**Introduzione**

Oggi giorno uno dei mercati più floridi è quello della vendita di dati in quanto molte compagnie li utilizzano per effettuare studi e previsioni di mercato, analisi dei gusti degli utenti, ecc. I maggiori acquisitori di dati sono senza dubbio i social network, come Twitter, Facebook, Instagram, visto il loro utilizzo massivo da parte degli utenti di tutto il mondo. Tuttavia, la compagnia che riesce ad acquisire il maggior numero di dati è Google, in quanto negli anni è diventato il motore di ricerca per eccellenza. L’analisi di questi dati ha portato alla realizzazione di nuovi approcci per poterli gestire in modo veloce, cosa che non si sarebbe riuscita a fare con le normali tecniche, vista la loro grandezza e varietà. Il modo più efficiente ad oggi è quella di analizzare i dati attraverso l’elaborazione parallela che permette di gestire problemi di grandi dimensioni in tempi relativamente brevi. Le elaborazioni parallele sono implementate distribuendo la computazione su un grande numero di computer.

Il nostro elaborato mira ad evidenziare la potenza del calcolo parallelo distribuito su reti di calcolatori confrontandolo con un’elaborazione locale sia in maniera sequenziale, sia sfruttando i thread. Le varie elaborazioni andranno ad analizzare un dataset formato da un insieme di tweet estratti dal social network Twitter.

Nei successivi capitoli introduttivi andremo a spiegare i big data e le tecnologie hadoop e spark. Successivamente verranno presentati i vari passi che hanno portato alla realizzazione di questo progetto.

Big Data

I Big Data sono una raccolta di dati grande e complessa così estesa in termini di volume, velocità e varietà da richiedere tecnologie e metodi analitici specifici per l'estrazione di valore o conoscenza. Con i big data la mole dei dati è dell'ordine degli zettabyte, ovvero miliardi di terabyte, quindi si richiede una potenza di calcolo parallelo e massivo con strumenti dedicati eseguiti su decine, centinaia o anche migliaia di server.

Le caratteristiche fondamentali dei Big Data possono essere descritte nelle tre “V”:

* Volume: capacità di acquisire, memorizzare ed accedere a dati di grandi dimensioni;
* Velocità: capacità di analizzare i dati in tempo reale o quasi;
* Varietà: riferito alle varie tipologie di dati che possono essere strutturate o non strutturate.

Hadoop

Apache Hadoop è un framework sotto una licenza libera che supporta applicazioni distribuite con elevato accesso ai dati, permettendo alle applicazioni di lavorare con migliaia di nodi e petabyte di dati. Ispirato dalla MapReduce di Google e dal Google File System, è un progetto Apache di alto livello costruito e usato da una comunità globale di contributori, che usano i linguaggi di programmazione Java e Scala.

Hadoop è composto da HDFS, MapReduce e YARN. Col passare del tempo si è costruito un ecosistema aggiungendo nuovi strumenti progettati appositamente per integrarsi con questi tre.

- HDFS

L'Hadoop Distributed File System (in sigla HDFS) è un file system distribuito, portabile e scalabile scritto in Java. Un cluster in Hadoop tipicamente possiede uno o più name node (su cui risiedono i metadati dei file) e un insieme di data node (su cui risiedono, in blocchi di dimensione fissa, i file dell'HDFS).

I file in HDFS sono suddivisi in blocchi e distribuiti tra i vari nodi del cluster mantenendo anche

delle repliche per garantire maggiore sicurezza.

Hadoop cerca di far eseguire a ciascun nodo i calcoli sui blocchi di file presenti nel proprio disco, in

questo modo si suddivide il carico di lavoro aumentando le prestazioni di calcolo e diminuendo

traffico di rete.

L’HDFS non permette operazioni di modifica su file, ma solo di creazione e cancellazione.

-MapReduce

Il MapReduce è stato ispirato dalle funzioni Map e Reduce utilizzate nella programmazione funzionale. Nella fase di Map i dati di input sono una serie di record che vengono elaborati singolarmente (da processi processi chiamati Mapper) in modo da restituire risultati che verranno successivamente aggregati (da processi chiamati Reducer).

Ogni singolo nodo possiede un blocco del file su cui esegue la sua parte di elaborazione.

-YARN

Introdotto in hadoop 2.0, YARN si impegna ad allocare le risorse a varie applicazioni in modo efficace. Esegue due demoni, che si occupano di due diverse attività:

* il gestore delle risorse, che esegue il monitoraggio dei lavori e l'allocazione delle risorse alle applicazioni
* il master dell'applicazione, che monitora lo stato di avanzamento dell'esecuzione.

Spark

Apache Spark è una piattaforma open source per l’elaborazione di analisi dei dati su larga scala, progettata per essere veloce e generica, tuttavia non include un sistema di gestione dei dati e pertanto viene solitamente distribuito su Hadoop o su altre piattaforme di archiviazione.

La sua caratteristica più importante è il suo cluster computing in memoria che è responsabile di aumentare la velocità di elaborazione dei dati.

Spark sfrutta il concetto di RDD, Resilient Distributed Dataset, che rappresenta una collezione immutabile e partizionata di dati su cui è possibile operare in parallelo.

Le proprietà chiave di un RDD sono:

* Ogni RDD è immutabile, cioè una volta creato non lo si può cambiare, se non creandone un altro mediante una trasformazione;
* Ogni RDD può solo essere creato inizialmente a partire dai dati su disco (presi da HDFS) oppure a partire da altri RDD;
* Le trasformazioni possibili per creare nuovi RDD sono poche, deterministiche e ripetibili: si può mapparli (cioè trasformarli da un array di chiave-valore a un altro array di chiave-valore), filtrarli (partire da un array e crearne un altro filtrando i dati), unire due RDD;
* Questo approccio serve perchè un singolo pezzo (partizione) del RDD possa essere ricostruito a partire dalla sequenza di trasformazioni che lo hanno generato;
* Ogni RDD può restare in memoria oppure essere materializzato su disco, a scelta del programmatore (RDD in memoria ma inutilizzati da tempo vengono comunque automaticamente scaricati su disco dal processo di gestione che gira sul worker node);
* Ogni RDD è descritto da un set completo di metadati che consentono la ricostruzione di una delle sue partizioni in caso di fault: dove si trovano le partizioni, quali sono gli RDD padre, quale è la sequenza di trasformazioni, detta LINEAGE, che lo hanno generato.

Spark nasce come un sistema per creare e gestire job di analisi basati su trasformazioni di RDD. Dato che gli RDD nascono e vivono in memoria, l’esecuzione di lavori iterativi o che trasformano più volte un set di dati, sono immensamente più rapide di una sequanza di map-reduce. 10, anche 100 volte più veloci, perchè il disco non viene mai (o quasi mai) impiegato nell’elaborazione.

Un’altra caratteristica è che supporta più linguaggi fornendo API scritte in Scala, Java, Python o R rendendolo così molto flessibile.

**Progetto**

Scopo del progetto

Il sentiment analysis è un campo dell'elaborazione del linguaggio naturale che si occupa di costruire sistemi per l'identificazione ed estrazione di opinioni dal testo. Si basa sui principali metodi di linguistica computazionale e di analisi testuale. Viene utilizzato in molteplici settori, ma il suo campo di applicazione principale è quello legato al marketing.

Il nostro progetto ha lo scopo di analizzare il “sentiment”, in particolare associare un tipo di emozione ad un testo, di un tweet. Per fare ciò ci siamo avvalsi di una libreria, opportunamente modificata, chiamata TextToEmotion situata su GitHub (link).

La libreria permette di distinguere 7 tipi di emozioni:

1. Neutral
2. Happiness
3. Sadness
4. Fear
5. Anger
6. Disgust
7. Surprise

Essa permette l’elaborazione del testo, attraverso un singolo metodo statico, estraendo le singole parole ed utilizzando un dizionario in cui ogni vocabolo ha un peso associato ad un’emozione. I risultati di ogni parola vengono poi raggruppati e trasformati in percentuale. Nello stesso modo, vengono valutate anche le emoticon presenti nel testo. L’emozione risultante sarà quindi quella a cui è associato il valore più alto.

Visto l’utilizzo del dizionario in cui sono presenti solo termini in lingua inglese, non è stato possibile estrarre il sentiment da testi in altre lingue. Tuttavia sviluppi futuri possono portare alla costruzione di ulteriori dizionari in modo da coprire un più ampio range di lingue.

Studio del dataset

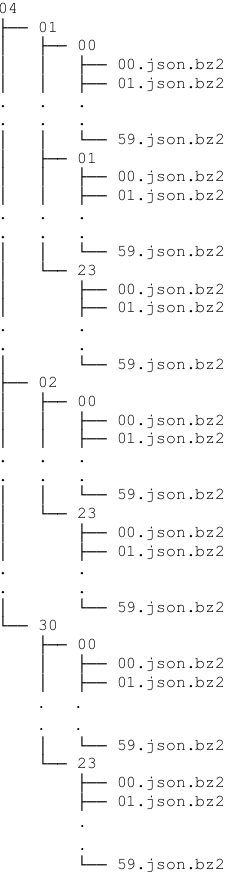
Il social media Twitter è un servizio di notizie e microblogging creato nel Marzo 2006. Si stima che ad oggi abbia più di 300 milioni di utenti che generano 65 milioni di tweet al giorno e gestisca più di 800.000 richieste di ricerca giornaliere. La rete consente di postare messaggi di testo di breve lunghezza, con un massimo di 280 caratteri, chiamati tweet, che vengono visualizzati nella pagina principale dell'utente. In questi 280 caratteri l’utente riesce ad esprimere opinioni ed emozioni condividendoli con la sua rete di follower.

L’azienda ha messo a disposizione degli sviluppatori delle API che consentono di interagire con l’intera piattaforma. Grazie a queste API è possibile reperire un gran numero di dati da poter analizzare.

Per il nostro progetto è stato utilizzato un dataset compresso in formato tar di circa 49GB (link). All’interno del dataset troviamo un’organizzazione gerarchica ben definita di file e cartelle:

1. Una cartella principale 04
2. 30 cartelle di secondo livello
3. Ogni cartella di secondo livello contiene 24 cartelle di terzo livello
4. Ogni cartella di terzo livello dovrebbe contenere 60 file in formato *.bz2*

La gerarchia può essere riassunta in modo semplice dal seguente schema



Per estrarre il dataset in formato *.tar* è stato utilizzato il seguente script:



La grandezza di un file *.bz2* è in media di 1MB. In ogni file compresso è presente un json, della dimensione di circa 10MB, contente una lista di oggetti, ognuno rappresentante un Tweet.

La struttura di un file JSON è simile a quella mostrata in figura:



Per estrarre tutti i file *.bz2* è stato utilizzato il seguente script:



Lo script individua tutti i file *.bz2* contenuti in una cartella principale, includendo anche le sotto-directory. Ogni file viene decompresso, assegnandogli un nuovo nome, costituito dal suo path a partire dalla cartella di base. È stato necessario rinominare i file in quanto molti file nelle sotto-directory hanno lo stesso nome.

All’estrazione completa del dataset, la sua dimensione è aumentata di circa 10 volte, arrivando a circa 460GB. Sarebbe stato impensabile quindi lavorare con un dataset così vasto formato da tanti piccoli file, in quanto avrebbe portato ad un’enorme spreco di memoria e lunghi tempi di caricamento di questi dati nell’HDFS. Il consumo di memoria si sarebbe verificato in quanto le dimensioni dei blocchi presenti sull’HDFS sono molto maggiori rispetto al singolo file JSON.

Per l’elaborazione si è scelto di utilizzare file compressi .bz2 in modo da contenere le dimensioni del dataset. Questo tipo di compressione resta comunque splittabile in Hadoop.

Abbiamo ridotto poi ulteriormente la taglia del dataset andando a filtrare tutti quei Tweet che non sono in lingua inglese, in quanto la libreria utilizzata non è pensata per analizzare più lingue.

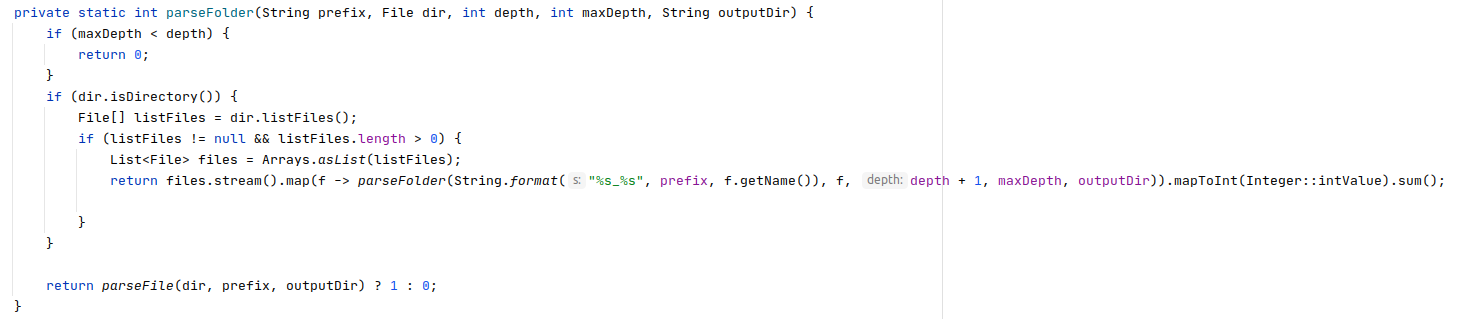
Prima di procede al filtraggio, attraverso un applicativo Java, abbiamo conteggiato sia il numero totale che il numero di Tweet in lingua inglese.

* Tweet totali: 157.158.045
* Tweet inglese: 44.015.065
* Tweet restanti: 113.142.980

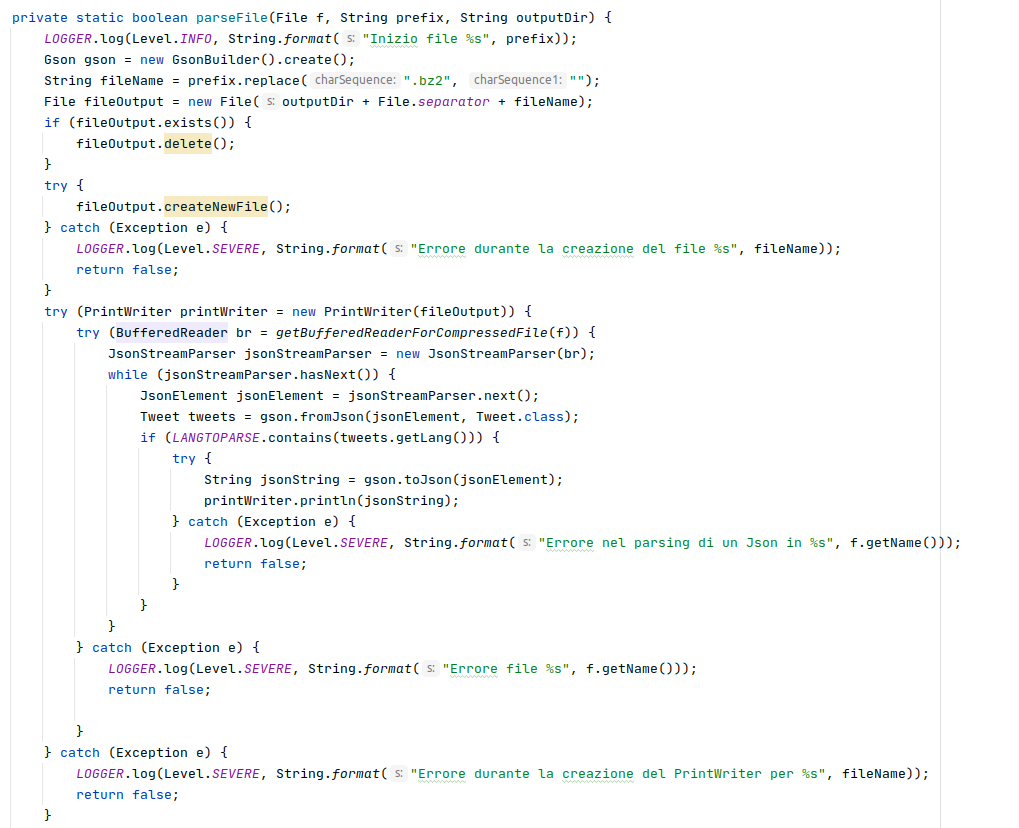
Per poter isolare i Tweet in lingua inglese, abbiamo costruito un applicativo Java che individua ricorsivamente, partendo da un path, i file in formato *.bz2.*

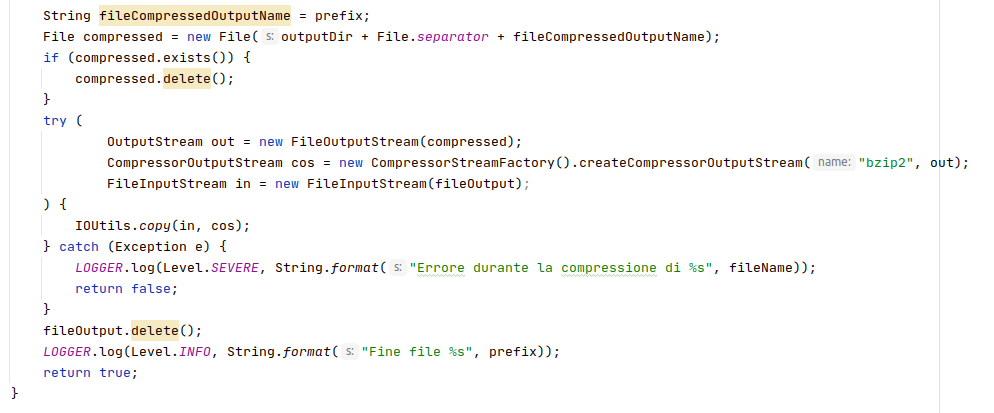
Ogni file individuato viene decompresso e filtrato. Infine il risultato viene salvato sempre in formato *.bz2* in uno specifico path.

Nell’immagine successiva è mostrato il metodo ricorsivo che analizza ogni cartella. Il valore di ritorno indica il numero di file analizzati.



Nel caso in cui venga passato un file come parametro, questo viene filtrato attraverso il metodo *parseFile*.



  
Dopo aver eseguito questo applicativo, il risultato è stato quello di avere 43.179 file *.bz2* contenenti solo Tweet in lingua inglese.

La dimensione tuttavia si è ridotta ad un dataset compresso di 14.6GB.

Per questo motivo abbiamo sviluppato uno script che ci permette di unire diversi file *.bz2* in blocchi da *n* MegaByte, in modo da permetterci di modificare  
le dimensioni dei file da caricare sull’HDFS in base alla grandezza dei blocchi.



Lo script va semplicemente ad unire i file successivi finché non raggiunge il numero di Byte prestabilito. Per semplicità e questioni di velocità abbiamo deciso di non utilizzare algoritmi specifici per la costruzione di blocchi in maniera precisa.

Approcci

Per valutare la soluzione ottima abbiamo costruito tre approcci:

* Sequenziale
* Thread
* Spark

A supporto dell’elaborazione abbiamo costruito un POJO che rappresenta l’oggetto JSON, chiamato Tweet, contenente due proprietà, *text* e *lang.*

È stata implementata una classe chiamata *MapCounter* che attraverso un metodo *parseText(String)* elabora il testo attraverso la libreria TextToEmotion e mantiene un contatore per ogni emozione individuata salvando i risultati in un *HashMap*.

Per gli approcci Sequenziale e Thread, sono stati inserite delle istruzioni che permettono di conoscere il tempo di inizio e di fine d’esecuzione del programma. Inoltre vengono salvati eventuali errori durante le aperture dei file o durante il parsing del testo. Queste informazioni vengono stampate nei log alla fine dell’elaborazione.

Durante l’elaborazione, questi due approcci, forniscono informazioni sullo stato di avanzamento attraverso un logger.

Sequenziale

Il nostro programma sequenziale prende in input una cartella contenente uno o più file in formato *.bz2*. Ogni file viene decompresso in memoria in un *BufferedReader, che viene poi* utilizzato dalla classe *JsonStreamParser* per iterare su tutti gli oggetti JSON presenti nel file.

Ogni oggetto JSON viene trasformato in una istanza della classe *Tweet* e nel caso in cui il testo sia in lingua inglese la sua proprietà *text* viene utilizzata come argomento per il metodo *parseText* della classe *MapCounter.*

Alla fine delle iterazioni, vengono stampate le emozioni e il numero di volte che queste sono state individuate.

Nell’immagine seguente, viene mostrato l’interno dell’iterazione principale:



Thread

L’approccio Thread prende spunto da quello sequenziale. È stata costruita una classe chiamata *TweetThread* che estende la classe *Thread*. La classe viene istanziata passando come argomenti l’oggetto *MapCounter* e una lista di *File*, oltre ad un nome e ad una serie di variabili utili per salvare il numero di successi e di errori nell’apertura dei file e nel parsing del testo.

Nel metodo *run()* un thread itera sulla sua sequenza di file e li elabora secondo l’approccio del programma Sequenziale.

  
  
I thread vengono avviati da una classe *Main* che prende in input la cartella contenente i file in formato *.bz2* e un intero che indica il numero di thread da avviare per eseguire l’elaborazione (minimo due). La lista dei file viene quindi divisa in parti uguali tra i thread, che vengono lanciati poi in sequenza.

Il programma termina quando tutti i thread hanno finito l’elaborazione.



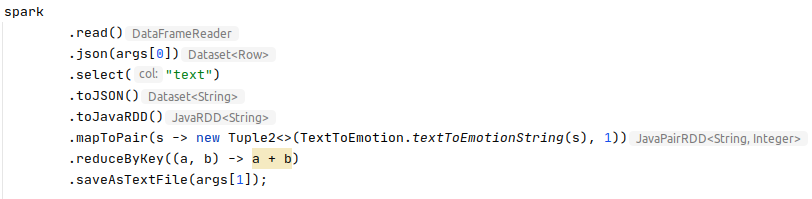
Spark

Il programma Spark prende in input due argomenti, il path dell’HDFS dove sono contenuti i file in formato *.bz2* e il path di output.

Dopo aver istanziato la *SparkSession* i file in input vengono letti e trasformati in *Dataset* in maniera trasparente all’utente. Attraverso la funzione *mapToPair* ad ogni testo viene associata una tupla contenente il valore dell’emozione e il counter 1.

Questo RDD viene poi ridotto attraverso la funzione *reduceByKey* che raggruppa i valori in base alla chiave.

Il risultato infine viene salvato nel path di output passato come argomento.



**Testing**

In questa sezione, dopo aver illustrato le configurazioni delle macchine, vengono mostrati i risultati ottenuti attraverso i vari approcci.

Configurazione computer

I test degli approcci Sequenziale e Thread sono stati lanciati su due differenti macchine. Di seguito vengono riportati i dettagli:

* Computer 1
  + Intel i7-4700HQ, 4-core 8-thread 2.4GHz – 3.4GHz con 6MB di cache
  + 8GB RAM LPDDR3 1600 MHz
  + Disco Samsung 2.5 pollici 1TB 5400 RPM SATA 6Gbps 8MB Cache
* Computer 2
  + Intel i7-6500U, 2-core 4-thread 2,5GHz – 3.1GHz con 4MB di cache
  + 8GB RAM DDR3
  + Disco Hitaci Travelstar 2.5 pollici 1TB 5400RPM SATA 6Gbps 8MB Cache

Su entrambe le macchine è stato installato il sistema operativo Ubuntu nella versione 20.04.01.

Configurazione cluster

Le esecuzioni dell’applicativo Spark sono state eseguite su un cluster di 8 nodi, dove ogni nodo ha le seguenti caratteristiche:

* CPU: 2,70GHz con 8 core
* RAM: 32 GB
* Disco: circa 209GB
* Linux 4.4.0-193-generic (x86\_64)

La versione di Spark eseguita è la 2.3.2, mentre quella di Hadoop è la 3.1.1.

L’HDFS è configurato per conservare i file in blocchi da 128MB.

YARN è stato configurato in modo da poter assegnare ad ogni nodo 7 core e 30GB di RAM. Sul cluster è sempre in esecuzioni il servizio HBase che occupa 4 core e 11GB di RAM divisi tra 3 nodi.

Risultati Sequenziale e Thread

L’esecuzione del programma Sequenziale e Thread sono state testate, per entrambe le configurazioni, utilizzando un dataset composto da 115 file *.bz2* da circa 128Mb ognuno.

Nella seguente tabella sono riportati i risultati del programma Sequenziale per entrambe le configurazioni di computer:

|  |  |  |
| --- | --- | --- |
|  | Computer 1 | Computer 2 |
| 128Mb | 6 ore, 18 minuti, 37 secondi | 5 ore, 41 minuti, 7 secondi |

Il programma Thread è stato valutato utilizzando 4 thread.

Nella seguente tabella sono riportati i risultati per entrambe le configurazioni di computer:

|  |  |  |
| --- | --- | --- |
|  | Computer 1 | Computer 2 |
| 128Mb – 4 Thread | 4 ore, 38 minuti | 3 ore, 58 minuti, 20 secondi |

Come è possibile notare dalle tabelle utilizzando la versione del programma multi-thread si ha un vantaggio in termini di tempo, in quanto si riescono a ridurre i tempi di attesa per la fine.

Per tutte le esecuzioni, c’è stato un forte consumo di CPU, soprattutto nel multi-thread, dove tutti i core avevano un carico che variava tra il 95% e il 99%. Per quanto riguarda la memoria RAM, non abbiamo notato nessun utilizzo anomalo.

Risultati Spark

La prima esecuzione dell’applicazione sul cluster è stata avviata con una configurazione di 8 executors, 7 cores (per executor) e 21 GB di RAM (per executor). Il tempo impiegato è stato di 20 minuti.

La minima qunatità RAM disponibile su un nodo era di 24GB a causa del servizio di HBase. Spark prende, rispetto alla RAM assegnata agli executor, il 10% di overhead abbiamo scelto 21GB in quanto 21GB+2,1GB=23,1GB.

YARN è configurato con 8 nodi da 7 core l’uno abbiamo deciso di assegnare a Spark 8 executors da 7 cores.

La nostra applicazione Spark, viene eseguita in 4 stage. Il primo stage (stage 0) consiste nell’individuare le partizioni su cui operare e assegnarle ai vari executors.