Elaborato Finale PC: LambdaArchitecture

Bernardo Tiezzi
bernardo.tiezzi@stud.unifi.it

Abstract

L'elaborato presentato è stato realizzato come elaborato finale per l'esame di Parallel Computing, appartenente al corso di Laurea Magistrale in Ingegneria Informatica dell'Università degli studi di Firenze. Il lavoro consiste nella definizione di una Lambda Architecture che scarichi dei tweets, utilizzando le Twitter Api, ed esegua una Sentiment Analysis su di essi. Il codice è realizzato in linguaggio Java 8. I risultati forniti consentono di comprendere i benefici offerti dall'impiego di questa architettura, designata per gestire grandi quantità di dati.

1. Introduzione

L'elaborato illustra l'implemenrazione di una **Lambda Architecture** che effettua il download di tweets dal social network **Twitter**, eseguendo al contempo memorizzazione ed "analisi del sentimento" dei dati ottenuti. Una Lambda Architecture è un'architettura *data-processing* designata per gestire una significativa quantità di dati immutabili, i quali crescono con continuità nel tempo. Un sistema software così realizzato può essere impiegato quando è necessario definire un'applicazione che, oltre a processare i dati a sua disposizione, è in grado di fornire in real-time ulteriori dati in ingresso al proprio database. I vantaggi forniti dall'archiettura in questione sono dovuti alla suddivisione in livelli dei processi di lavorazione. Il livello di batch processing, o **batch layer**, ha il compito di processare i dati contenuti all'interno del database dell'applicazione; è in grado di eseguire un'analisi dettagliata per grandi quantità di dati. L'altro livello è noto come **speed layer**, che elabora flussi di dati real-time, forniti successivamente in ingresso al database. Quest'ultimo si differenzia dal livello di batch per un processing dei dati senza requisiti di correzione o completezza: viene quindi sacrificato il throughput in favore di una forte riduzione dei ritardi. Le informazioni aggiunte al database dallo speed layer vengono rielaborate nel successivo ciclo di batch. Le viste (*views*) presentate in uscita da entrambe le fasi di processamento sono memorizzate da un livello ulteriore, noto come **serving layer**, che è in grado di restituire viste ad-hoc o crearne di nuove a partire dai dati elaborati. Dalla breve descrizione fornita è facile notare come questa particolare architettura si adatti perfettamente alla realizzazione di un'applicazione per l'analisi di tweets, in cui si rende necessaria la gestione di grandi quantità di dati in costante crescita nel tempo. Il comando dell'elaborato richiedeva di eseguire una **sentiment analysis** sui testi dei tweets raccolti: ad ogni tweet viene assegnata una *label* intera che identifica il testo contenuto nel tweet generato come espressione di sentimento negativo ($label < 0$), neutro ($label = 0$) o positivo ($label > 0$). Per semplicità, nel caso di studio descritto, la classificazione avviene in base a tre differenti valori della label:

- $label = 4$: tweet positivo.
- $label = 2$: tweet neutro.
- $label = 0$: tweet negativo.

Per la valutazione è stato opportunamente addestrato un algoritmo di machine learning LingePipe. I dati ottenuti durante l'esecuzione

dell'applicazione sono raccolti in base ad una *query* di riferimento, che può essere un hashtag o una keyword contenuta all'interno dei tweets. Per fornire una visione delle prestazioni ottenute dal sistema software implementato viene riportata, in forma grafica, la quantità di dati processati al termine dei cicli d'esecuzione.

2. Implementazione

Per realizzare un'applicazione che presenti la struttura di una lambda architecture è necessario implementare i vari livelli che la compongono: batch layer, speed layer e serving layer. Per il progetto realizzato la scelta sui sistemi software da utilizzare è ricaduta su:

- **Apache Hadoop**: framework che supporta applicazioni distribuite che consentono di processare quantitativi di dati dell'ordine del *petabyte*; viene spesso utilizzato per l'elaborazione dati in modalità batch. Il batch layer è stato definito sfruttando le librerie fornite da Hadoop.
- **Apache Storm**: è un sistema di calcolo distribuito in real time per la gestione di grandi quantità di dati ad alta velocità; la combinazione di Storm, con altre applicazioni, ed Hadoop può essere impiegata in vari ambiti, come la gestione della sicurezza in ambienti finanziari e commerciali. All'interno del progetto, Storm viene utilizzato per realizzare lo speed layer.
- **Apache Hbase**: è un file system di database NoSQL distribuito ed orientato alle colonne; è stato costruito per immagazzinare grandi quantità di dati salvati direttamente sull'hardware e fornisce l'accesso in lettura e scrittura. Essendo orientato alle colonne, Hbase sfrutta le righe delle tabelle per ordinare i dati ed ogni riga è una raccolta di famiglie di colonne. Ogni famiglia è costituita da un insieme di colonne identificate attraverso una coppia chiave-valore. Questo software permette di memorizzare i dati gestiti dai livelli batch e speed, i quali sono in grado di comunicare tra loro attraverso le informazioni disponibili sul database; è un'ottima scelta per svolgere le funzioni del serving layer.

Il progetto viene eseguito dopo aver installato i sistemi software elencati in modalità pseudo distribuita. É necessario esegure l'installazione e la configurazione dei file `.xml` prima di procedere alla scrittura del codice sorgente.

2.1. Prerequisiti: Installazione e Configurazione

Le versioni di Apache Hadoop, Storm ed HBase sono rispettivamente 2.10.0, 2.2.0, 2.2.0. La modalità pseudo distribuita è anche conosciuta come un single-node cluster, dove ogni componente risiede sulla stessa macchina. Il cluster di Hadoop è costituito da nodi differenti che si occupano di organizzare gerarchicamente i file HDFS, file system distribuito su cui si basa la struttura di Hadoop stesso. Il **NameNode** è un'applicazione eseguita sul server principale che gestisce sia il file system che il namespace, ovvero l'elenco di blocchi e dei file appartenenti al sistema. La dimensione di ciascun file è di 64/128 MB. É quindi richiesta una memoria fisica di almeno 16 GB per non riscontrare problemi durante l'esecuzione. I **DataNode** sono applicazioni che si trovano su altri nodi del cluster e gestiscono fisicamente lo storage dei dati. Inoltre eseguono le operazioni di lettura e scrittura richieste dal client. Il **SecondaryNameNode** è un servizio aggiuntivo che supporta il NameNode nel suo lavoro di amministratore. Ogni nodo del cluster, per il caso d'uso considerato, è allocato sulla stessa macchina. La configurazione di Hadoop si concentra su 3 file principali, letti durante l'avvio del programma:

- `core-site.xml`: contiene le informazioni generali, quali le informazioni del file system e le cartelle dei checkpoint.
- `hdfs-site.xml`: contiene le informazioni su dove è allocata la tabella dei nomi, sull'utilizzo della security e sulla posizione del namenode secondario.

- `mapre-site.xml`: contiene le informazioni relative al MapReduce, come la directory di sistema e l'indirizzo del JobTraker.

MapReduce è un framework per il calcolo parallelo distribuito in grado di processare grandi quantità di dati. A differenza della programmazione multi-threading, in cui i dati vengono condivisi tra i vari threads aumentando la complessità dell'esecuzione, MapReduce rimuove la condivisione dei dati che vengono passati come parametri o valori di ritorno. Il **JobTraker** si occupa di gestire l'allocazione delle risorse di sistema (CPU e memoria) per ogni Job MapReduce, la cui esecuzione verrà spiegata in seguito. Il TaskTraker è colui che effettivamente esegue i task sotto la direzione del JobTraker, che si occupa anche di monitorare e ripetere l'esecuzione se insorgono errori. Nelle versioni 2.x di Hadoop, il JobTraker viene sostituito da due componenti, ResourseManager e ApplicationMaster, che svolgono le funzioni di assegnazione delle risorse e monitoraggio della singola applicazione: per ciascun nodo è presente un NodeManager che gestisce i processi. Il software aggiuntivo responsabile della modifica è lo **YARN** (Yet-Another-Resource-Negotiator), il quale presenta una struttura più generica rispetto al JobTracker. Il file di configurazione per lo YARN è noto come `yarn-site.xml`.

La configurazione psuedo distribuita di HBase prevede che ogni *deamom* sia eseguito singolarmente sullo stesso server. I file a cui apportare le dovute modifiche sono `hbase-site.xml` e *hbase-env.sh*. Tra le componenti fondamentali per il mantenimento dei cluster allocati è incluso **Apache Zookeeper**. Questo strumento di gestione garantisce consistenza sequenziale nella notifica degli aggiornamenti richiesti dal client, affidabilità ed atomicità delle operazioni eseguite. Inoltre Zookeeper garantisce la sincronizzazione degli oggetti in comune tra i nodi del cluster. Nella configurazione del file .xml è necessario definire una directory su cui salvare i dati generati da HBase e Zookeeper.

Per definire un cluster di Apache Storm è necessario aggiornare il file `storm.yaml` che configura i deamons di Storm (Nimbus, Supervisor, Zookeeper). Il nodo **Nimbus**, in maniera analoga al JobTracker di Hadoop, esegue un caricamento delle operazioni da eseguire distribuendo i calcoli ed avviando i processi; durante l'esecuzione monitora e rialloca eventualmente i processi. I nodi **Supervisor** (singolo nel caso descritto) comunicano con il Nimbus tramite il nodo Zookeeper avviando ed arrestando i processi worker in base agli ordini ricevuti.

2.2. Architettura dell'applicazione

L'architettura dell'applicazione realizzata è mostrata in figura 1. Il suo funzionamento prevede che il batch layer acceda alla tabella Hbase contenente i tweets da valutare, esegua uno scorrimento sulle righe della tabella assegnata, filtri i tweets in base alla **query** ed esegua una sentiment analysis sui tweets. Dopo aver completato la valutazione, ogni tweets viene reinserito nella tabella del serving layer di riferimento. Per ogni query viene eseguito l'aggiornamento della tabella relativa alla classificazione dei tweets (Positive, Negative, Neutral). Lo speed layer si occupa di eseguire il download di tweets relativi alle queries selezionate, ne esegue una sua valutazione ed aggiorna le tabelle Hbase per consentire al batch layer di processare i nuovi dati al ciclo successivo aggiornando il client riguardo le nuove valutazioni eseguite. Il serving layer raggruppa i dati in tabelle ed in caso di richiesta fornisce in output i dati raccolti. I tweets contenuti nel tabella del serving layer apriori sono forniti da un opportuno dataset allocato in una tabella **derbySQL**. Un ruolo cruciale è svolto dal `timestamp` che consente al batch layer di non processare i tweets aggiunti alla tabella dallo speed layer nel ciclo corrente.

2.3. Batch Layer

Il batch layer è implementato utilizzando le librerie di Hadoop. MapReduce, su cui Hadoop si basa, sfrutta il principio *divide et impera*, suddivi-

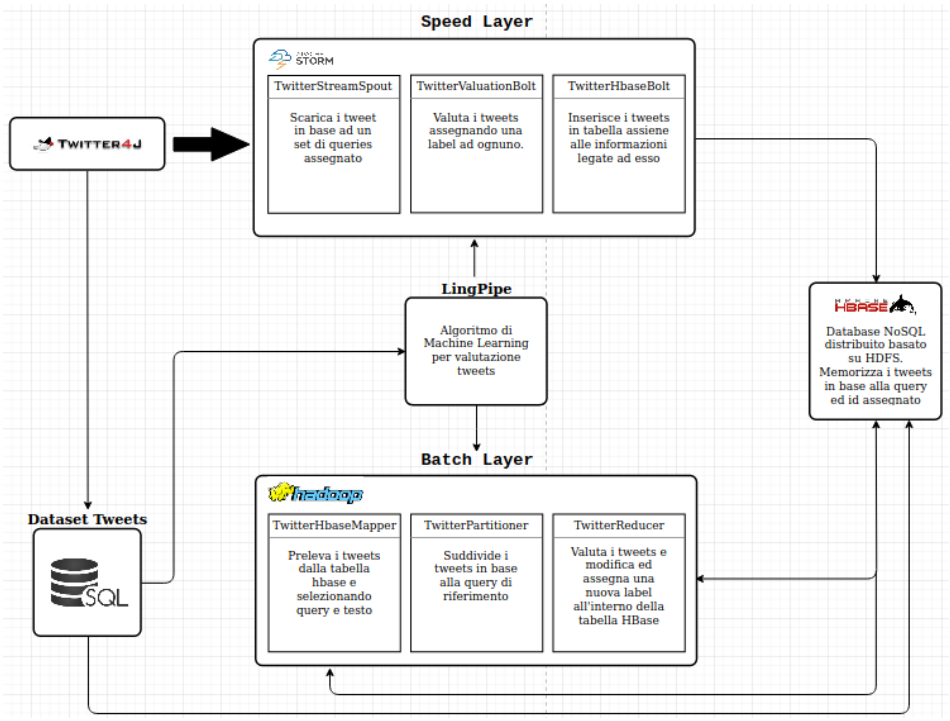

Figure 1. Diagramma Lambda Architecture

dendo l'operazione di calcolo in più parti, processate in modo autonomo. Una volta che ciascuna parte del problema è stata calcolata, i risultati parziali vengono ricomposti in un unico risultato finale. Il monitoraggio e l'eventuale ripetizione dei processi dipende da MapReduce stesso. Il compito del livello di batch è quello di processare i tweets inseriti all'interno di una tabella HBase, opportunamente generata a priori, eseguendo una sentiment analysis del testo di ogni tweet ed aggiornando nuovamente la tabella con il *sentiment value* aggiornato. Nello specifico sono state definite 4 moduli software differenti:

- **TwitterJob**: si occupa di inizializzare il job MapReduce allocando, se necessario, le tabelle di HBase. Definisce il ciclo di batch successivo assegnando al Mapper le righe da scorrere in base all'ultima riga inserita dallo speed layer. La classe `Configuration`, definita in fase di inizializzazione, consente di impostare le proprietà necessarie al job per essere eseguito in modalità distribuita, direttamente dall'IDE Netbeans. Il numero di reducer da eseguire in parallelo viene settato tramite il comando `job.setNumReduceTasks(int num)`: conviene definire un numero di reducer pari alle queries da processare. In questo modo ogni reducer esegue operazioni su tweets contenenti una query specifica.
- **TableHbaseMapper**: estende la classe TableMapper, che riceve in ingresso un oggetto di tipo `Result`, il quale fornisce l'accesso ad una riga della tabella HBase. Per ogni valore di ingresso restituisce una coppia <Key,Value>, dove Key è la query a cui il tweet è legato, mentre Value è il testo del tweet da valutare.
- **TwitterPartitioner**: è la classe Partitioner che si occupa di raggruppare le coppie <Key, Value>, in base al valore della Key ed assegnarle ad uno specifico reducer.
- **TwitterReducer**: riceve in ingresso i tweets relativi alla query assegnata ed esegue la sentiment analysis. Per la valutazione è

necessario che, prima di avviare il processo di *reduce*, sia allocato come attributo della classe l'algoritmo **LingPipe** che sarà addestrato su un dataset contenuto nel database derbySQL. Dopo aver ricavato il sentiment value di un tweet viene eseguita un'operazione di `Put()` sulla tabella Hbase, aggiornando il valore del campo *Label*. Per ogni query, su un'apposita struttura dati, viene salvato il numero di tweets processati, in base al classe di appartenenza (Positive, Negative, Neutral). Al termine del ciclo di batch, viene eseguito un aggiornamento della tabella `SentimentAnalysisTweets`, contenente il numero complessivo di dati processati suddivisi in base alla `Query` e alla classe `Sentiment` di appartenenza.

2.4. Speed Layer

Il framework adatto alla realizzazione dello speed layer è Apache Storm. Questo livello ha il compito di eseguire il download dei tweets online, sfruttando le **Twitter4j Api**, librerie di Twitter scritte in linguaggio Java. Il filtraggio dei tweets viene eseguito sulla base delle queries scelte. I dati vengono processati in *realtime*: una volta scaricato un tweet viene eseguita la sentiment analysis per ricavarne il sentiment value, in modo da poterlo classificare come positivo, negativo o neutro. Infine lo speed layer deve aggiungere alla tabella di batch i nuovi tweets elaborati, in modo da consentire l'esecuzione del modulo Hadoop durante il ciclo di batch successivo. Per lo svolgimento dei compiti assegnati, viene definita una nuova topologia che presenta le seguenti componenti:

- **TwitterStreamSpout**: scarica i tweets direttamente da **Twitter** e li fornisce in input al primo bolt di processazione. la libreria Twitter4j consente di inizializzare uno stream dei dati, inserendo le chiavi di accesso appartenenti all'applicazione generata presso la pagina developer di Twitter. Come requisito di sistema è quindi necessario creare un'applicazione al link di riferimento `https://developer.twitter.com` [2] ed utilizzare le chiavi generate in fase di configurazione del programma.

- **TwitterValuationBolt**: riceve in ingresso i tweets scaricati e per ognuno calcola il sentiment value. Esegue inoltre una ricerca della query di riferimento per il tweet processato. La ricerca può essere inoltrata su più campi del file Json ottenuto durante il download (l'hashtag o la keyword può essere contenuta anche all'interno dei campi `RetweetStatus` e `QuotedStatus`). Infine viene generato un nuovo output con l'aggiunta dei campi Query e Label legati al tweet processato.

- **TwitterHbaseBolt**: riceve lo status del tweet assieme al sentiment value e alla query di riferimento. Inserisce i nuovi dati all'interno della tabella di batch, in modo da consentire al job MapReduce di processare il nuovo dato al ciclo successivo. Aggiorna la tabella `LastRowTimeStamp` con l'indice dell'ultima riga inserita assieme al *timestamp* di inserimento.

2.5. Serving Layer

Il serving layer è implementato sul database distribuito Apache HBase. Grazie alle tabelle definite su di esso, è stato possibile memorizzare i dati processati durante la fase di batch aggiungendone di nuovi quando richiesto dallo speed layer. Ci sono inoltre tabelle specifiche che consentono di sincronizzare i livelli dell'architettura, mantenendo disguinti i set di dati su cui vengono eseguiti il batch e lo stream processing. Le views generate dal programma vengono salvate sull'apposita tabella `SentimentAnalysisTweets`.

2.6. Machine Learning: LingPipe

La scelta riguardo l'algoritmo di apprendimento, necessario per eseguire la sentiment analysis del testo, è ricaduta su **LingPipe**. Inizialmente si era optato per utilizzare l'algoritmo **Stanford Core**

NLP, che aveva il vantaggio di essere preaddestrato e di supportare la valutazione di testi in lingue diverse. Lo svantaggio era dovuto alla complessità dell'algoritmo che generava ritardi durante l'elaborazione dei dati. Non era quindi adatto a valutare i dati durante l'esecuzione del programma (soprattutto se consideriamo le caratteristiche dello speed layer). L'algoritmo LingPipe ha un tempo d'esecuzione più rapido ma non risulta preaddestrato. Dopo aver scaricato ed installato l'algoritmo deve essere definito il dataset su cui eseguire l'addestramento e le fasi di testing. Dopo aver ottenuto i dati necessari, viene eseguita la fase di training: le stringhe processate devono essere salvate su una directory, suddivise in subdirectory in base alla classe di appartenenza. Si ricorda che la classificazione è definita su 3 possibili valori: 4 se il testo è positivo, 2 se è neutro e 0 se risulta negativo. In ogni subdirectory è definito un cluster di file, contenenti le *sentences* di addestramento. La classe LingPipe deve conoscere il path sia della directory **Train**, sia della directory **Test**, la cui costruzione è analoga. Ogni file, inserito nelle rispettive directory, contiene un solo dato, letto dal classificatore nelle fasi di addestramento e testaggio.

3. Dataset

Il progetto richiede di avere a disposizione una notevole quantità di dati per inizializzare i processi. La tabella HBase, su cui è eseguito il batch processing, è richiesta durante l'inizializzazione del job MapReduce ed al momento dell'inserimento di una nuova riga riportata dallo speed layer. La tabella è inizialmente riempita inserendo i dati contenuti all'interno del database derbySQL. Al suo interno è presente la tabella `TweetsTable`, che contiene i dati del dataset reperibile al link `https://www.kaggle.com/kazanova/sentiment140` [1]. Il file scaricato in formato .csv è costituito da 1600000 di tweets valutati positivi e negativi, con classificazione analoga al programma. Le informazioni contenute all'interno del file specificano, per ciascun tweet, l'id, la lingua, la data di creazione e il testo. Essendo ogni testo scritto in lingua inglese, i tweets processati durante l'esecuzione del programma saranno saranno esclusivamente scritti in inglese. Parte dei dati contenuti nella tabella SQL TweetsTable sono utilizzati da LingPipe nella fase di configurazione (train e test). Per ovviare alla mancanza di tweets neutri su cui eseguire la classificazione, LingPipe ha accesso alle tabelle *Neutrals* del database SQL che contengono i tweets del dataset reperibile al link `http://claws.cc.gatech.edu/covid` [3]. Quest'ultimo è costituito da un elevato numero di tweets neutrali. Le informazioni contenute non includono il testo del tweet ma esclusivamente l'id e l'affidabilità in percentuale della classificazione neutrale. Per ovviare alla lacuna viene eseguita una ricerca dei tweets con affidabilità maggiore (99.9% o superiore), salvando il testo sulle tabelle **TrainNeutralTweets** e **TestNeutralTweets**. I dati ottenuti sono utilizzati dall'algoritmo di Machine Learning.

4. Esperimenti

Per comprendere i benefici che una Lambda Architecture è in grado di fornire quando struttura un'applicazione per gestire quantità massive di dati, sono stati eseguiti alcuni test in modalità distribuita. Gli esperimenti hanno riguardato nello specifico la ricerca di alcune parole, "Trump", "Biden", "Covid", indicate come queries nelle fasi di batch e speed processing. La scelta è ricaduta su tali chiavi poichè richiamano temi sensibili nel periodo storico corrente: le ultime elezioni presidenziali USA hanno visto la popolazione schierarsi su posizioni fortemente contrapposte, individuate nella figura dei due candidati alla presidenza, le cui divergenze non sono ancora state appianate. É interessante riportare una valutazione delle emozioni provate dai cittadini americani o di altri paesi riguardo gli ultimi avvenimenti politici. Il discorso è analogo per la scelta del termine "Covid", riferito alla crisi pandemica del 2020 che ha compromesso la vita di ogni individuo. Come test

iniziale si è eseguito un ciclo di batch sulla tabella HBase `DatasetTweetsAnalyzed`, riportando i risultati ottenuti in forma grafica. Successivamente è stato eseguito lo speed layer, sfruttando una topologia di Storm. La durata del processo è stata di un'ora, in modo da confrontare i risultati con quelli forniti dal batch processing. Infine i processi di entrambi i livelli sono stati eseguiti in parallelo, riportando la quantità di dati processata in un periodo di 5 minuti. Il modulo software dedicato alla realizzazione dei grafici, sfrutta le tabelle Hbase indicando, per ogni query, la dimensione delle *sentiment classes*. La forma grafica privilegiata è il *bar chart*, generato utilizzando la libreria `jfreechart`.

5. Analisi dei Risultati

Come è stato detto nei paragrafi precendenti il batch layer viene eseguito su un'opportuna tabella HBase, **DatasetTweetsAnalyzed**, che contiene circa 100000 tweets, prima dell'esecuzione dello speed layer. I dati inseriti appartengono al dataset *sentiment140* [1], contenente circa 1600000 tweets. Quest'ultimi non presentano un valore nel campo `Flag` che consenta di individuare le queries da ricercare. Dopo aver eseguito l'inserimento sulla tabella di batch, sono ricercate le parole 'Trump', 'Biden', 'Covid' all'interno del testo di ogni tweet, modificando il campo Flag. In seguito si esegue il batch layer che processa i tweets inseriti in tabella. Tuttavia i risultati ottenuti sono poco rilevanti:

- **Trump**: 18 tweets
- **Covid**: 13 tweets
- **Biden**: 2 tweets

L'aspetto negativo del batch processing è che non riesce ad ottenere più informazioni di quelle che già possiede; il grafico relativo è riportato in figura 2. Al contrario, eseguendo lo speed layer per un tempo di circa 60 minuti, il numero di tweets classificati è molto consistente. La Storm Topology è in grado di eseguire il download e la valutazione di ben 33625 nuovi tweets, aggiunti

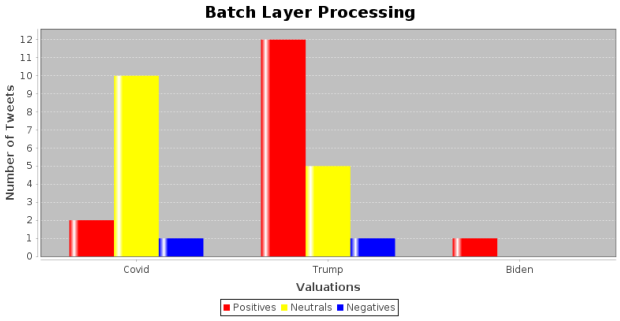

Figure 2. Processo del batch layer

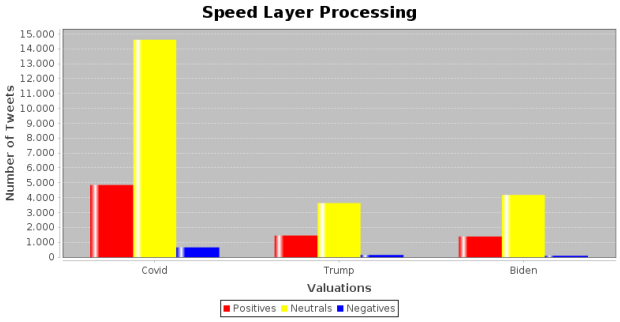

Figure 3. Processo dello speed layer

alla tabella principale. Come si evince dal grafico di figura 3 la query che ha avuto maggiore riscontro è 'Covid', i cui tweets neutri sono di gran lunga la maggioranza. Se analizziamo entrambi i grafici riportati è facile comprendere il motivo per cui i due livelli devono supportarsi durante l'elaborazione dei dati. Infatti, grazie all'ausilio dello speed layer, il batch layer otterrà nuovi dati di elaborazione al ciclo successivo. Eseguendo la Lambda Architecture per un periodo di 5 minuti, corrispondente a circa 3 cicli di batch, vediamo che il numero di tweets processati viene incrementato di 2761 unità.

Il risultato dell'esperimento è mostrato figura 4. Grazie ai risultati dei test si può concludere che l'esecuzione di una sentiment analysis è ottimizzata quando viene genstita da una lambda architecture; sia la qualità che la quantità dei dati influenzano fortemente la consistenza dell'analisi eseguita. Ogni livello dell'architettura funge da complemento per l'altro pur mantenendo la propria indipendenza; è quindi possibile aumentare le possibilità d'impiego dell'applicazione an-

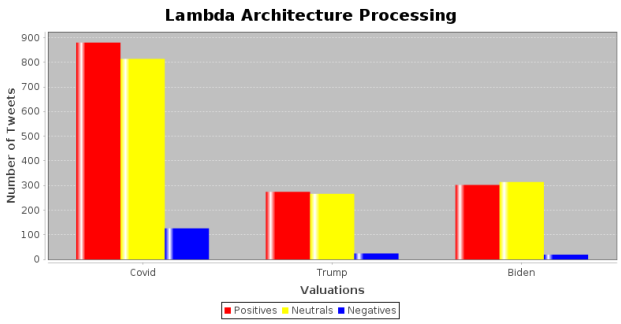

Figure 4. Processo della Lambda Architecture

dando ad agire singolarmente su ciascun livello.

6. Considerazioni Finali

La configurazione distribuita della Lambda Architecture apre a molteplici possibilità di ampliamento; un esempio è la creazione di un vero cluster di server, riportando l'architettura in modalità fully distributed. Altra considerazione è l'aggiornamento dei dati raccolti, aggiungendo per esempio le coordinate relative al Paese di appartenenza. Per aggiornamenti legati alla privacy non è più possibile accedere alla posizione precisa dei tweets ma è comunque consentito di conoscere il paese di origine. La gestione di una sentiment analysis da parte di un'applicazione lambda architecture consente di mantenere sempre aggiornato il dataset, incrementando il numero di tweets utili alla fase di processing. La suddivisione in livelli garantisce che ogni dato sia processato una volta per ciclo in modo da non penalizzare nè l'elaborazione di nuovi dati nè l'elaborazione dei dati gia valutati nelle fasi precedenti.

References

[1] KazAnova. Sentiment140 dataset with 1.6 million tweets, 2018. https://www.kaggle.com/kazanova/sentiment140.

[2] Twitter. developer twitter, 2020. https://developer.twitter.com/.

[3] S. S. Ziems, He Bing and K. Srijan. Racism is a virus: Anti-asian hate and counterhatein social media during the covid-19 crisis, 2020. http://claws.cc.gatech.edu/covid.