Analisi Tweet

Un’architettura lambda per l’analisi di flussi di dati provenienti da Social

DRData

De Matteis Valerio

Ronchetti Claudio

Sommario

Introduzione

Inutile ricordare quanto i social network vengono utilizzati quotidianamente da miliardi di persone. Secondo un'infografia pubblicata da *Tracx* sulla demografia dei Social Networks nel 2017, quasi 3 miliardi di persone attive utilizzano i Social Networks con conseguente enorme quantità di dati prodotti giornalmente. La natura del contenuto pubblicato può essere molteplice, e ovviamente alcuni di questi sono illeciti. Mediante i contenuti che circolano attraverso Internet possono essere commessi molti tipi di illeciti: alcuni reati molto gravi, come la propaganda terroristica o la pedopornografia; altri reati più circoscritti alla dimensione individuale, connessi alla violazione dei diritti della personalità, come il trattamento illecito dei dati personali, la diffamazione, il cyberbullismo, il furto d'identità.

Da un articolo del *The Guardian*, sembrerebbe che i social network siano preoccupati dai risarcimenti chiesti in seguito ad un fenomeno chiamato *porn revenge*. Si tratterebbe di procedimenti intentati da ex fidanzate (o fidanzati) per la pubblicazione online di foto o filmati con esplicite scene di sesso. Queste immagini, inizialmente realizzate con il consenso dei protagonisti ma destinate a visioni private, una volta mutati i rapporti, vengono diffuse per ripicca sul web. Spesso a questo scopo vengono utilizzate pagine aperte dei più noti social media, che ormai sono diventate il veicolo più veloce per far circolare ampiamente una notizia.

La domanda che ci possiamo porre è chi può essere ritenuto responsabile e, dunque, chiamato a risarcire?

Certamente la persona che ha reso di pubblico dominio il materiale. Con ogni probabilità non soltanto chi per primo lo ha divulgato in rete, ma anche chi lo ha "rilanciato", contribuendo alla diffusione. È assai più difficile individuare una responsabilità in capo agli internet service provider (Isp) o a chi gestisce il social network.

Su questo tema, fornisce qualche spunto il Dlgs 70/2003, che, in attuazione di una direttiva europea, ha disciplinato «alcuni aspetti giuridici dei servizi dell'informazione nel mercato interno, con particolare riferimento al commercio elettronico».

Si tratta di una disciplina senz'altro datata, ma che finora ha permesso di attribuire in qualche caso ai provider la responsabilità civile per gli illeciti commessi dai loro utenti. La direttiva europea è stata adottata con l'obiettivo di favorire la libera circolazione e la promozione dei servizi della società dell'informazione, eliminando gli ostacoli allo sviluppo del commercio elettronico. In quest'ottica, occorreva favorire l'attività degli Internet Service Provider (ISP) senza gravarli di oneri eccessivi – quali, ad esempio, la sorveglianza ex ante o ex post sui contenuti diffusi dagli utenti attraverso la rete Internet, al fine di prevenire o reprimere gli illeciti – che avrebbero rallentato lo sviluppo del mercato digitale.

Quindi questa disciplina esclude una responsabilità del sito che si limita a ospitare contenuti, a meno che il sito stesso non abbia una diretta conoscenza della illiceità o eviti di adempiere a un ordine di cancellazione proveniente dalla pubblica autorità.

Questo per non obbligare il provider a predisporre un sistema di filtraggio preventivo dei contenuti caricati dagli utenti, essendo tale sistema complesso, costoso e permanente.

Il problema del bilanciamento fra il diritto alla libera manifestazione del pensiero e quello alla protezione della sfera privata e della reputazione individuale è stato preso in considerazione dalla Corte europea dei diritti dell'uomo (Corte di Strasburgo) in alcune recenti sentenze.

Il vero problema sembra quello di capire se oggi chi ospita in rete contenuti generati da terzi non abbia davvero alcun controllo su tali contenuti.

Tuttavia, i gestori dei social network traggono profitto proprio attraverso la raccolta dei dati personali e sensibili degli utenti (il cosiddetto user data profiting), di cui gli inserzionisti pubblicitari si servono al fine di individuare i profili cui destinare pubblicità mirata in base ai gusti, alle preferenze e alle loro attitudini individuali. Inoltre, attraverso l'utilizzo di appositi algoritmi, i gestori delle piattaforme sono in grado di evidenziare alcuni contenuti (trending feeds) rispetto ai quali le interazioni fra utenti sono più frequenti, raccogliendo intorno ad essi pubblicità mirata. Dunque, i social network provider non si limitano a svolgere un'attività di hosting neutrale, ma intervengono direttamente nell'organizzazione, nella gestione e talvolta anche nell'editing dei contenuti, al fine di aumentare i ricavi derivanti dalla raccolta pubblicitaria.

I tempi si sono aggiornati e le tecnologie si sono evolute anche rispetto a pochi anni fa, quindi in questo documento vogliamo mostrare una semplice soluzione che ha l'obiettivo di filtrare i contenuti ritenuti illeciti in un ambiente attinente alle tematiche di Big Data, ma senza esprimere giudizi sugli effettivi costi di integrazione dei sistemi adottati.

Il sistema presenta un'architettura lamba, composta da due strati: *speed layer* e *batch layer*. Il primo con l'obiettivo di analizzare i dati in real-time al fine di individuare e segnalare i contenuti presunti illeciti, mentre il secondo analizzerà il database contenente tutti i contenuti per ottenere risultati interessanti da un punto di vista del marketing e di visione della concreta validità delle segnalazioni.

Il primo capitolo andrà a delineare gli obiettivi del progetto, mentre l'architettura e le tecnologie usate verranno descritte nel secondo capitolo. Infine, all'interno del terzo e quarto capitolo vengono segnati i risultati dell'anali batch e streaming e le osservazioni conclusive.

Obiettivi

Il seguente progetto ha come obiettivi principali quelli di risolvere il problema, da noi su espresso, che tratta tematiche di Big Data, inoltre sperimentare le diverse metodologie di Big Data ed analizzare e confrontare le tecnologie.

Il problema fa riferimento agli Internet service provider e ai Social Network. Noi, per semplicità, ci riferiremo solo ai Social Network ed in particolare parleremo di Twitter in quanto ampiamente discusso durante il corso e perché la società offre una libreria in grado di acquisire in real-time i tweet degli utenti.

Risoluzione problema di Big Data

Il sistema si basa su due principali strati:

- ***Speed layer***: provvede di analizzare i dati ricevuti in streaming di Twitter al fine di verificare se il contenuto interno a ciascun tweet potrebbe violare il sistema di sicurezza e se così fosse il sistema segnala il tweet illecito.
- ***Batch layer***: svolge analisi sui tweet memorizzati e sui log di segnalazione per trarne informazioni utili da un punto di vista di marketing e di ottimizzazione del servizio offerto.

L'obiettivo base è quello di costruire l'intero sistema andando ad unire le diverse tecnologie al fine di segnalare i tweet dove, all'interno del testo, c'è una parola chiave. Il passo successivo, se possibile, sarà quello di sostituire la verifica base ed aggiungere un classificatore di immagini in grado di verificare se l'immagine del tweet contiene materiale pornografico.

Analizzare e confrontare le tecnologie

Ultimo obiettivo sarà quello di analizzare e confrontare le tecnologie al fine di comprendere le caratteristiche di forza e/o punti deboli delle tecnologie utilizzate, ed infine confrontarle dal punto di vista dell'utilizzo.

In particolare, tutte le tecnologie verranno analizzate e descritte fornendo approfondimenti sul lavoro svolto e le problematiche risolte, ma il confronto avverrà solo tra quelle che appartengono allo *speed layer*. Il confronto verrà svolto andando a sostituire al sistema la sola tecnologia che svolgerà l'analisi su streaming di dati, ma mantenendo le altre componenti intatte.

Il confronto verrà calcolato sulla base di quelli che sono i vantaggi e gli svantaggi di ogni strumento.

Revisione di testi letterali

L'obiettivo principale è elaborare il dato appena ricevuto al fine di verificare se il dato pubblicato dai clienti sul social network o l'Internet service provider contiene del materiale illecito che si vuole segnalare nell'immediato.

Come espresso precedentemente il progetto base prevede la segnalazione dei testi con contenuto offensivo perché semplice da realizzare (il testo contiene parole chiavi).

Il secondo step è sostituire questa verifica con un classificatore di immagini per verificare se l'immagine pubblicata presenta materiale pornografico (o pedopornografico).

Per approfondire quest'ultimo argomento al fine di ottenere una maggiore panoramica sulle tecnologie e soluzioni adottate, abbiamo preso visione di alcuni recenti (da 2015 a 2018) paper scientifici. Da questi si evincono due principali tecnologie utilizzate a questo scopo:

- Decision Tree
- Artificial Neural Network

Ci sono diversi metodi per determinare se un'immagine o un video è porno. Un fattore determinate è rappresentato dalla nudità presente nell'immagine o nel frame video, ovvero la quantità di pixel inerenti alla pelle della persona (human skin) rispetto ai pixel totali. Ma questo non può essere l'unico fattore determinante.

Infatti, l'articolo di Chetneti Srisa-an [1] propone una nuova tecnica per classificare immagini pornografiche usando range di YCbCr [1] e tre misurazioni in percentuali: % Face Area, %AHB (Area Human Body) e % Eccentricy (rapporto tra la distanza dal centro a un fuoco e la distanza da quel fuoco a un vertice). Infine, un algoritmo computerizzato (C4.5) viene applicato per costruire un albero di decisione. Il contributo principale di questo documento è la semplicità nel mantenere un'elevata precisione entro un tempo di elaborazione accettabile. L'accuratezza dei risultati sperimentali è dell'85,2% con un tempo di elaborazione medio di 0,21314 secondi per immagine. Risultato ottimo se si prospetta nell'ottica di classificazione del dato in streaming.

La seconda alternativa è rappresentata dalle reti neurali espresse in *deep learning* [2]. In particolare, il principale metodo utilizzato a questo scopo è *Convolutional Neural Network* (CNN o ConvNet).

Nell'articolo di Mohamed N. Moustafa [2], mostra diversi classificatori di immagini che utilizzano questa tecnologia chiamati Alex-Net (ANet) e GoogLeNet (GNet). L'articolo propone una combinazione (AGNet e AGbNet) dei due per aumentare l'accuratezza come in Figura 1.

Approach	Accuracy (%)
BossaNova (HueSIFT) [2]	89.5±1
BossaNova VD (BinBoost16) [3]	90.9±1
Proposed ANet	92.01±3
Proposed GNet	93.7±3
Proposed AGNet	**93.8±2**
Proposed AGbNet	**94.1±2**

Fig.1 Accuratezza dei classificatori di immagini e video

[1] YCrCb: Famiglia di color space usata come parte della pipeline di immagini a colori nei sistemi di video e fotografia digitali. Y è la componente luminosa, Cb e Cr sono le componenti cromatiche del blu e del rosso.
[2] Deep learning: è quel campo di ricerca dell'apprendimento automatico (*machine learning*) e dell'intelligenza artificiale che si basa su diversi livelli di rappresentazione, corrispondenti a gerarchie di caratteristiche di fattori o concetti.

I dati relativi all'accuratezza dei classificatori sono eccellenti, ma non viene riportata alcuna informazione aggiuntiva riguardante le tempistiche.

Mentre nell'articolo [3], viene proposta una soluzione per rilevare se il contenuto è di tipo pedopornografico attraverso l'uso di una combinazione di due classificatori di immagini. Il primo mira a verificare se l'immagine o il video è porno, mentre il secondo effettua *face detection* per individuare la fascia d'età presente nell'immagine o nel video. Il metodo proposto raggiunge il 42% di accuratezza con un tempo medio di elaborazione di circa 5 secondi per ogni frame. Soluzione sufficiente nell'analisi real-time di immagini, ma non per i video. Infatti, il documento evidenzia (in Future work) che per l'analisi di video in real-time è necessario implementare una pipeline parallela.

In ultima aggiunta, la maggior parte degli approcci di visione artificiale per la rilevazione di immagini pornografiche utilizza il dataset NPDI fornito da Avila, Thome, Cord et al. [4].

Architettura e tecnologie

Il sistema presenta una architettura lambda, ovvero una architettura di *data-processing* progettata per modellare quantità enormi di dati prendendo i vantaggi dei metodi sia *batch* e sia *stream-processing*. Questo approccio cerca di bilanciare latenza, throughput, e fault-tolerance attraverso l'uso del processamento batch che fornisce comprensione e accuratezza nella visione del dato processato, mentre simultaneamente, usando un processamento stream in real-time, fornisce una vista dei dati appena ricevuti. I due risultati possono essere accumunati prima di essere presentati. La crescita dell'architettura lambda è correlata con la crescita di grandi dati (big data), analisi in tempo reale e al mitigare delle latenze di map-reduce.

Architettura lambda

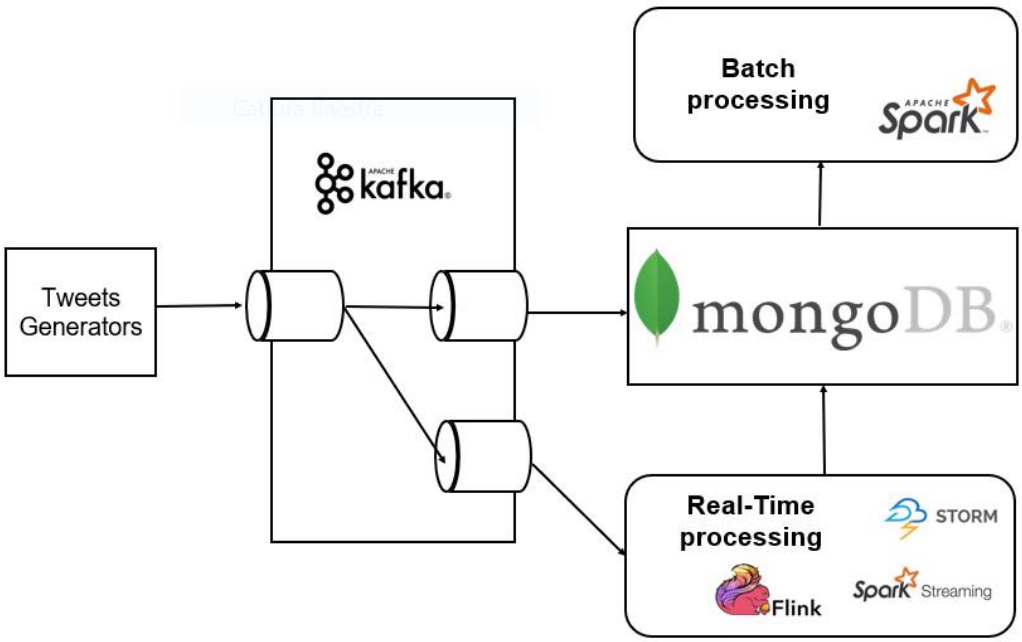


Fig. 2 Architettura lambda di riferimento

Tecnologie usate

JSON Data Generator:

Inizialmente avevamo parlato di una libreria resa disponibile da Twitter per ottenere i tweet in tempo reale. Ovviamente l'accesso ai tweet è limitato. E per ottenerli costantemente sono necessari dei certificati rilasciati a pagamento.

Per questo motivo abbiamo preferito utilizzare un generatore di file json, il quale rappresenta la nostra sorgente di tweet di esempio. Il formato json utilizzato è stato costruito copiando il formato dei tweet reali.

Il generatore che abbiamo utilizzato, è uno strumento molto flessibile, in quanto ci ha permesso di modificarlo in base alle nostre esigenze solamente dalla modifica di due file di configurazione:

1. *Workflow file*: qui si possono impostare la frequenza con la quale sono generati eventi, e inoltre si configura lo schema dell'evento generato, permettendo di personalizzare chiavi e valori del json che restituisce e randomizza il contenuto. Di seguito verrà riportato un esempio in figura 2 che ne descrive una parte del contenuto:
2. *Config file*: In questo file è possibile invece specificare il nome del Workflow file, e permette di specificare dove tale evento generato andrà immagazzinato. Tale luogo può essere semplicemente un log, o una coda Kafka. In tal caso è necessario specificare indirizzo ip, porta, e nome del topic dove andranno pubblicati i messaggi.

Apache Kafka

Nel nostro sistema, Kafka ha due compiti principali:

- Data ingestion: processo di acquisizione ed importazione dei dati per l'uso o l'archiviazione immediata in database. I tweet prodotti da JSON-Data-Generator vengono pubblicati su un apposito topic chiamato 'tweet-input'. Da qui, Kafka svolgerà la funzione di data ingestion acquisendo i tweet e importandoli allo speed-layer e nell'archiviazione immediata su database (MongoDB).
- Data filtering: processo di filtraggio sui dati acquisiti svolto prima di esser importati verso i consumatori. I tweet ricevuti che devo essere analizzati immediatamente dallo speed-layer vengono prima filtrati di tutti quei campi ininfluenti ai fini dell'analisi.

Nella nostra architettura, il flusso di informazioni viaggia attraverso un totale di tre topic:

1. *Tweet-input*: il tweet prodotto dal generatore viene inviato presso il topic iniziale.
2. *Tweet-intermadiate*: rispetto al tweet iniziale, Kafka esegue una operazione di trasformazione, assegnando ad ogni tweet un id di generazione che lo identifica in modo univoco (idealmente da un utente). Infine, viene immesso nel topic intermedio. Consumer iscritto a questo topic è MongoDB.
3. *Tweet-output-for-streaming*: Kafka prende i dati rilasciati da tweet-intermediate, esegue un filtraggio delle informazioni lasciando solo quelle essenziali (nome utente, testo, data, ecc.) per l'analisi streaming, allo scopo di ridurre i dati trasmessi in rete e la memoria necessaria ai consumatori per effettuare il calcolo. Il consumer iscritto a questo topic avrà lo scopo di effettuare analisi streaming come Flink, Storm o Spark Streaming.

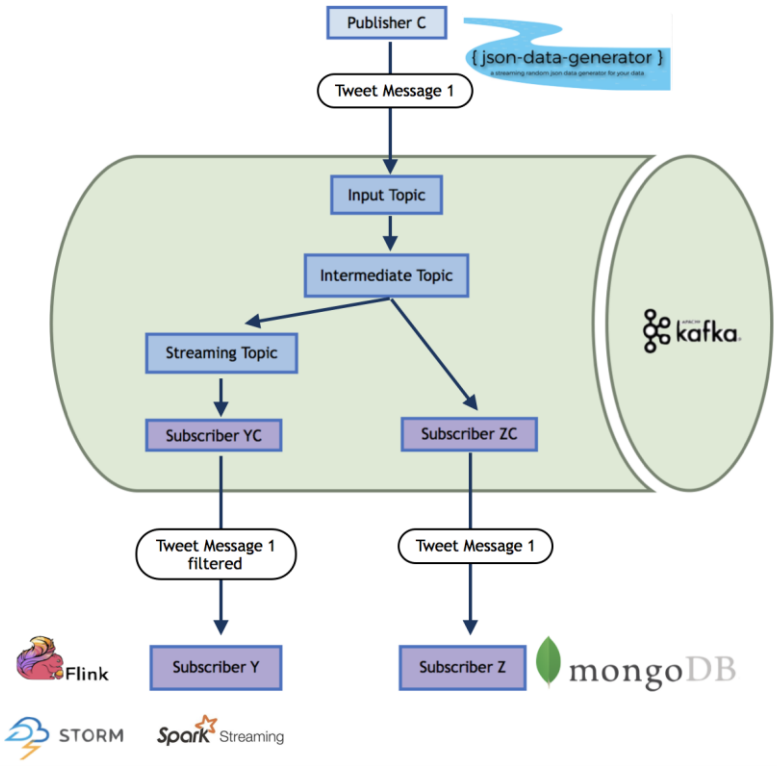


Fig.3 Esempio di data ingestion svolto da Kafka

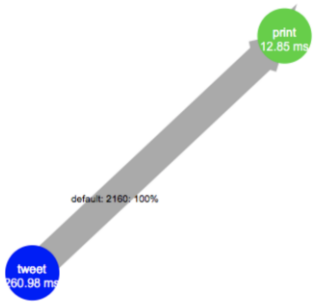
Apache Storm

Apache Storm è un sistema open source distribuito per computazioni in real-time. Storm rende facile lavorare in modo affidabile gli stream illimitati di dati, facendo per l'elaborazione in tempo reale quello che ha fatto Hadoop per l'elaborazione batch. Storm è veloce, scalabile, fault-tolerant.
Una topologia Storm consuma flussi di dati e processa tali flussi in modi arbitrariamente complessi, ripartizionandoli tra ogni fase del calcolo se necessario.

I principali vantaggi che espone Apache Storm sono (da noi riscontrati):

- Modello di programmazione semplice: simile all'approccio di MapReduce;
- Esegue qualsiasi linguaggio di programmazione: clojure, java, ruby, python. Il supporto per altri linguaggi può essere aggiunto mediante un protocollo di comunicazione di Storm;
- Modalità locale: Storm ha una modalità "locale" dove si simula completamente un cluster Storm in-process. Ciò consente di sviluppare e testare topologie molto velocemente;
- Comunità molto attiva.

Fig.4 grafo della topologia costruita in Storm



Nella nostra topologia di Storm sono stati definiti due nodi del grafo:

Spout Tweet: il primo nodo è ovviamente uno spout, una sorgente da cui provengono le informazioni. Tale spout preleva le informazioni dal topic tweet-output-for-streaming descritto precedentemente le quali a loro volta saranno inoltrate presso un bolt.

Bolt Print: il secondo e ultimo nodo del grafo, il quale riceve i tweet dal nodo spout, analizza il testo del tweet e se contiene un contenuto che non rispetta le politiche di privacy del social network questo viene segnalato e inserito in una collezione di documenti di MongoDB, creata appositamente per il risultato dello streaming di Storm. La segnalazione viene effettuata scandendo il contenuto dei singoli Tweet e banalmente verifica se esso contiene delle parole chiavi che abbiamo fissato in precedenza. Una soluzione più completa ed efficace sarebbe quella di utilizzare un classificatore tramite l'uso di conoscenze di machine learning, non ritenuta essenziale ai fini di questo progetto.

Apache Spark Streaming

Spark Streaming è un'estensione di Spark che sfrutta la capacità di programmazione veloce del suo Core per eseguire analisi in streaming. Prende dati in mini-batch (micro-batching) ed esegue trasformazioni RDD su essi.

L'approccio che si ha nell'impletare il codice su Spark Streaming restituisce lo stesso filing di Spark. Infatti, permette allo stesso codice applicativo scritto per l'analisi di batch si poter esser utilizzato per analisi in streaming, su un unico engine.

Come avviene in Spark, si inizia andando a specificare il Master in SparkConfig e in aggiunta la finestra temporale (in secondi) su cui effettuare l'analisi. Infatti, una volta avviato il codice, noteremo subito che i risultati dell'elaborazione vengono svolti nell'intervallo specificato. I dati vengono raggruppati in Dstream che a loro volta sono composti da RDD. Il grande vantaggio di Spark Streaming è il poter integrare questa tecnologia con tantissime altre, conosciute e non, attraverso l'uso di specifici connettori. Inoltre è possibile utilizzare l'API Dataset / DataFrame in Scala, Java, Python o R per esprimere aggregazioni di streaming, event-time windows, join stream-to-batch, ecc. Il calcolo viene eseguito sullo stesso motore Spark SQL ottimizzato.
Il problema principale di Spark streaming è che non è stato costruito per supportare al massimo dell'efficienza lo stream processing ma si adatta di più ad un modello di microbatching.

Fig. 5 Rappresenta le analisi streaming in finestre di 10 sec.

Apache Flink

Il nucleo di Apache Flink è concentrato per gestire flusso di dati in streaming scritto in Java e Scala.

Dall'homepage di Flink possiamo scegliere se scaricare la versione liscia o quella con Hadoop.
Flink non fornisce un proprio sistema di archiviazione dei dati,i dati di input devono essere conservati in un sistema di storage distribuito come HDFS o HBase. Per l'elaborazione del flusso di dati, Flink consuma i dati da code di messaggi come Kafka.

Durante la fase di scrittura del codice, abbiamo riscontrato numerosi rallentamenti dovuti a problemi inerenti alle versioni delle dipendenze (anche se di poco). Inoltre, la community non è

sufficientemente grande e attiva. In più Flink non supporta al meglio le espressioni lambda. Con l'utilizzo di esse è possibile ridurre notevolmente il volume del codice, e lo mantiene più pulito e leggibile.
Anche se le espressioni Lambda offerte da Java 8 sono molto convenienti, tuttavia, non forniscono quasi informazioni di tipo tramite la riflessione, motivo per cui Flink ha problemi con la generazione dei serializzatori sottostanti. Per poter eseguire i programmi Flink nell'IDE, si consiglia di utilizzare classi anonime Java invece di lambda, ogni volta che sono coinvolti tipi generici.

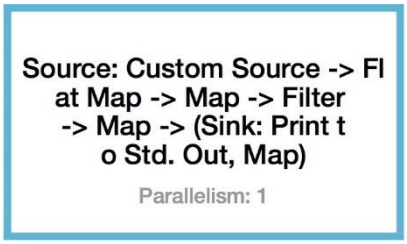


Fig.6 Rappresenta un Flink node locale

I programmi in Flink possono essere scritti in Java o Scala e vengono compilati automaticamente e ottimizzati in modo tale che i dati vengono eseguiti in un ambiente cluster o cloud.

Start Time	End Time	Duration	Name	Bytes received	Records received	Bytes sent	Records sent	Parallelism	Tasks	Status
2018-07-19, 18:33:46	2018-07-19, 18:38:35	4m 48s	Source: Custom Source -> Flat Map -> Map -> Filter -> Map -> (Sink: Print to Std. Out, Map)	0 B	0	0 B	10	1	0 0 1 0 0 0 0	RUNNING

Fig. 7 Informazioni dei task eseguiti su cluster Flink

Il principale vantaggio di Flink è che permette in uno stesso programma l'elaborazione batch e lo stream processing secondo un'architettura Lambda, ovviamente questo può essere visto anche come uno svantaggio se si vuole solo il massimo dell'efficienza sviluppata per stream processing.

MongoDB

MongoDB è un database Document Oriented con possibilità di replica e sharding dei dati. È dotato di una struttura di replica Primary/Secondaries che garantisce, sotto certi limiti e tramite un sistema di elezioni appropriato, il fail-over automatico del Primary a vantaggio di un dato Secondary.

Nella nostra architettura, MongoDB è un cluster che contiene un unico database che abbiamo chiamato *clusterdb* al cui interno abbiamo definito varie collezioni:

- *Tweets*: nella quale saranno immagazzinati i tweet ricevuti da Kafka dal topic tweet-intermediate;
- e altre collezioni dove ciascuna conterrà i risultati delle analisi Batch e Streaming.

Confronti

In questa sezione si discuterà della ricerca che è stata fatta in merito alla comparazione delle tre soluzioni tecnologiche streaming utilizzate. Inizialmente, l'idea era di sfruttare il timestamp che si trova di base all'interno del tweet al momento della creazione (*timestamp_1*) e confrontarlo con il timestamp al momento della trascrizione della segnalazione su database (*timestamp_2*):

Tempo di processamento = timestamp_2 - timestamp_1

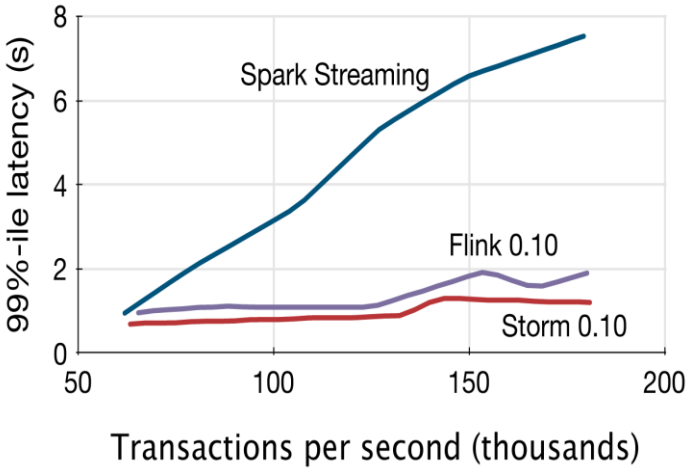
Questa informazione, non ci dà alcuna informazione utile al fine di valutare la specifica soluzione tecnologica, il tempo infatti riguarda il processamento del singolo tweet. La tempistica calcolata potrebbe essere influenzata in merito ai rallentamenti del singolo computer (funzionamento in locale), oppure più veloce dovuta al fatto che si è totalmente trascurato il costo di trasmissione di dati in rete. Considerazioni identiche anche se fosse stata considerata un numero finito di tweet.

Si è dunque ritenuto opportuno evitare di confrontare con questa metodologia la nostra architettura in quanto riteniamo non sia sufficiente e si è preferito utilizzare un altro approccio.

La nostra soluzione è stata quella di paragonare le tre soluzioni tecnologiche su un ambito più generale e allo stesso tempo più completo, che possa dare un'idea di quelli che sono le differenze in termini di caratteristiche tra le varie soluzioni, dei vantaggi e degli svantaggi, e di mostrare ciò che è in generale già noto su internet, ovvero il rapporto latenza/throughput di Storm, Spark e Flink.

In particolare, questa tabella mostra come effettivamente si differenziano le soluzioni tecnologiche se confrontate in un contesto reale per la quale esistono diversi e differenti cluster su nodi fisici diversi che tengono in considerazione di eventuali rallentamenti in locale e rallentamenti di trasmissione in rete.

	STORM	**SPARK STREAMING**	**FLINK**
VANTAGGI	Una latenza molto bassa O(msec), vero streaming e un alto throughput. Eccellente per casi d'uso di streaming non troppo complessi.	Supporta espressioni Lambda, derivanti da Spark Alto throughput, buono per molti casi d'uso dove non è richiesta bassa latenza. Analisi microbatch. Ha le API più semplici da usare. Grande community di persone. Exactly One.	Leader dell'innovazione nello streaming opensource. Bassa latenza nel range di O(msec) e alto throughput, configurabile in base alle necessità. Auto-configurato, non troppi parametri da impostare. Exactly One.
SVANTAGGI	Non esiste un supporto specifico per la gestione dello stato del sistema. Non esistono caratteristiche avanzate che mi permettano di fare processamento di eventi nel tempo come aggregazioni, windowing, watermarks e sessioni. At least Once.	Non è vero streaming, streaming basato su microbatch. Alta latenza O(sec), nel range di secondi. Troppi parametri da configurare e difficile farlo bene.	Community non grande come quella di Spark, ma sta crescendo



La figura mostra come in un contesto applicativo reale, all'aumentare delle transazioni al secondo, partendo da circa 50.000, aumenta la latenza. È particolarmente evidente come la soluzione streaming di Spark, non è adatta in contesti dove si richiede una latenza molto bassa

Fig.8 Grafico che confronta le tre soluzioni in termini di latenza/throughput.

Scelta del framework di streaming:

È importante tenere a mente che nessun singolo framework di elaborazione può essere un proiettile d'argento per ogni caso d'uso. Ogni framework ha alcuni punti di forza e anche alcuni limiti. Tuttavia, con una certa esperienza, si può arrivare a prendere una decisione.

Dipende dai casi d'uso:
Se il caso d'uso è semplice, non è necessario cercare la soluzione tecnologica più aggiornata e completa, soprattutto se è complicata da imparare e implementare. Molto dipende da quanto siamo disposti a investire. Ad esempio, se si tratta di un semplice tipo di avviso IOT basato sugli eventi, Storm o Kafka Stream è perfettamente adatto per funzionare.
Allo stesso tempo, dobbiamo anche avere una considerazione consapevole su quali saranno i possibili casi d'uso futuri. È possibile che richieste di funzionalità avanzate, come l'elaborazione dell'evento, aggregazione, join di stream, ecc., possano avvenire in futuro e se la risposta è sì, allora è meglio procedere con framework di streaming avanzati come Spark Streaming o Flink. Una volta investito in una tecnologia, il suo costo è enorme se si volesse cambiare soluzione in seguito.

Stack tecnologico esistente:
Un altro punto importante è considerare lo stack tecnologico esistente. Se lo stack esistente ha Kafka in posizione end-to-end, allora Kafka Streams o Samza potrebbero essere più facili da montare. Allo stesso modo, se la pipeline di elaborazione è basata sull'architettura Lambda e Spark Batch o Flink Batch sono già presenti, è quindi opportuno considerare Spark Streaming o Flink Streaming.

Risultati

In questa sezione del documento saranno mostrati vari risultati in merito alle analisi Batch e alle analisi Streaming. Essi sono stati ottenuti montando l'intera architettura lambda mostrata nella prima sezione del documento. I componenti Streaming e dunque Flink, Storm e Spark Streaming sono gli unici elementi che non erano attivi contemporaneamente, ma sono stati testati a rotazione. L'output corrispondente è invece generalizzato allo streaming in generale poichè essi restituiscono tutte e tre lo stesso risultato.

Alcune soluzioni sono state implementate in java, altre in python. La scelta che ha portato a utilizzare più di un linguaggio di programmazione è stato frutto di numerosi esperimenti relativi alle API delle soluzioni tecnologiche in entrambi i linguaggi.

Di seguito sono riportate le query che abbiamo ritenuto più significative da implementare, con una breve descrizione del codice che descrive la rispettiva soluzione e i risultati ottenuti quando l'intera architettura lambda era up, suddividendole per Analisi Batch, e analisi streaming:

Analisi Batch

1. *Mostrare il testo dei primi 10 tweet aventi il maggior numero di retweet: Ad es.*

TESTO TWEET	N. RETWEET
"Oh that beautiful girl"	1000
"I hate monday"	999
"I love cats"	999
"Do we need to go at school this week?"	998
"Happy new year bro!"	990
"Fuck you man!"	985
"Good!"	984

Query adottata e risultato ottenuto (si noti la saturazione a 1000 perché *retweet-count*, nel file di configurazione di Json-Data-Generator, è stato impostato come *range*(0,1000)):

```
spark.sql("SELECT text, retweet_count FROM tweets ORDER BY retweet_count DESC LIMIT 10");
```

```
"text" : "Good!", "retweet_count" : 1000 }
"text" : "oh that beatiful girl", "retweet_count" : 1000 }
"text" : "i hate monday!!!", "retweet_count" : 1000 }
"text" : "i love cats", "retweet_count" : 1000 }
"text" : "i hate monday!!!", "retweet_count" : 1000 }
"text" : "do we need to go at school this week?", "retweet_count" : 999 }
"text" : "I will be spending twice as much time analyzing my fantasy basel
"text" : "Happy new year", "retweet_count" : 999 }
"text" : "fuck you bro", "retweet_count" : 999 }
"text" : "sell MacBookPro at 1700$", "retweet_count" : 999 }
```

2. *Calcolare la media giornaliera di tweet inviati raggruppati per città:*

Ad es.

CITTA	MEDIA TWEET PER GIORNO
Roma	68
New York	65
Venezia	63
Barcellona	62
Berlino	60
Valencia	56
Palermo	53
El Cairo	51
Milano	49
Mosca	48

Query adottata e risultato ottenuto:

```
spark.sql("SELECT AVG(t.totale) as averageForDay, t.place FROM (SELECT COUNT (1) AS totale,"
        + " formatTime(created_at) as date, place FROM tweets GROUP BY formatTime(created_at),place) t"
        + " GROUP BY t.place ORDER BY AVG(t.totale) DESC");
```

```
"averageForDay" : 69, "place" : "Mosca" }
"averageForDay" : 68, "place" : "Roma" }
"averageForDay" : 65, "place" : "New York" }
"averageForDay" : 63, "place" : "Venezia" }
"averageForDay" : 62, "place" : "Barcellona" }
"averageForDay" : 60, "place" : "Cosenza" }
"averageForDay" : 60, "place" : "Parigi" }
"averageForDay" : 56, "place" : "Praga" }
"averageForDay" : 53, "place" : "Berlino" }
"averageForDay" : 53, "place" : "Valencia" }
"averageForDay" : 51, "place" : "Palermo" }
"averageForDay" : 51, "place" : "Hawaii" }
"averageForDay" : 49, "place" : "El Cairo" }
"averageForDay" : 48, "place" : "Milano" }
```

Nella query su implementata, la funzione formatTime ha lo scopo di tradurre la data di creazione del tweet nel formato yyyy-mm-ddT h:m:s.millisecZ come ad es.

2018-07-14T14:40:06.181Z

in yyyy-mm-dd come ad es.

2018-07-14

3. *Calcolare e ordinare in modo decrescente gli utenti che hanno ricevuto il maggior numero di segnalazioni:*

Ad es.

UTENTE	N. SEGNALAZIONI
Cristopher	12
Martin	11
Mike	11
Cliff	10
Dominic	10
Bart	10
Marc	8
Homer	6
Larry	2

Query adottata (in modo differente dalle precedenti) e risultato ottenuto:

```
user_count = df.groupBy('name')\
    .count()\
    .sort('count',ascending=False)
```

```
+----------+-----+
|      name|count|
+----------+-----+
|   Valerio|    5|
|Cristopher|    5|
|      Alan|    4|
|    Mathew|    4|
|      Paul|    4|
|   Derrick|    3|
|    Hector|    3|
|   Dominic|    3|
|      Marc|    3|
|     Peter|    3|
|     Cliff|    3|
|     Claud|    3|
|   Fellman|    2|
|    Steewe|    2|
|    Martin|    2|
|   Emanuel|    2|
|   Everitt|    2|
|      Lisa|    2|
|      Mike|    2|
|    Marcus|    1|
+----------+-----+
```

Analisi Streaming

Per le analisi Streaming, è stata effettuata una sola query, e viene mostrato un solo risultato in quanto è lo stesso prodotto da entrambe le tre soluzioni tecnologiche utilizzate.

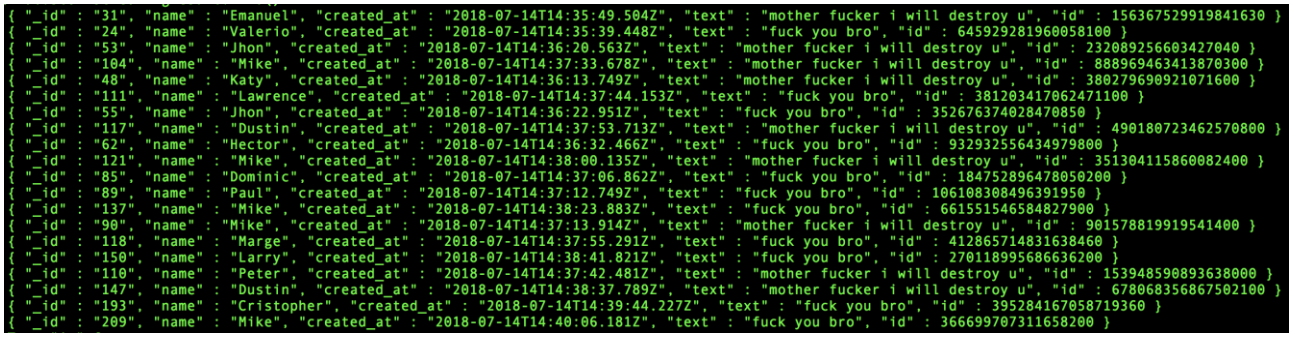
Abbiamo opportunamente scelto una analisi streaming che fosse il meno forzata possibile e che si potesse in un contesto reale prendere seriamente in considerazione. È utile infatti controllare tempestivamente i contenuti come testi/foto degli utenti che inseriscono in un social network, al fine di prendere immediatamente provvedimenti, come evitare la pubblicazione del tweet ad esempio.

1. *Segnalare e immagazzinare in MongoDB tweetID, nome, timestamp , testo del tweet e userID di tutti i tweet che non rispettano le politiche di sicurezza di Twitter:*

Ad es.

TWEET_ID	NOME	DATA	TESTO	USER_ID
31	Emanuel	2018-07-14	“Mother Fucker I will destroy you”	983591648
24	Valerio	2018-07-14	“Fuck you bro”	309457634
53	Federica	2018-07-14	“Shit!”	397834956
117	Carlo	2018-07-14	“Fuck you!!!”	534538962
89	Lucia	2018-07-13	“Jesus Christ, help me!”	425436806
44	Claudio	2018-07-13	“Asshole!!”	334687947

Risultato ottenuto dallo speed-layer dalle tecnologie Storm, Spark-Streaming e Flink:



Conclusioni

Arrivati alla conclusione del progetto, siamo soddisfatti dell'architettura che abbiamo creato e che siamo riusciti a far interamente funzionare. Si sono incontrate numerose difficoltà riguardo la gestione delle dipendenze e dei pacchetti per ogni soluzione tecnologica, in particolare nella gestione dei conflitti delle varie versioni dei pacchetti, integrazione dei componenti e connettori. Si è infine avuto modo di ricercare, studiare e riflettere sulle soluzioni streaming che abbiamo adottato, avendo un quadro molto chiaro di quali sono i vantaggi e svantaggi di ognuna, e il trade-off che si deve pagare nel caso si sceglie una soluzione rispetto ad un'altra. È importante far notare che la scelta delle tecnologie che abbiamo voluto sperimentare, e il modo in cui abbiamo voluto fare analisi è stata esclusivamente una nostra scelta, non dettata effettivamente da motivi di business che in un contesto reale sarebbe ciò che muoverebbe la scelta di una architettura pensata meglio.

Per fare un esempio pratico, sarebbe stato possibile analizzare flussi di stream real time anche utilizzando kafka stesso, analizzare i tweet con Hive e utilizzare un NOSQL db come Cassandra.

La natura didattica del progetto ha fatto sì che potessimo avere una certa libertà nello sperimentare e nel provare soluzioni di cui molte solamente accennate durante il corso, ma che al giorno d'oggi sono molto richieste.

Riferimenti

[1] Chetneti Srisa-an, A classification of Internet pornographic images, *International Journal of Electronic Commerce Studies* Vol.7, No.1, pp.95-104, 2016.

[2] Mohamed N. Moustafa, Applying deep learning to classify pornographic images and videos, 28 Nov 2015.

[3] Jongbin Jung, Rahul Makhijani, Arthur Morlot, Combining CNNs for detecting pornography in the absence of labeled training data, 2017.

[4] S. Avila, N. Thome, M. Cord, E. Valle, and A. D. A. ArauJo, “Pooling in image representation: The vi- ´ sual codeword point of view,” Computer Vision and Image Understanding, vol. 117, no. 5, pp. 453–465, 2013.

Sitografia

https://www.woodmark.de/blog/apache-spark-vs-apache-flink/

https://www.ericsson.com/research-blog/apache-storm-vs-spark-streaming/

https://www.slideshare.net/HadoopSummit/performance-comparison-of-streaming-big-data-platforms