface*) apposita, semplice e user-friendly (figura 4.1).
Dalla UI, oltre a questa prima schermata generale che permette una panoramica su quella che è la parte più tecnica in riferimento alla struttura *fisica* di Storm (i.e. numero di *worker*, numero di *executor*, parallelismo dei task), è integrata per ogni topologia, quindi per ogni applicazione sottomessa sul cluster, una pagina *.HTML* dedicata. In quest'ultima è disponibile la visualizzazione delle statistiche e della suddivisione *logica* dei componenti. Inoltre, informa su come il Nimbus man mano distribuisca i vari elementi di elaborazione (spout e bolt) sulle varie macchine (figura 4.2)

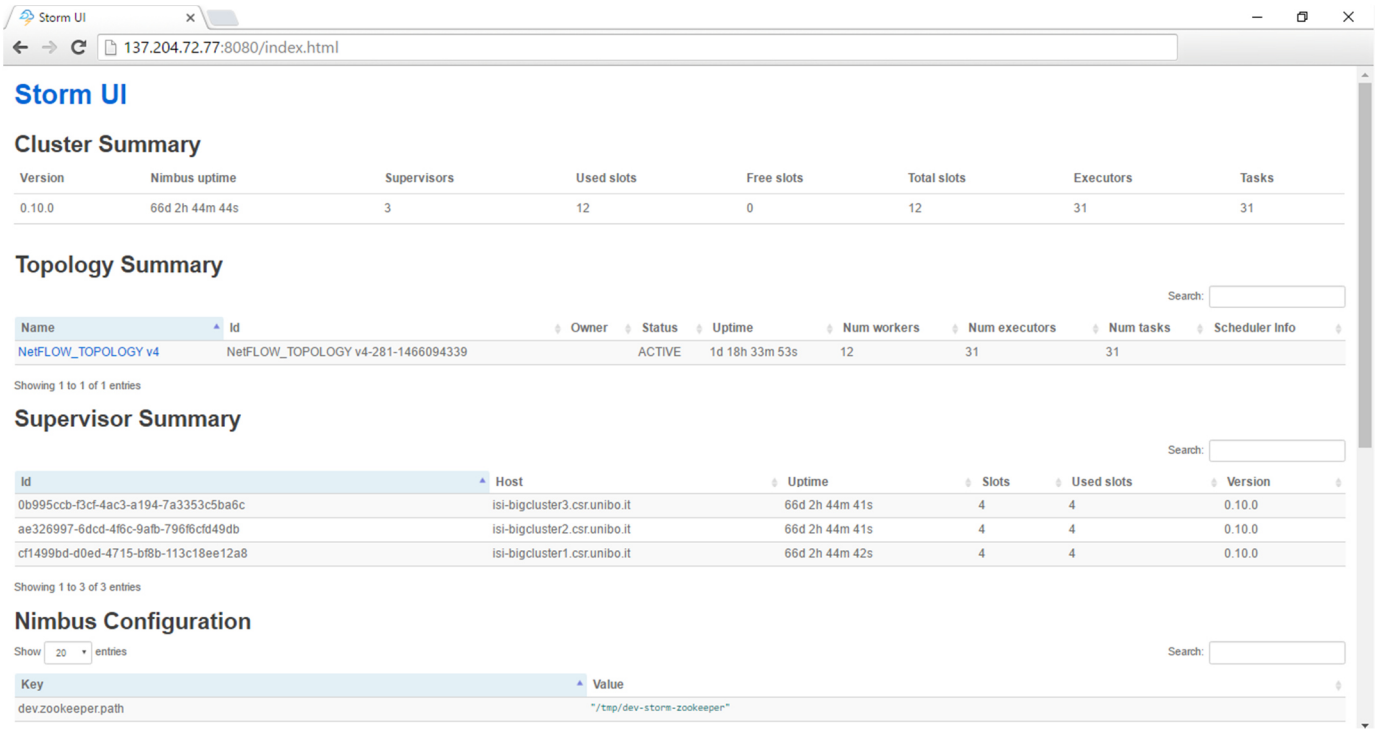


Figura 4.1 – Apache Storm UI, informazioni struttura fisica

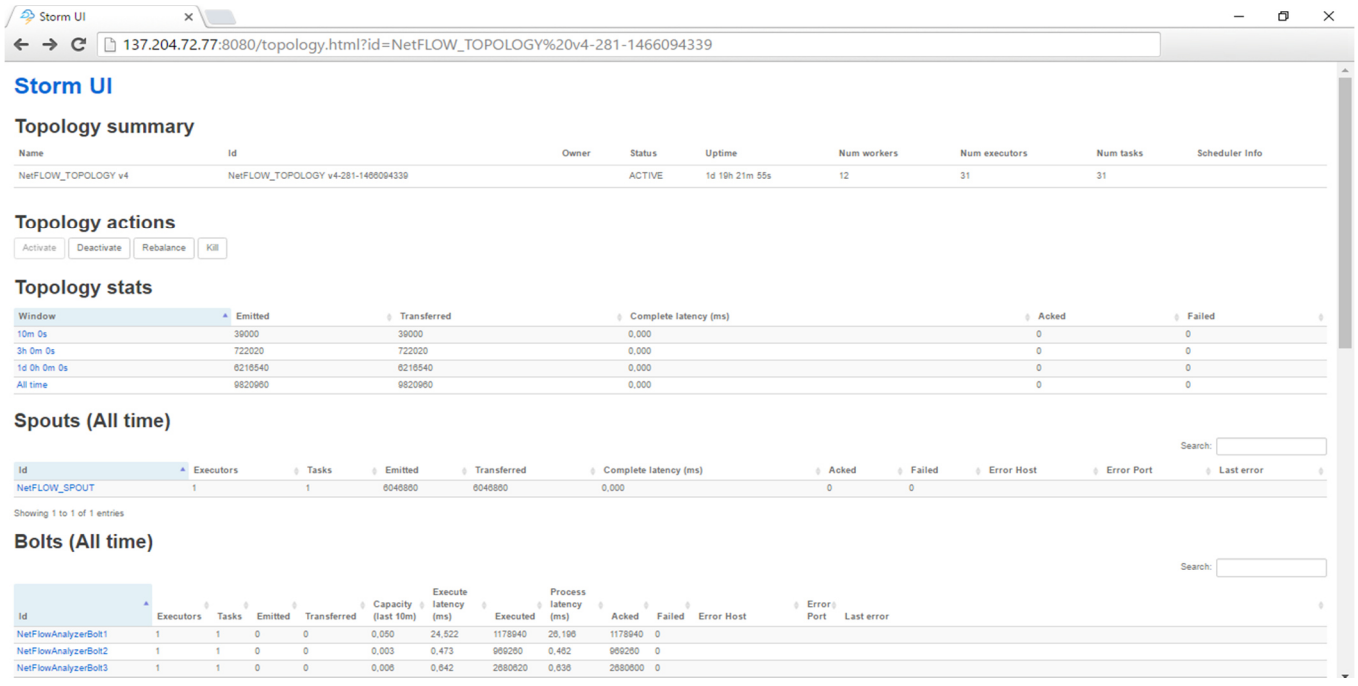


Figura 4.2 – Apache Storm UI, informazioni struttura logica

Il terzo elemento considerato parte integrante della struttura di analisi è proprio la sorgente dei dati, ossia il *firewall* da cui viene trasmesso il traffico di rete da analizzare. Viene brevemente spiegata la pratica utilizzata per il reindirizzamento di quest'ultimo sul cluster Storm. Il firewall Fortinet, modello Fortigate, utilizzato dal centro di sicurezza informatica del campus si occupa, come mansione principale, del filtraggio dei vari pacchetti che circolano tra le macchine collegate alla rete universitaria e quelle appoggiate su ISP[11] esterni, e quindi LAN[12] private. L'intero traffico di pacchetti, anzitutto, è stato reindirizzato sul server *csi-traffic.campusfc.unibo.it.* A questo punto il server, in ascolto sulla porta 5556, riceve il netflow e, tramite il software *nprobe,* trasforma il traffico dati in formato universale *JSON.* I dati convertiti sono trasferiti al master-node del cluster, intanto in ascolto, con un semplice demone appositamente creato per lo scopo, sulla porta 3333.
Di seguito uno schema dell'intera struttura per la raccolta dei dati dalla rete appena descritta (figura 4.3).

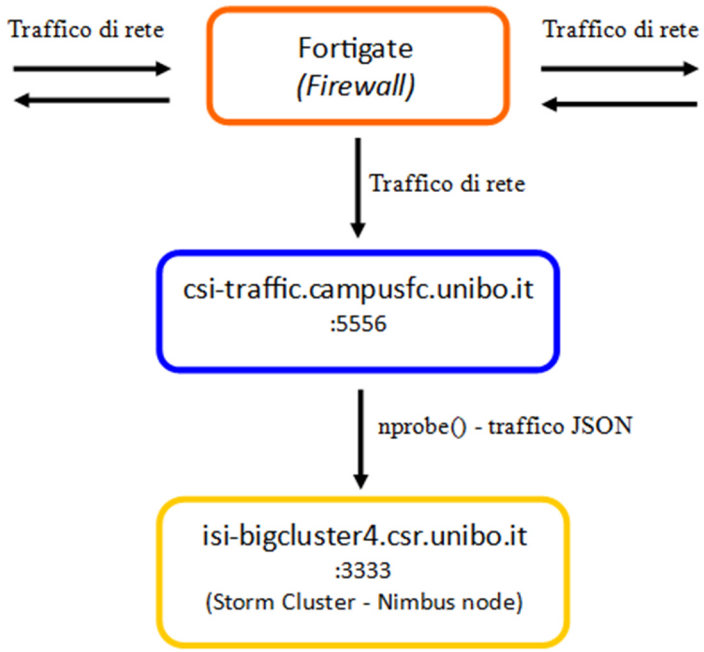


Figura 4.3 - Meccanismo di reindirizzamento del traffico di rete sul cluster Storm

[11]Si intende una struttura commerciale o un'organizzazione che offre agli utenti (residenziali o imprese), dietro la stipulazione di un contratto di fornitura, servizi inerenti a Internet
[12]Fa riferimento ad una rete informatica di collegamento tra più computer, estendibile anche a dispositivi periferici condivisi, che copre un'area limitata, come un'abitazione, una scuola, un'azienda o un complesso di edifici adiacenti.

4.3 Design e topologia

Nel capito 3 è stato possibile notare come possa essere intuitivo progettare sistemi di Data Stream Processing tramite l'utilizzo della piattaforma offerta da Apache. La struttura e il design infatti, sono composti da una seria limitata di componenti, ognuno con uno o più task specifici interagenti tra loro secondo i requisiti che l'analisi richiede. Nel corso di questo paragrafo si descrivono quelli che sono i componenti utilizzati e la logica adottata nella realizzazione dell'applicazione sviluppata. L'intero sistema si basa sulla composizione di diversi elementi che, data la sua scalabilità, è possibile in ogni momento replicare. La topologia è, perciò, così composta:

- 1 **NetflowManagerSpout** collegato al netflow con filtering dei dati in input.
- 6 **SplitterBolt** per lo splitting concettuale ed emissione tramite hashing delle tuple.
- 4 **SessionAnalyzerBolt** per la mappatura e il tracciamento delle sessioni.
- 1 **WarningBolt** per le eventuali segnalazioni.
- 1 **LoggerBolt** per la memorizzazione dei dati utili per le statistiche.

E' riportata di seguito una descrizione dettagliata per ogni elemento facendo particolare attenzione a come collaborano nell'elaborazione delle tuple specificando i task assegnatigli.

Netflow Topology

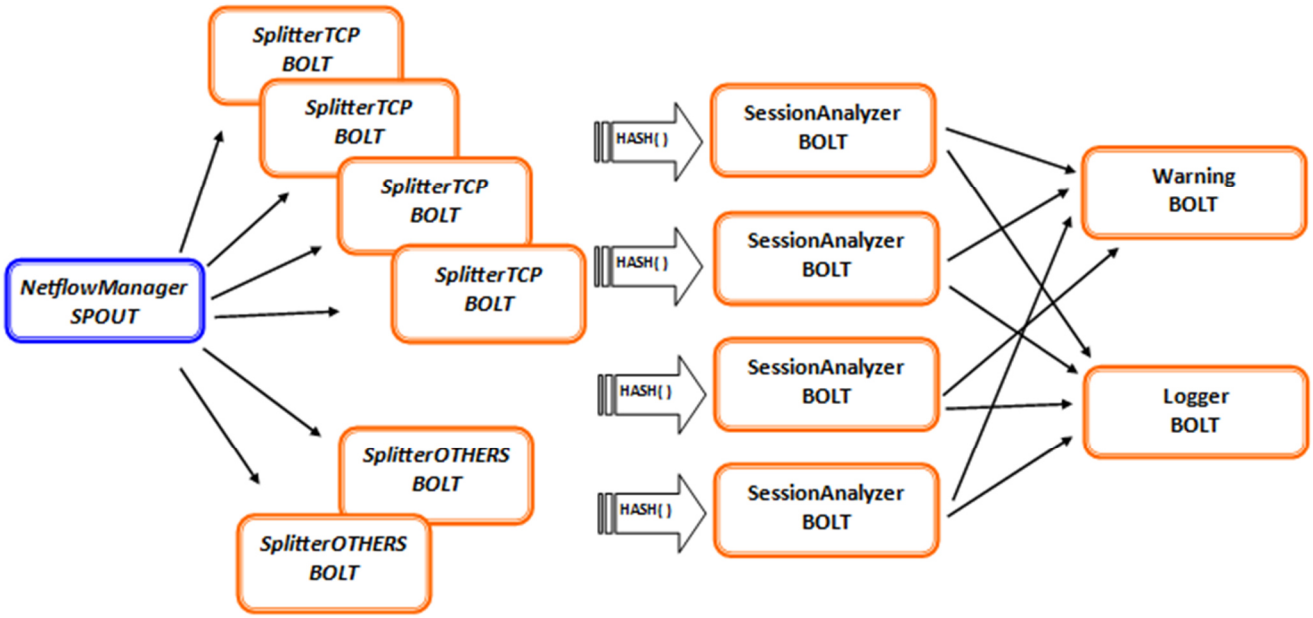


Figura 4.4 - Struttura e componenti Netflow Topology

In figura 4.4, la topologia completa del sistema. Da notare che potenzialmente tutti gli SplitterBolt, in base al reindizzamento con hashing (mediante *sharding*), possono comunicare con tutti i SessionAnalyzer. Un'informazione, perciò, percorre *almeno una volta* (principio at-least-once) tutto il percorso dall'emissione iniziale del NetflowManagerSpout fino, dopo le dovute modifiche dei bolt intermedi, ai due end-point che ne ricavano l'output.

NetflowManagerSpout

Una volta che il flusso di dati è giunto al cluster, l'intera applicazione può iniziare ad elaborare il traffico in input. Tuttavia, il nodo master, cioè il Nimbus, si occupa soltanto di funzioni di scheduling delle risorse e gestione dei componenti. Il ruolo di emettitore (spout), per tal motivo, non potrà ricadere direttamente su quest'ultimo. Occorre, perciò, instaurare un canale di comunicazione diretto con gli altri nodi a cui verrà assegnato il vero e proprio task di generazione ed emissione delle tuple da analizzare, filtrando quelle non utili a fini analitici. Inizialmente, per l'appunto,

non è stato possibile far convergere direttamente il flusso su uno dei 6 supervisor, in quanto Storm, come probabilmente già accennato, è responsabile della distribuzione dei componenti logici (spout e bolt) non permettendo all'utente di gestire come meglio preferisce i vari assegnamenti. Proprio per la sua natura real-time, è plausibile che il sistema prenda le decisioni in automatico di come effettuare al meglio il *deployment* dei vari worker in base a fattori ben precisi (i.e. preferire un nodo con latenza più bassa rispetto ad un altro, o anche in base alle risorse disponibili in quel momento). Questa piccola problematica è stata risolta sviluppando uno script che è in grado, una volta sottomessa la topologia sul cluster, di creare una comunicazione tramite socket tra il nodo master, che perciò fungerà da *gateway* con la rete esterna, e uno dei nodi assegnato come emettitore (*spout*).
I compiti dello spout (figura 4.5), che si presenta nella topologia con una sola istanza, sono molteplici:

- Mettersi in comunicazione con il nodo master per ricevere il flusso dalla rete esterna.
- Dividere il traffico utilizzando come discriminante il protocollo che i pacchetti utilizzano.
- Filtrare, facendo riferimento ad una **blacklist,** i dati più comuni che, ai fini dell'analisi, risulterebbero poco interessanti.
- Emettere le tuple modificate verso gli SplitterBolt.

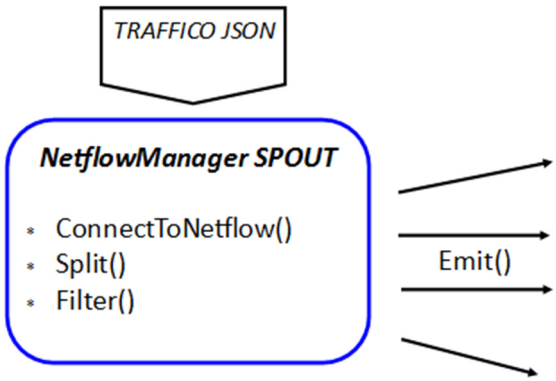


Figura 4.5 - Struttura e metodi del NetflowManager Spout

SplitterBolt

Il *NetflowManagerSpout* emette, perciò, le tuple divise secondo il protocollo di comunicazione utilizzato verso i vari *SplitterBolt* in ascolto su quel determinato stream. Per effettuare una raggruppamento logico delle sessioni si distinguono due

diversi tipi di *SplitterBolt*. Una versione che elabora soltanto le tuple contenenti informazioni inerenti a sessioni *TCP* e una dedicata a tutti gli altri protocolli di rete individuati (*UDP*, *ICMP*, etc).
Lo *SplitterBolt* (figura 4.6), a sua volta, indipendentemente dalla sua specializzazione, effettua le seguenti operazioni di base:

- Eseguire operazioni di *splitting* sulle tuple ricevute incapsulando il risultato escludendo alcuni campi a favore di altri, utili per il tracciamento delle sessioni e per l'analisi in generale.
- Per ogni tupla, creare una chiave primaria per identificare la sessione di appartenenza.
- Riemettere, verso i *SessionAnalyzerBolt*, le tuple riorganizzate secondo una *chiave di hash* basata sull'identificativo di ogni sessione.

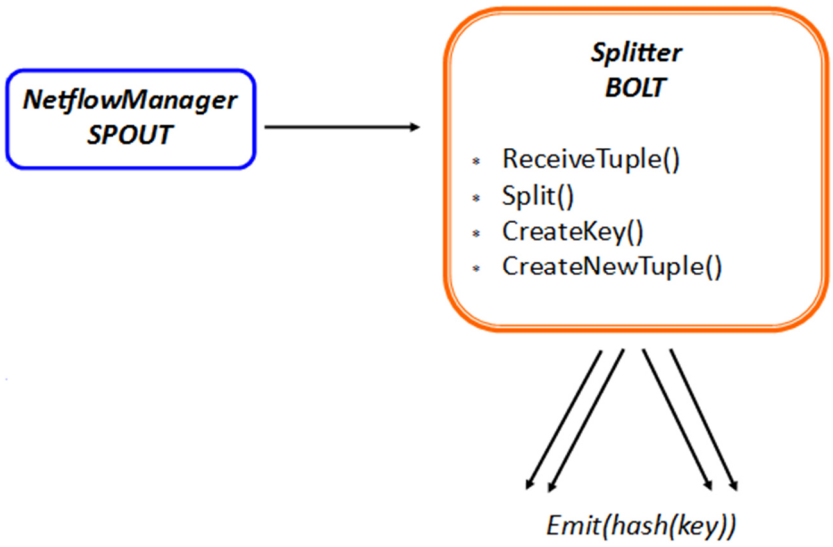


Figura 4.6 - Struttura e metodi dello SplitterBolt

L'operazione di *splitting* rimuove, come già detto, una parte poco importante di ogni pacchetto che percorre la topologia. Le tuple, infatti, già in questa seconda elaborazione, acquisiscono una forma ben definita. Da ogni pacchetto sono estrapolate informazioni riguardo i seguenti campi:

- Indirizzo IP sorgente + porta.
- Indirizzo IP destinazione + porta.
- Viene mantenuto il protocollo di comunicazione.
- Quantità di byte trasferiti per ogni pacchetto ricevuto.

In fase di esecuzione, inoltre, prima dell'effettivo incapsulamento e creazione delle nuove tuple, viene generata, per ognuna di esse, una chiave identificativa primaria formata secondo la logica:

Indirizzo IP maggiore + porta – Indirizzo IP minore + porta

Così facendo il *verso* di trasferimento non è più considerato ed ogni tupla in input è mappata rispetto ad una ed unica sessione di rete.
Di seguito un esempio:

IP sorgente + porta = **137.204.78.77:123**
IP destinazione + porta = **109.104.89.6:456**

In questo caso, l'indirizzo IP maggiore è ovviamente il primo. La chiave testuale che identificherà la sessione sarà quindi **137.204.78.77:123 - 109.104.89.6:456**. Ciò significa che le informazioni riguardanti quella sessione, ogni qual volta verranno riscontrate, in un verso di comunicazione o nell'altro (indirizzi IP scambiati), sono ricondotte sempre alla stessa chiave e quindi alla stessa tupla. In questo modo non solo è possibile avere un'analisi completa e attendibile sulle sessioni, ma si dimezzano drasticamente il numero di informazioni da mantenere in memoria, agevolandone le prestazioni in esecuzione.

Le tuple, inizialmente ricevute in forma "grezza", sono riorganizzate ed emesse secondo uno schema di campi ben definito:

1. Chiave primaria.
2. Timestamp di creazione.
3. Timestamp di update dei valori.
4. Byte scambiati.
5. Protocollo di comunicazione.

A livello di codice sorgente, le tuple emesse sono incapsulate in istanze della classe *SessionDetail*[13], strutturata nel medesimo modo.
E' essenziale, inoltre, che una tupla appartenente ad una determinata sessione (stessa chiave primaria) dopo essere ricevuta in input dal netflow ed emessa dallo spout percorra sempre lo stesso percorso tra i nodi del cluster. Questo requisito è ampiamento soddisfatto con l'utilizzo di una funzione di hashing che permette la

[13] Source Code: bitbucket.org/alessiomagno/design-and-prototyping-of-an-apache-storm-application-tesi

possibilità di istanziare più bolt dello stesso tipo e traferire determinate tuple verso specifici bolt; evitando in ogni caso che informazioni di sessioni diverse vengano ricevute ed elaborate da componenti differenti.

SessionAnalyzerBolt

L'ultima porzione del design è rappresentata dal *SessionAnalyzerBolt* (figura 4.7), parte finale della topologia. Quest'ultimo, dopo aver ricevuto, mediante reindirizzamento in chiave di hashing, le tuple dagli SplitterBolt produce sostanzialmente l'output finale. Suoi task principali sono:

- Allocare memoria necessaria per poter tracciare contemporaneamente numerose sessioni in un'apposita tabella (mappa *"key-sessionDetail"*).
- Elaborare le informazioni estrapolate dalle tuple in input individuando sessioni nuove, o già riscontrate, aggiornandone i valori.
- Controllare periodicamente la tabella di tracciamento *eliminando sessioni concluse* e individuando eventuali valori anomali da segnalare.

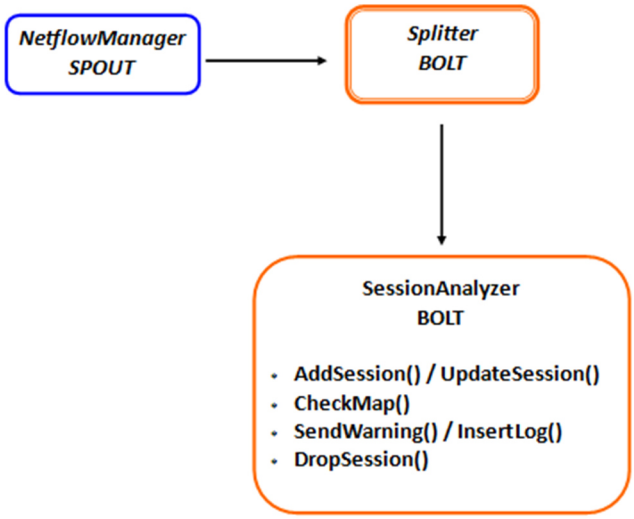


Figura 4.7 - Struttura e metodi del SessionAnalyzerBolt

Ogni SessionAnalyzerBolt mantiene, in memoria, una propria mappa dove per ogni chiave, rappresentata dalla primaria definita nel bolt precedente, corrispondono i valori di sessione organizzati secondo lo schema precedentemente riportato.
Quando un SessionAnalyzerBolt riceve una determinata tupla, per prima cosa, controlla se è già presente in memoria. In tal caso, facendo riferimento ai valori della tupla, aggiorna la vecchia versione salvata in memoria, modificando il *timestamp di update* a quel preciso istante. Se, al contrario, la tupla appartiene ad

una nuova sessione ancora mai riscontrata, quest'ultima viene inserita nella mappa e man mano aggiornata nel corso della comunicazione.
Dopo il controllo della tupla e l'eventuale aggiornamento/creazione, il processo è congeniato per controllare l'intera mappa al fine di individuare tre diverse situazioni ed emetterle verso i bolt specifici:

- Sessioni scadute/concluse -> **LoggerBolt** (*eliminando* dalla mappa la sessione in questione, liberando spazio in memoria).
- Sessioni lunghe (una durata insolita di comunicazione) -> **WarningBolt**.
- Sessioni "pesanti" (un numero elevato di byte scambiati) -> **WarningBolt**.

LoggerBolt e WarningBolt

Nel caso in cui il LoggerBolt o il WarningBolt ricevano un evento dei precedenti, se si tratta di sessioni scadute, o concluse, il primo si limita a memorizzare un *log* in un *flat file* che risiede sulla macchina. Altrimenti, in presenza di rilevamenti ben più allarmanti, come sessioni pesanti o lunghe, il WarningBolt crea un *report* specifico con tutte le generalità della sessione "sospetta".
Un attività sospetta legata ad una sessione può avere varia natura, e potrebbe anche, in realtà, essere ricondotta a nessuna vera e propria segnalazione importante. Per sessioni pesanti, per esempio, intendiamo quelle macchine che di continuo scaricano o distribuiscono grandi file. Tali macchine devono essere individuate comprendendone che tipo di attività stanno effettuando. Un server che distribuisce ISO per la didattica, infatti, è normale che costituisca tante sessioni pesanti. Non è invece normale che un PC di un laboratorio ne presenti altrettante. Questo può indicare attività di download da siti di file-sharing o anche che su tale macchina sia stato installato un servizio non autorizzato di sharing. Alcune botnet[14], a tal proposito, utilizzano le macchine colpite per depositare materiale coperto da copyright ma possono essere facilmente individuate perché generano, appunto, sessioni pesanti. Per ciascuna di esse bisogna capire che tipo di attività si sta effettuando.

Una sessione lunga, analogamente, potrebbe dar modo di scoprire delle intrusioni "silenti" nel sistema interno a causa di *malware* (e.g. Trojan) che in background

[14]Una botnet è una rete formata da dispositivi informatici collegati ad Internet e infettati da malware, controllata da un'unica entità, il *botmaster*. A causa di falle nella sicurezza o per mancanza di attenzione da parte dell'utente e dell'amministratore di sistema, i dispositivi vengono infettati da virus informatici o trojan i quali consentono ai loro creatori di controllare il sistema da remoto.

continuano a trasmettere verso l'esterno informazioni di dominio privato (i.e. dati sensibili di utenti, configurazioni di sistema etc.).
Inoltre, è possibile introdurre *tunnel VPN* sempre attivi non autorizzati, come anche oppure *reverse tunnel ssh*. Anche tali casi possono indicare l'esistenza di una potenziale botnet all'interno del PC.
Come nel caso precedente, prima di giungere a conclusioni affrettate scambiando una segnalazione per qualcosa piuttosto che un'altra si parte individuando chi utilizza quel determinato IP.

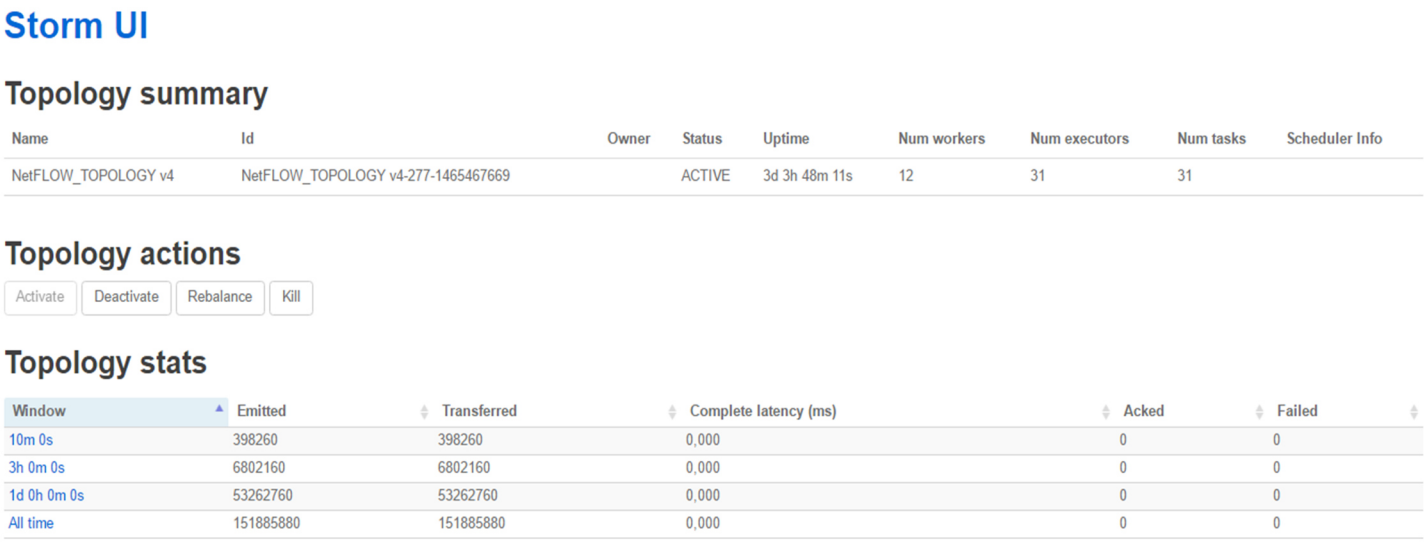
Motivo per cui, il report con i vari warning può essere trasmesso (i.e. *via email*) dal **WarningBolt** al centro di sicurezza dove verrà valutato se ignorarlo o continuare con l'indagine con la competenza dei tecnici specializzati.
Ovviamente, l'individuazione di queste ipotetiche "intrusioni" o "attacchi" di rete avviene in base a delle soglie precise per ogni diverso contesto. I valori standard per le sessioni scadute, per quelle pesanti e per quelle lunghe sono stati forniti dal centro di sicurezza e verranno approfonditi nel paragrafo successivo.
In sintesi, l'output dell'intera applicazione è dato quindi dai **LoggerBolt** e dai **WarningBolt**.

4.4 Test e simulazioni

Successivamente alla fase di design e di sviluppo che hanno portato alla creazione di un prototipo di applicazione DSP in Apache Storm, si è proceduto con una fase di testing.
I test, naturalmente effettuati sul cluster, sono stati gestiti secondo direttive comunicate dal centro di sicurezza. E' stata fornita, innanzitutto, una **lista di regole** per rendere sensato il filtraggio dei dati in input, e quindi delle tuple.
Dai dati in questione, a tal proposito, sono stati rimosse in partenza le informazioni relative a:

- Rete interna verso rete interna.
- Rete Wi-Fi verso rete interna, e viceversa.
- Rete interna verso videosorveglianza, e viceversa.
- Dispositivi network verso rete, e viceversa.
- Rete verso terminali Server, e viceversa.
- Rete stampanti verso terminali stampanti.

In questo modo le tuple effettivamente analizzate sono ridotte notevolmente limitando l'analisi ad una sfera di campionamento specifica non ridondante.
Per il processo di individuazione delle eventuali anomalie, invece, sono state fornite delle soglie da considerare come standard per ogni eventi da valutare.
Per le sessioni lunghe, ad esempio, è stato impostato come valore base una soglia di *9 ore di attività* di sessione *continua*. Oltrepassato la sessione può definirsi "lunga" e perciò necessita di segnalazione.
Analogamente, una sessione per definirsi "pesante" deve aver superato, nel corso dell'intero trasferimento, una soglia di 4096MB (4 Gigabyte) *scambiati*.
Una sessione scaduta/chiusa, infine, rispetto alle configurazioni di default del firewall, nonché sorgente dei dati, non può superare *1 ora di inattività*. Passato l'intervallo di tempo, se non sono stati riscontrati ulteriori aggiornamenti, la sessione scaduta può definirsi tale e viene, come già illustrato, eliminata dal processo di tracciamento e storicizzata per tutte le possibili future analisi.
Dalla Storm UI (interfaccia web) della piattaforma, è stato possibile seguire tornate di test, della durata media di 3 giorni. L'ultima svolta nel corso di un weekend. Di seguito alcune schermate che ne documentano lo svolgimento (figure 4.8 e 4.9).



Storm UI

Topology summary

Name	Id	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Scheduler Info
NetFLOW_TOPOLOGY v4	NetFLOW_TOPOLOGY v4-277-1465467669		ACTIVE	3d 3h 48m 11s	12	31	31	

Topology actions

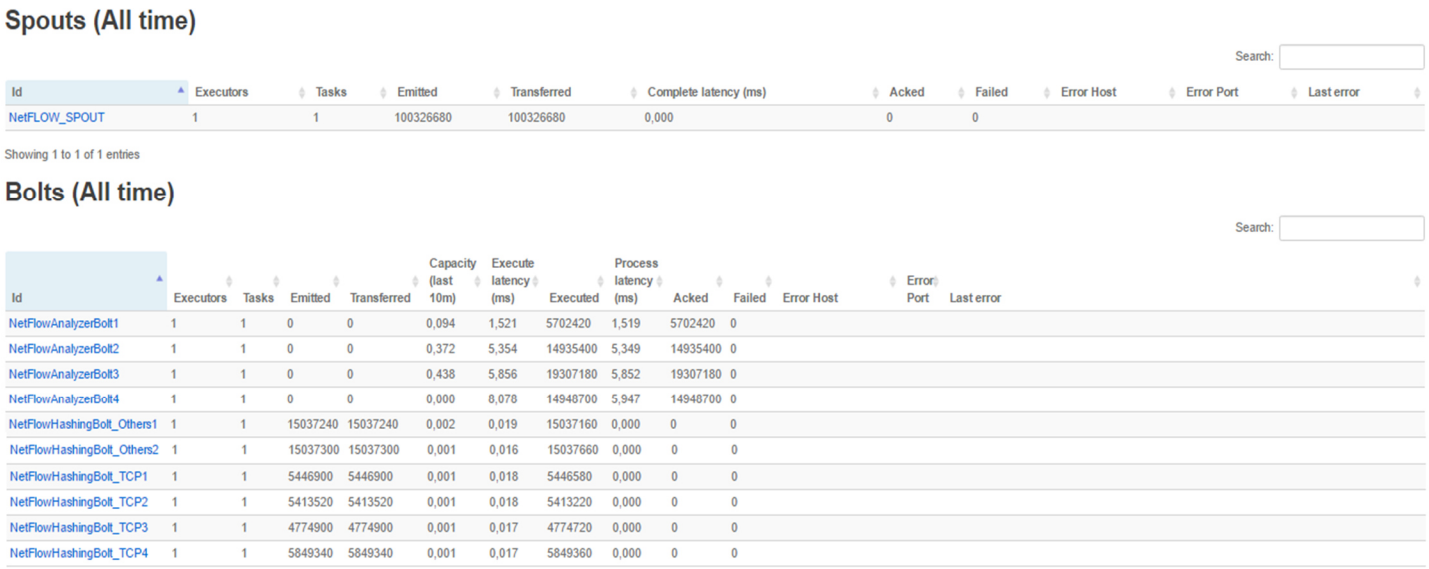
Activate | Deactivate | Rebalance | Kill

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	398260	398260	0,000	0	0
3h 0m 0s	6802160	6802160	0,000	0	0
1d 0h 0m 0s	53262760	53262760	0,000	0	0
All time	151885880	151885880	0,000	0	0

Figura 4.8 - Storm UI, attività di test dell'applicazione

In figura 4.8 è possibile notare come, in un lasso di tempo sostanzialmente limitato, siano stati emesse e, successivamente trasferite, più di 150 milioni di tuple. Ossia, più di 150 milioni di pacchetti contenenti informazioni di rete. Scendendo più nel dettaglio (figura 4.9) si riporta una panoramica generale sui diversi parametri di esecuzione dei vari componenti; *il* tempo di esecuzione, le tuple verificate tramite *ack*, le tuple non trasferite, eventuali errori run-time, etc.



Spouts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error
NetFLOW_SPOUT	1	1	100326680	100326680	0,000	0	0			

Showing 1 to 1 of 1 entries

Bolts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error
NetFlowAnalyzerBolt1	1	1	0	0	0,094	1,521	5702420	1,519	5702420	0			
NetFlowAnalyzerBolt2	1	1	0	0	0,372	5,354	14935400	5,349	14935400	0			
NetFlowAnalyzerBolt3	1	1	0	0	0,438	5,856	19307180	5,852	19307180	0			
NetFlowAnalyzerBolt4	1	1	0	0	0,000	8,078	14948700	5,947	14948700	0			
NetFlowHashingBolt_Others1	1	1	15037240	15037240	0,002	0,019	15037160	0,000	0	0			
NetFlowHashingBolt_Others2	1	1	15037300	15037300	0,001	0,016	15037660	0,000	0	0			
NetFlowHashingBolt_TCP1	1	1	5446900	5446900	0,001	0,018	5446580	0,000	0	0			
NetFlowHashingBolt_TCP2	1	1	5413520	5413520	0,001	0,018	5413220	0,000	0	0			
NetFlowHashingBolt_TCP3	1	1	4774900	4774900	0,001	0,017	4774720	0,000	0	0			
NetFlowHashingBolt_TCP4	1	1	5849340	5849340	0,001	0,017	5849360	0,000	0	0			

Figura 4.9 - Storm UI, attività di test dell'applicazione

Dalla UI, inoltre, è disponibile una versione sempre aggiornata di una visualizzazione della topologia (figura 4.10), utile per valutare la corretta distribuzione degli stream di dati e l'eventuale anomalia in esecuzione di un qualsiasi nodo. In base alle scale cromatiche visibili sul pannello laterale, fondamentalmente, un nodo verde lascia intendere un esecuzione regolare del componente. Rosso, invece, comunica una situazione di sovraccarico della memoria.

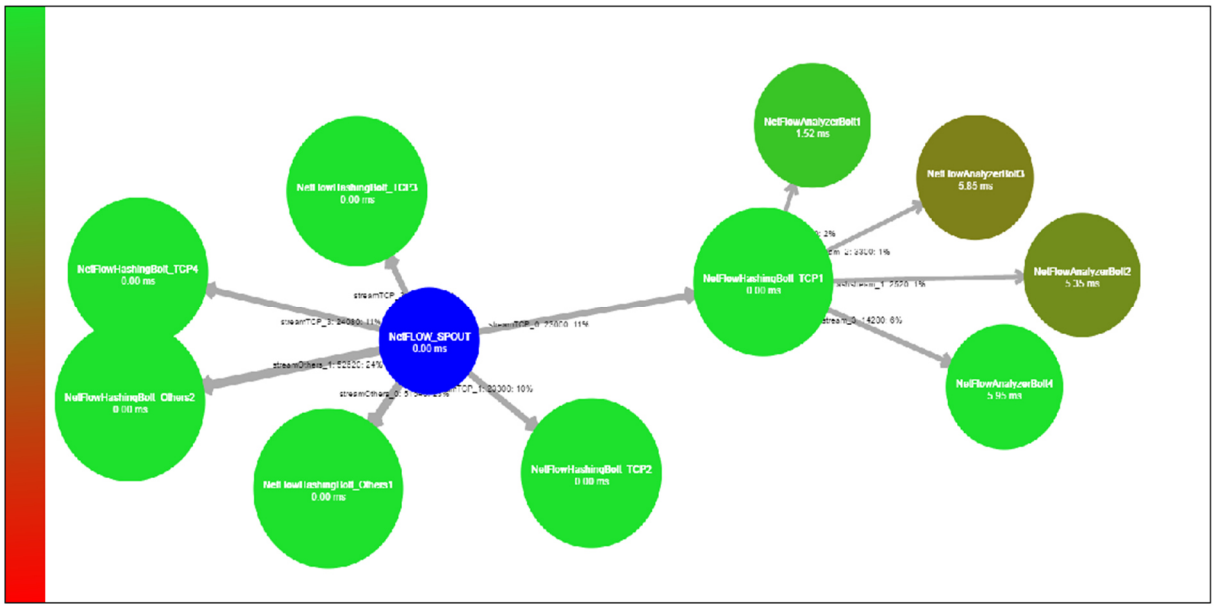


Figura 4.10 - Storm UI, attività di test dell'applicazione: visualizzazione della topologia.

La figura riporta tutti i possibili stream che mettono in comunicazione il NetflowManagerSpout (*in blue*) con i 6 SplitterBolt (*in verde*) ad esso direttamente collegati. Sono stati omessi, per rendere più comprensibile la visualizzazione, gli stream che partono da ogni SplitterBolt verso i 4 SessionAnalyzerBolt (*due dei quali in verde scuro*) riportandone soltanto un caso.
Il colore verde dei nodi, come anche le percentuali adeguatamente distribuite su ogni stream, sono sintomo di un'organizzazione delle risorse ben strutturata. Invece, il colore scuro degli ultimi nodi, i SessionAnalyzer, comunica un sovraccarico di lavoro in due dei totali quattro bolt. In fase di esecuzione tuttavia, visti i numeri di tuple riportati, è plausibile che più nodi, anche temporaneamente, attraversino determinati intervalli di tempo in cui l'attività si presenta a massimo regime. Nel corso dei vari test, in ogni caso, non si sono verificati eventi di perdita dei dati, di malfunzionamenti delle macchine, o di esaurimento della memoria disponibile.

4.5 Risultati

Questo paragrafo è dedicato ai risultati ottenuti dopo la prima fase di test. Al termine dei tre giorni sono state riscontrate, su un totale di 150 milioni di tuple scambiate, circa 450mila sessioni scadute/chiuse comprendenti sia sessioni di comunicazione TCP sia messaggi di rete scambiati mediante altri protocolli (UDP, ICMP) in cui non è concettualmente corretto poter parlare di sessione. In questo secondo caso, infatti, si considera "sessione" l'insieme di tutte le comunicazioni avvenute in un lasso di tempo raggruppate.
Le sessioni, sono state inserite nel file system distribuito di Hadoop, presente sul cluster, a cui è stato integrato l'utilizzo di *Tableau,* un software utile per la creazione di viste, statistiche e grafici riguardo dati direttamente presenti sulle basi di dati.
In figura 4.11, per esempio, in relazione alle sessioni storicizzate è possibile notare come delle totali, l'87% circa non supera i 10KB di trasferimento, il 3% non va oltre i 20KB e così via.

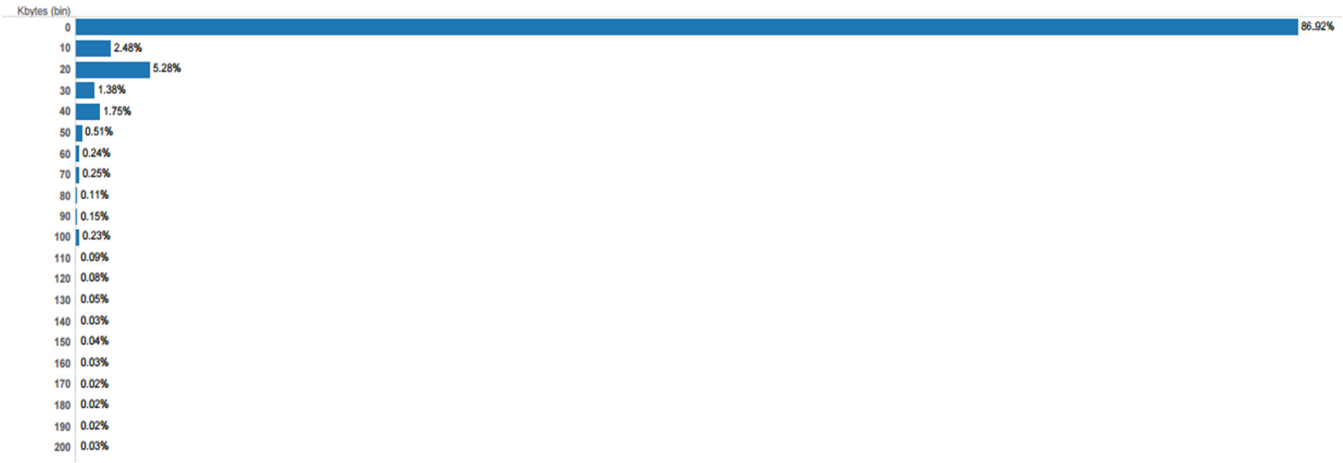
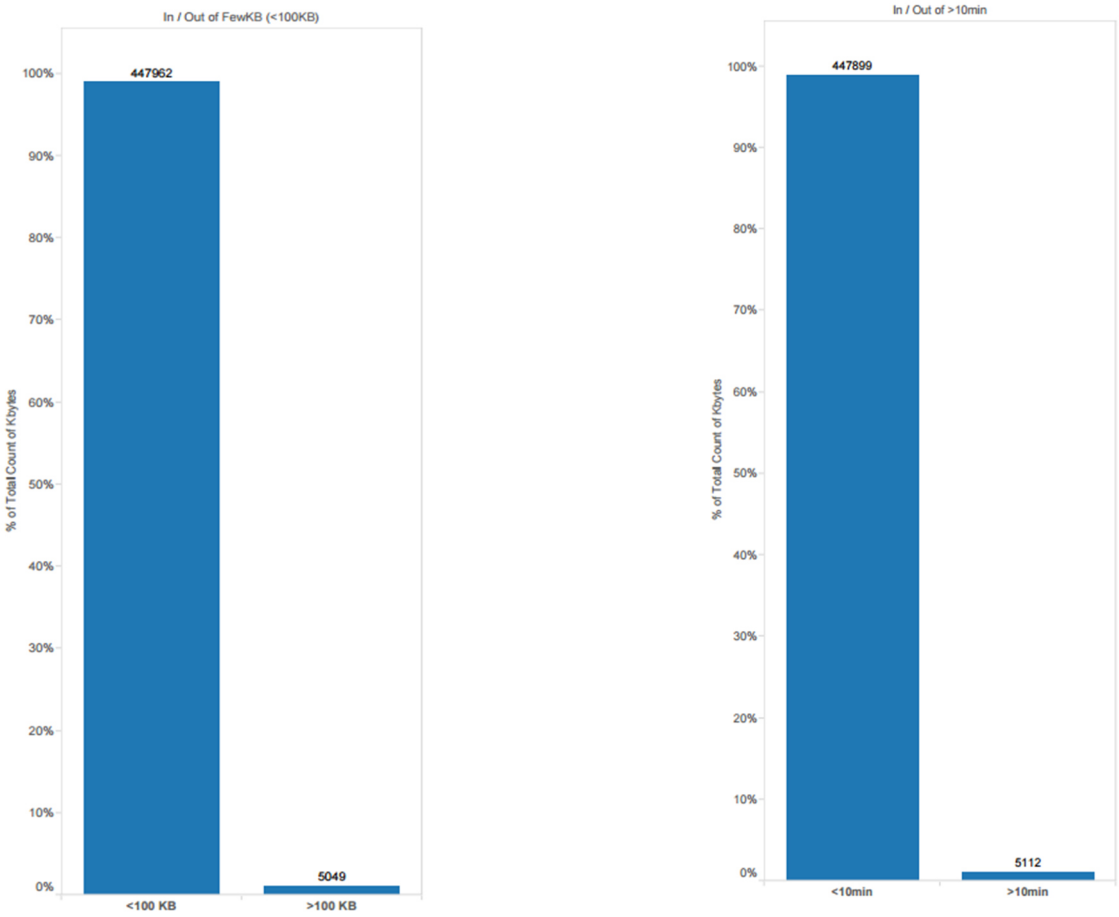


Figura 4.11 - Sessioni scadute - dati scambiati

Il dato, sebbene possa sembrare insolito, in realtà è completamente conforme alla logica di rete. Tra le sessioni scadute, infatti, come è stato già accennato, sono presenti anche i trasferimenti "fulminii", ossia i trasferimenti di breve durata interrotti nel giro di pochi minuti (i.e. controllo dello status di tutti i terminali con dei processi di ping). Ed ecco perché ne rappresentano la maggioranza.

In aggiunta, come si evince dalla figura 4.12a sotto riportata, delle sessioni totali soltanto circa 5mila hanno registrato un trasferimento di dati superiore a 100KB. Analogamente, nella figura 4.12b, con circa lo stesso rapporto con le totali, soltanto un numero di sessioni limitato ed equivalente a quello precedente (*sessioni coincidenti*) presenta un trasferimento della durata maggiore di 10 minuti.



(a) Kbyte scambiati.

(b) Durata totale.

Figura 4.12 – Statistiche sulle sessioni scadute.

Tramite lo strumento, è stato inoltre possibile, soltanto come *Proof of Concept*, individuare relazioni 1-N e N-1, tra gli indirizzi IP delle sessioni. Questo vuol dire che dato un indirizzo di rete è possibile analizzare quali sono i terminali con cui maggiormente quest'ultimo interfaccia sessioni di comunicazione (1-N) oppure, nel verso opposto, quanti indirizzi di rete comunicano, anche contemporaneamente, con un'unica macchina (N-1). La figura 4.13 ne riporta alcune prove effettuate sulle sessioni scadute storicizzate durante i test.

				DST IP				
SRC IP	137.204. .	137.204. .	10.100.250.249	224.0.0.252	137.204. .	137.204. .	137.204. .	123.30.183.98
137.204. .			31326					
69.164.195.45	18095							
64.235.150.189	16215							
137.204. .								
137.204. .				23				
137.204. .								6052
137.204. .				7381				
188.166.175.65	4672						2119	
74.207.233.253	6728							
64.62.228.217	4713							
137.204. .				161				
137.204. .				1553				
193.43.192.81	1782	1807						

Figura 4.13 - Relazioni 1-N e N-1 tra gli IP delle sessioni scadute.

Ovviamente, dai dati sono state escluse, proprio per agevolare l'analisi, le sessioni interne tra terminali di uso interno (es. laboratori, videocamere di sorveglianza). A maggior ragione, perciò, i dati risultano più che sensati.
Le eventuali segnalazioni di rete, se presenti, si celano in quel limitato numero di sessioni (5000) che presentano una durata significativa e una quantità di dati scambiati sostanziosa. Di seguito si riportano alcune delle situazioni "sospette", individuate anche grazie alla consulenza del centro di sicurezza, alle quali potrebbe essere integrata un'ulteriore analisi nel dettaglio.

Traffico troppo voluminoso

Tra i warning riscontrati tra le sessioni per così dire "pesanti", sono saltati all'occhio, in una fase di prima analisi, generalizzando, due diversi casi che si replicavano nei report:

(1) 137.204.*.* (indirizzo interno) - 91.197.44.11:80 (risolto come *rapidgator.net*)
Quantità di dati scambiata per sessione: >4GB.

(2) 137.204.*.*: 30741 (indirizzo interno) - 151.42.51.152:44 (indirizzo irrisolvibile)
Quantità di dati scambiata per sessione: >4GB.

Nel caso (1) si tratta sicuramente di un terminale che ha effettuato attività di elevato download di grandi file da host adibiti al filesharing. Il caso, per la sua natura, rappresenta una segnalazione innocua che non richiede, perciò, ulteriore approfondimento.

Il caso (2), al contrario, presenta una comunicazione di una macchina interna con un host appartenente ad un dominio *non risolvibile* mediante server *dns*. Il suo anonimato lascia intendere la presenza di un'attività in ogni caso sospetta (i.e. file trojan) per cui vale la pena continuare un'eventuale "indagine".

Il centro di sicurezza, infatti, dopo un'eventuale procedura di risoluzione dell'ip "sospetto" (nslookup), figura 4.14, interviene al fine di individuare la natura del problema. Gli indirizzi IP sono stati oscurati per una questione di privacy.

Figura 4.14 - Procedura di risoluzione degli indirizzi

Traffico di durata anomala

Anche tra le sessioni lunghe, sono stati riscontrati diversi casi che lasciavano spazio ad una possibile attività insolita. Se ne riportano tre situazioni.

(1) 137.204.78.90 (csi-printest.unibo.it) - 10.204.116.105:161 (stampante di rete)
Durata della sessione: >9h.

(2) 137.204.*.*(indirizzo interno) – 54.94.105.57 (risolto in *figura 4.18*)
Durata della sessione: >9h.

(3) 192.168.100.9:2593 (indirizzo locale) - 192.168.100.7:1416 (indirizzo locale)
Durata della sessione: >9h.

Il caso (1), fondamentalmente, è un vero e proprio warning "innocuo". Per il semplice motivo che si tratta di un *print-server* interno (IP 137.204.78.90) che contatta una stampante (IP 10.204.116.105) tramite protocollo *snmp*[15] (porta 161) per controllarne periodicamente il suo stato.

Il caso (2), presenta una particolare sessione di comunicazione tra una macchina interna e un indirizzo di rete pubblico (54.94.105.57), di cui si è riusciti a risalire, tramite un risolutore di indirizzi online, alla localizzazione del terminale. Quest'ultimo, come si evince dalla figura 4.15, è appoggiato ad un ISP sudamericano e desta non poco interesse. E' necessario, infatti, approfondire la situazione dati i valori rilevati.

Figura 4.15 - Risoluzione indirizzo IP di una sessione sospetta.

Il caso (3), presenta due IP, naturalmente non risolvibili perché riferiti a due macchine locali, che hanno effettuato una sessione di comunicazione oltre la soglia impostata. E' necessario capire, perciò, se l'attività è dovuta ad una particolare applicazione che giustifica questo warning oppure se è effettivamente riconducibile ad una procedura di intrusione non autorizzata.

[15] In informatica e telecomunicazioni Simple Network Management Protocol (SNMP) è un protocollo di rete che opera al livello applicativo del modello OSI e consente la configurazione, la gestione e la supervisione (monitoring) di apparati collegati in una rete.

5. Conclusioni e sviluppi futuri

Il fenomeno dei Big Data ha ormai preso il sopravvento, è sfida continua nel mondo aziendale cercare, e ottimizzare al massimo, sempre nuovi processi di analisi. Le informazioni, accumulate per anni e anni, che sembravano non aver nessun importanza effettiva si stanno velocemente rivalutando diventando protagoniste in questa nuova "era" del Data Management. Il progetto di tesi si muove proprio in questa nuova concezione della Business Intelligence, introducendo nuovi mezzi concreti per il supporto all'analisi, rivoluzionando la realtà gestionale.
Nel giro di tre mesi è stato possibile, dopo una prima fase di progettazione, rilasciare un primo prototipo conforme all'analisi dei requisiti che il dominio applicativo ha comportato. E' stato possibile far uso di tecnologie sostanzialmente nuove nella realtà informatica, dando modo di carpire al meglio cosa vuol dire non solo progettare un sistema per l'elaborazione di dati complesso e distribuito, ma approfondirne una sua specializzazione: il Data Stream Processing. Utilizzare sistemi innovativi, inoltre, partendo da un approccio, almeno iniziale, prettamente neofita, si è rivelato molto stimolante. Il progetto, infatti, sebbene ancora in una fase ultra-preliminare ha cominciato fin da subito a fornire, in fase di test, risultati abbastanza soddisfacenti. Con il supporto del centro di sicurezza, in non molti cicli di prime prove, infatti, con le dovute modifiche volta per volta aggiunte si è arrivati a riscontri positivi con l'individuazione di pattern di rete (*possibili attacchi, intrusioni*) in un primo momento solo teoricamente ipotizzati. Sicuramente l'applicazione, visti i tempi di sviluppo non troppo estesi, necessita ancora di rimodulazioni, azioni di manutenzione sia evolutiva che correttiva, fattorizzazioni e nuove valutazioni. Questo implica, tuttavia, un lavoro comunque non concluso, che potrebbe senza esitazione essere ripreso per portare alla produzione di uno strumento di grande utilità all'intero Campus.
Per come è stato ideato il progetto si muove in un dominio molto delicato protagonista oggi nel panorama dell'informatica. L'idea di far convergere due branche apparentemente concettualmente poco conciliabili come possono essere la sicurezza informatica e i sistemi elaborati di analisi dei dati, lascia intendere, pertanto, l'utilità in ogni possibile dominio dei sistemi Data Analytics. In sostanza, ovunque ci sia una quantità di informazioni mediamente elevata è probabilmente presente una possibilità di analisi e di estrapolazione di schematizzazioni e conclusioni concrete riutilizzabili nel campo decisionale e, nel caso del settore vendite, di marketing.
Storm, inoltre, si è rivelato un sistema per il real-time effettivamente intuitivo, molto potente prestazionalmente e ben congeniato. La sua intuitività permette di progettare in poco tempo sistemi complessi ma concettualmente ben divisi.

Anche la fase di test, disponibile sia in ambiente locale senza l'utilizzo obbligatorio di una piattaforma Cluster che, appunto, tramite l'accesso ad un sistema distribuito è semplificata. Data la sua natura ancora preliminare (versione 0.10.0) il sistema riceverà indubbiamente in futuro nuove evoluzioni che completeranno quelle funzioni in cui ancora presenta delle mancanze, in particolare nell'interfaccia di debugging, tuttora praticamente inesistente. Nel complesso la piattaforma, tuttavia, appartenente allo stack di Hadoop, si è presentata dal primo momento, come già detto, affidabile e oltremodo performante rendendo possibile l'ottenimento di risultati dal primo momento sensati e, soprattutto, interessanti e attinenti ai fini stabiliti in partenza con il centro di sicurezza del campus.
Sono presenti, naturalmente, tanti aspetti che il prototipo per ora non compre completamente. Come, per esempio, la gestione delle sessioni "non-sessioni", ossia quelle strutturate da protocolli di rete (i.e. *UDP*) che non permettono di individuare un vero e proprio concetto di interfacciamento continuo e pre-configurato come può essere appunto una sessione di comunicazione *TCP* (vedi capitolo 4*)*.
Tuttavia, in futuro, se il progetto avesse la possibilità di proseguire, si potrebbe espandere la struttura, vista anche la sua elevata scalabilità, in modo da risolvere tutte queste problematiche che solo un'analisi dei requisiti più approfondita e dei tempi di sviluppo e progettazione più estesi possono far emergere.
Ad oggi, la presentazione del primo prototipo può ritenersi, nel complesso, rispetto alle premesse poste in partenza con i "committenti" dell'ufficio di sicurezza informatica d'Ateneo, molto soddisfacente e da non abbandonare.

6. Bibliografia

[1] Pescatore F., La Storia dei Database, le origini, in appuntiDigitali.it, Luglio 2011.

[2] Gandomi A., Haider M., Beyond the hype: Big data concepts, methods, and analytics, in "International Journal of Information Management", December 2014.

[3] Schroeck M., Shockley R., Smart J., Romero-Morales D., Analytics: The real-world use of big data. How innovative enterprises extract value from uncertain data. IBM Institute for Business Value. June 2012.

[4] Torrey M., The 5 Vs of Big Data, Immoore-Oracle, October 2015.

[5] Dettagli JSON Object: json.org/json-it.html

[6] Cattel R., Roe C., Scalable SQL and NoSQL Data Stores, originally published 2010, last revised December 2011.

[7] Ranking DBMS aggiornato: db-engines.com.

[8] Roe C., 2016 Trends in NoSQL: Next Stages of Development, in Dataversity.net, January 2016.

[9] White T., Hadoop: The Definitive Guide, O'Relly Media Inc., 2009.

[10] High-level Architecture of Hadoop: opensource.com/life/14/8/intro-apache-hadoop-big-data

[11] Nardelli M., Distributed Data Stream Processing, 2015.

[12]Apache Software Foundation. Official Website Apache Storm: storm.apache.org

[13] Quercioli D., Storm and Surroundings, a brief introduction to Big Data stream processing, December 2015.

[14] Allen S., Jankowski M., Pathirana P., Storm applied, strategies for real-time event processing, Manning, 2015.

[15] Eisbruch G., Leibiusky J., Simonassi D., Getting Started with Storm, 1st Edition, O'Reilly Media Inc., 2012.

[16] Cordova P., Analysis of Real Time Stream Processing Systems considering Latency, 2015.

Ringraziamenti

Ringrazio il Prof. Matteo Golfarelli e il Dott. Lorenzo Baldacci che mi hanno seguito nel percorso di tesi ad ogni passo con grande professionalità, preparazione e pazienza. Un ringraziamento speciale va, inoltre, a tutto il team del BI Lab e al Dott. Ciro Barbone, responsabile del centro di sicurezza informatica del Campus, senza i quali l'intero progetto non sarebbe stato possibile.

Esprimo la mia più estesa gratitudine agli "…Aspiranti Ingegneri…", amici e colleghi preziosi fin dall'inizio di questo percorso; nella buona e nella cattiva sorte.

Un ringraziamento generale, infine, è destinato a chiunque abbia rasserenato e arricchito, anche soltanto per una volta, le mie giornate universitarie.