Università degli Studi di Bari

DIPARTIMENTO DI INFORMATICA
Corso di Laurea Triennale in Informatica

Piattaforma cloud per l'analisi di big data provenienti da social network

Tecnologie per la prevenzione di discriminazioni sociali

Relatori:
Prof. Pasquale Lops
Dott. Pierpaolo Basile

Candidato:
Gianvito Taneburgo
Matricola 587176

Anno Accademico 2013 – 2014

Indice

Parte I

Prefazione

Introduzione

Il progresso tecnologico degli ultimi anni ha rivoluzionato il modo con cui gli esseri umani conoscono, lavorano, si relazionano con gli altri e, più in generale, interagiscono in un contesto sociale. Il fattore decisivo per questo cambiamento è stato Internet, la più grande rete di computer del mondo, accessibile dal 40% circa della popolazione mondiale[http://www.internetworldstats.com/stats.htm].

Nato nel 1991 per opera di Tim Berners-Lee, il World Wide Web si è evoluto molto velocemente fino a diventare la più ricca sorgente di informazioni della storia dell'uomo, accessibile consultando liberamente miliardi di pagine web. Lo sviluppo di complesse tecnologie e di nuove tecniche di programmazione ha radicalmente modificato la struttura di queste pagine, che hanno perso la loro natura statica per acquisirne una notevolmente più dinamica. Gli utenti del Web hanno così potuto cominciare ad interagire con le pagine visualizzate ed a contribuire in prima persona alla pubblicazione di nuovi contenuti. In questa maniera si sono sviluppate nuove tipologie di siti, come forum, blog, chat, wiki e, solo in tempi più recenti, **social network**. La trasformazione è stata così profonda da indurre all'utilizzo dell'espressione *Web 2.0*, a testimoniare una completa evoluzione del più celebre servizio di Internet dalla sua forma iniziale.

Gli utenti hanno acquisito primaria importanza nell'era dell'informazione - non a caso la rivista TIME sceglie come persona dell'anno nel 2006 *You*[http://content.time.com/time/covers/0,16641,20061225,00.html][http://content.time.com/time/
- e sono diventati sorgenti copiose di nuove informazioni. Il volume di dati prodotti è così aumentato a dismisura col passare degli anni, avviando quella che viene definita come *era dei* ***big data***.

Diversi fattori hanno contribuito alla crescita della mole di informazioni condivise in rete: lo sviluppo degli smartphone e delle reti di comunicazione digitali per telefonia mobile, che hanno permesso di condividere nuovi contenuti in ogni momento; l'aumento della velocità media di connessione ad Internet; la presenza di sensori in molti dispositivi di uso quotidiano; ecc. Il risultato di questi processi è stata una produzione di informazioni senza precedenti, che si sono però concentrate nelle mani di pochi colossi

dell'*Information Technology* (IT), proprietari dei siti web più frequentati dagli utenti. Queste aziende hanno utilizzato i dati in loro possesso per creare nuove ed abbondanti fonti di guadagno, accrescendo la loro ricchezza ed il loro potere in modo proporzionale al volume di informazioni raccolte.

Nella classifica dei siti web più popolari al mondo[http://www.alexa.com/topsites] le prime posizioni sono spesso occupate dai social network, servizi che aggregano milioni di utenti al fine di costituire una rete sociale virtuale. Alcuni tra i più celebri sono Facebook e Twitter, ma ve ne sono altri specifici per determinati target di utenti (Instagram, Tumblr, Pinterest, LinkedIn, ecc.). Negli ultimi anni i social network sono stati protagonisti di una espansione imponente, superando la complessa fase di *start up* per merito soprattutto della loro semplicità d'uso e dell'innovazione portata nel panorama dei siti web preesistenti. Questa improvvisa ed inaspettata crescita ha presentato diverse sfide alle aziende proprietarie dei rispettivi siti. I principali problemi che hanno dovuto fronteggiare riguardano la tutela della privacy degli utenti, la necessità di trovare forme di monetizzazione per supplire agli elevati costi di gestione, la ricerca di interfacce facilmente usabili da individui profondamente diversi tra loro ma, soprattutto, lo sviluppo di **infrastrutture scalabili e performanti**, adatte a gestire enormi bacini di utenza e di dati. Quest'ultima, certamente, rappresenta la più complessa sfida che gli *entrepreneur* hanno vinto per merito di avanzate tecnologie progettate appositamente per **memorizzare** ed **elaborare** big data.

Tali tecnologie sono state oggetto di studio per questo lavoro e, successivamente, sono state impiegate per lo sviluppo di una **piattaforma cloud** per l'analisi di grandi moli di dati provenienti da social network. La piattaforma è stata progettata per essere di supporto alla *Mappa dell'Intolleranza*, uno strumento promosso da Vox, Osservatorio Italiano sui Diritti, che si propone di fornire alle amministrazioni locali un mezzo per monitorare le zone italiane dove omofobia, razzismo, discriminazione verso i disabili, misoginia ed antisemitismo sono maggiormente diffusi. La *Mappa dell'Intolleranza*, infatti, mira ad evidenziare i luoghi affetti da questi gravi problemi di discriminazione mostrandoli su una mappa di calore dell'Italia, ottenuta dall'analisi di messaggi geolocalizzati inviati su Twitter dal nostro Paese.

La piattaforma sviluppata si prefigge di essere l'infrastruttura del sistema ed uno strumento scalabile per la raccolta e l'analisi di grandi moli di dati provenienti da diverse sorgenti informative, in modo particolare dai social network, che si prestano facilmente a veicolare messaggi discriminatori. Per raggiungere questi obiettivi è stata progettata una architettura ibrida, che fa uso di tecnologie cloud per completare i compiti più esosi di risorse e tecnologie tradizionali per gli altri. Dopo diverse analisi, Google è stato selezionato come il *cloud provider* ideale per il caso di studio; per tale motivo

quest'ultimo impiega nel suo *core* diversi servizi offerti dalla Google Cloud Platform, un portfolio di prodotti cloud riservato ad aziende e privati.

Oltre ad essere stata utilizzata per memorizzare e processare i messaggi raccolti da Twitter, la piattaforma è stata anche impiegata per cercare di migliorare il procedimento di individuazione dei tweet discriminatori, conducendo indagini di natura statistica sui dati raccolti. L'obiettivo di questa analisi è stato quello di individuare nuove parole tipicamente impiegate in contesti discriminatori, al fine di aumentare il numero di informazioni da mostrare sulla Mappa dell'Intolleranza.

Uno studio comparativo tra le prestazioni raggiunte con sistemi tradizionali e quelle osservate nei sistemi cloud-based ha permesso di dimostrare l'effettiva utilità della soluzione adottata per condurre elaborazioni simili a quelle del caso di studio, che coinvolgono l'impiego di big data.

Sommario

Di seguito viene illustrata per sommi capi la struttura di questo lavoro.

Il capitolo 1 descrive i big data, analizzando i fattori che hanno permesso il loro sviluppo, le sorgenti contemporanee di nuove informazioni, con particolare riguardo per i social network, e le possibilità di profitto originabili dal loro impiego. Vengono, infine, spiegate le due criticità principali che emergono quando si gestiscono big data: l'elaborazione e la memorizzazione.

Per risolvere il primo problema, nel capitolo 2 viene illustrato il funzionamento di Apache Hadoop, un framework open-source per il calcolo distribuito attraverso cluster di computer, ispirato da una tecnologia di Google. Nel corso del capitolo vengono approfonditi il file system distribuito del framework, il modello di programmazione MapReduce da seguire nella scrittura di programmi ed i principi di funzionamento della tecnologia.

Il capitolo 3 è destinato alla descrizione dei database NoSQL, una nuova tipologia di database, appositamente progettata per memorizzare grandi moli di informazioni e spesso impiegata per risolvere i problemi annessi allo storage di big data. Nel capitolo vengono prima illustrate le origini di questi database e, successivamente, viene effettuato un confronto con i database relazioni e quelli NewSQL, la nuova generazione di database, al fine di evidenziare i punti di forza e di debolezza di queste tecnologie. La descrizione della tassonomia dei database NoSQL conclude la parte sulle tecnologie di memorizzazione di big data.

Il capitolo 4 descrive le piattaforme di cloud computing, infrastrutture ideali per eseguire le tecnologie descritte nei capitoli precedenti su macchine remote, beneficiando della scalabilità e delle performance garantite dal cloud provider. Particolare attenzione viene dedicata alla Google Cloud Platform, la piattaforma cloud utilizzata nel caso di studio, di cui sono descritti i due web service utilizzati: Compute Engine e Cloud Storage.

La piattaforma cloud di analisi di big data provenienti da social network che è stata sviluppata utilizzando tutte le tecnologie finora presentate viene descritta nel capitolo 5. Al suo interno è possibile leggere il contesto e le finalità del lavoro, la descrizione della sua architettura e dei suoi componenti

più importanti, l'utilizzo che ne è stato fatto, gli studi condotti ed i risultati ottenuti.

Completano il lavoro delle sezioni contenenti le conclusioni sul lavoro svolto, i possibili sviluppi futuri del caso di studio ed alcuni sentiti ringraziamenti.

Parte II

Caso di studio

Capitolo 1

Big data

1.1 Definizioni

Con l'espressione big data generalmente ci si riferisce ad una grande quantità di dati digitali di varia natura. Tali dati possono essere originati nei modi più diversi: transazioni bancarie, cronologie di acquisti su siti di e-commerce, registrazioni di fenomeni atmosferici, sms, e-mail, ecc.

Nel condurre uno studio su questi dati risulta inevitabile fare riferimento alle unità di misura dell'informazione digitale. La tabella 2.1, fedelmente alle indicazioni [http://www.bipm.org/utils/en/pdf/si-brochure.pdf - sezione 3.1, nota a margine] del Sistema internazionale di unità di misura (SI), riporta i multipli del byte che saranno incontrati in questo e nei successivi capitoli. La sua consultazione è utile per avere una percezione della dimensione di certe sorgenti informative ed è raccomandata ad ogni nuova occorrenza di ordini di grandezza con cui non si ha familiarità.

Definire i big data costituisce necessariamente il punto di partenza per l'analisi.

Il modo più semplice per formulare una definizione di big data è quello di fissare una soglia oltre la quale un insieme di data possa ragionevolmente essere considerato big. Si potrebbe, ad esempio, affermare che un qualunque insieme di informazioni avente dimensione superiore a 100 PB possa essere etichettato come big data. Questo approccio quantitativo, sebbene sembri universalmente valido, nasconde diverse insidie. Il primo problema che insorge è quello di riuscire a quantificare ragionevolmente questa soglia. Il concetto di big, infatti, è assolutamente relativo, poiché fortemente vincolato alle capacità di chi si trova a trattare la mole di dati. Un file di log contenente 35 milioni di operazioni, infatti, potrebbe risultare difficilmente processabile da un commerciante locale, che probabilmente non venderà

tanti oggetti in tutto l'arco della sua attività, ed assolutamente ordinario per Amazon, che è ha venduto oltre 37 milioni di prodotti in un solo giorno [http://expandedramblings.com/index.php/amazon-statistics/]. Ma vi è anche un altro problema che deriva da questo approccio: una definizione di big data ben formulata non dovrebbe essere soggetta ad obsolescenza. Fissando una precisa quantità, invece, si corre il rischio, un domani più o meno prossimo, di avere una definizione non più valida. Si supponga, infatti, di aver trovato una soglia che oggi risulti adatta ad identificare i big data; non serve troppa lungimiranza per comprendere che, qualunque essa sia, col passare del tempo essa risulterà inevitabilmente inadatta. La capacità di memorizzazione dei dispositivi, infatti, crescerà continuamente, trasformando quelli che avevamo definito come big data in dei not-so-big data.

Risulta chiaro, da quanto detto, che la definizione di big data deve essere inerentemente relativa, per rimanere appropriata, nel corso del tempo, agli strumenti che saranno prodotti dal progresso tecnologico. Questa idea è generalmente condivisa da tutti coloro i quali studiano e trattano big data.

Tra le tante definizioni di big data che è possibile trovare, una sintetica e diretta recita:

It's Big Data when it stops fitting on a single machine [Hbase in action – pag6]

Tale definizione potrebbe risultare accettabile, poiché è relativa alla capacità di memorizzazione di una macchina, che, verosimilmente, aumenterà nel corso degli anni. In tal modo, quelli che oggi sono considerati big data in futuro non lo saranno più, in favore di dataset aventi dimensione maggiore.

Una definizione migliore, tuttavia, è la seguente:

Big Data means the data is large enough that you have to think about it in order to gain insights from it [HBase in action – pag 6]

Questa definizione si sofferma non su un aspetto quantitativo, che, come è stato illustrato, può comportare diversi problemi, ma su un aspetto qualitativo, ben più interessante. I big data vengono definiti tali non più in virtù dello spazio fisico che occupano, ma in relazione alle modalità di gestione che essi richiedono. Si può parlare di big data quando le informazioni a disposizione sono così tante da richiedere un accurato ragionamento su come utilizzarle per estrarne qualche forma di conoscenza.

Anche l'IBM sottolinea come l'espressione big data alluda ad una realtà quotidiana che non lancia una sfida riducibile esclusivamente alla memorizzazione:

The term "Big Data" is a bit of a misnomer since it implies that preexisting data is somehow small (it isn't) or that the only challenge is its sheer size (size is one of them, but there are often more).

Vi è generale consenso sul fatto che i big data richiedano un modo sostanzialmente diverso di pensare alle modalità di utilizzo delle informazioni a disposizione, soprattutto se l'obiettivo è di trarne profitti.

Gli strumenti adatti all'elaborazione di dati di modeste dimensioni si rivelano del tutto incapaci di trattare big data. Apposite tecnologie sono state sviluppate per consentire di effettuare su enormi moli di dati le stesse operazioni che solitamente vengono eseguite su campioni più piccoli (acquisizione, memorizzazione, condivisione, analisi e visualizzazione). Alcuni degli strumenti che permettono di elaborare big data saranno analizzati e presentati in questo lavoro.

1.2 Fattori di sviluppo

La disponibilità di una enorme mole di dati che caratterizza i nostri giorni ha origine in una serie di fattori che, sinergicamente, hanno reso possibile la produzione di informazioni a ritmi elevatissimi.

Nessuno riuscirebbe, da solo, a generare big data in tempi modesti: essi sono il risultato di piccoli contributi di individui sparsi nel mondo. Risulta evidente, pertanto, che il fattore principale che ha avviato l'era dei big data è stato Internet, lo strumento che ha collegato gli abitanti del pianeta ed ha permesso a ciascuno di noi di poter fruire di una quantità enorme di contenuti.

Con oltre 14000 miliardi di pagine web consultabili, lo 0,35% delle quali è indicizzato da Google, 3 miliardi di internauti (circa il 41% della popolazione mondiale[http://www.internetworldstats.com/emarketing.htm]) hanno avuto la possibilità di condividere i loro dati ed attualmente si suppone che siano accessibili circa 672 EB di contenuti di natura digitale[http://www.factshunt.com/2014/01/total-number-of-websites-size-of.html].

Internet, nonostante il suo potenziale illimitato, è stato però solo il più determinante tra i key-enablers che hanno avviato l'era del Web 2.0.

Un altro fattore è stato, certamente, l'incremento del potere computazionale dei personal computer, che procede incessantemente, seppur con ritmi diversi, sin dalla nascita dei primi calcolatori. L'osservazione empirica di Moore del 1965[http://www.cs.utexas.edu/ fussell/courses/cs352h/papers/moore.pdf], secondo la quale il numero di transistor in un circuito integrato sarebbe raddoppiato approssimativamente ogni due anni, si è rivelata corretta per quasi mezzo secolo, tanto da divenire legge. Come era stato teorizzato, la crescita ha subito un rallentamento ed ora il numero di transistor raddoppia ogni tre anni: questi ritmi sono sufficienti a garantire, comunque, un incremento considerevole delle prestazioni. La figura 2.2 mostra l'aumento del

potere computazionale nelle ultime decadi[http://preshing.com/20120208/a-look-back-at-single-threaded-cpu-performance/].

Analogo ed essenziale è stato l'aumento della capacità di memorizzazione dei dispositivi in commercio. Negli anni '90 i computer possedevano hard disk con dimensione media di 120 MB e tale capacità sembrava ampiamente sufficiente per le esigenze di chi allora li utilizzava; nel 2001 sono arrivati ad avere una capacità media di 40 GB[http://arstechnica.com/business/2011/09/information-explosion-how-rapidly-expanding-storage-spurs-innovation/]; nel 2014, vengono venduti a prezzi modici hard disk di dimensione pari a 4 TB. Quello che stupisce non è l'incremento delle capacità di questi supporti, facilmente prevedibile osservando quanto accaduto per i processori, ma la facilità con cui questi dispositivi possano essere occupati in breve tempo, proprio a causa della enorme mole di dati a disposizione degli utenti. [[Inserire figura 2.3 con grafico]]

Per condividere tutte queste informazioni su Internet gli utenti hanno necessitato di una infrastruttura che fornisse loro un'adeguata velocità di connessione. Anche in questo caso lo sviluppo tecnologico ha fornito loro degli strumenti adeguati. Uno studio condotto da Jakob Nielsen nel 1998[http://www.nngroup.com/articles/law-of-bandwidth/] ha portato alla formulazione di una legge, simile a quella di Moore, che descrivesse bene il tasso di crescita della velocità di connessione, di seguito riportata nella forma sintetica:

Users' bandwidth grows by 50% per year (10% less than Moore's Law).

I dati raccolti negli ultimi 15 anni confermano la legge e nel 2013 si è raggiunta una velocità di connessione media di 58 Mbps, una larghezza di banda che consente di condividere contenuti a ritmi elevatissimi.

Infine, la diffusione di smartphone e di altri dispositivi portabili ha dato agli utenti la possibilità di condividere contenuti ovunque essi si trovino. Il risultato è stato un aumento dell'utilizzo giornaliero medio di Internet, che è passato dai 46 min/giorno del 2002 alle 4 ore/giorno del 2012[http://www.themainstreetanalys
growth-of-the-internet-over-the-past-10-years-infographic/].

L'incremento delle capacità dei computer, la diffusione di nuovi dispositivi mobili e la creazione di una rete che li connettesse tutti in modo veloce sono espressioni di un'equazione che ha come unica e scontata soluzione i big data.

1.3 Sorgenti di nuovi dati

Le sorgenti di informazioni che producono big data sono molte più di quelle facilmente immaginabili. Non siamo pienamente coscienti di quanto succede intorno a noi poiché i dispositivi tecnologici sono diventati parte della nostra

quotidianità: li utilizziamo in modo trasparente, senza renderci conto della loro presenza. Il risultato è che generiamo e consumiamo continuamente dati, ma non ce ne accorgiamo.

Gli utenti del web sono raramente consci di produrre nuovi dati con le loro operazioni: il più delle volte, infatti, non sono consapevoli dei processi che vengono avviati in server sparsi nel mondo in risposta alle loro azioni. Realizzano di stare generando nuove informazioni, invece, pubblicando video su YouTube (si stima che ogni minuto vengano caricate 100 ore di contenuti[http://blog.digitalinsights.in/social-media-users-2014-stats-numbers/05205287.html]), creando nuove pagine web o caricando i loro file su servizi di cloud storage come Dropbox o Google Drive. Non lo sono pienamente, al contrario, quando effettuano ricerche su motori di ricerca, comprano oggetti online, avviano un download o cambiano canale sul televisore. Ogni azione che coinvolge l'utilizzo di Internet produce nuovi dati, che vengono conservati ed analizzati, con insidie latenti per la privacy degli utenti[http://sozio-informatik.net/fileadmin/IISI/upload/2009/p265.pdf][http://www.pcworld.com/a
5-biggest-online-privacy-threats-of-2013.html] [[NSA?]].

Poiché i piccoli circuiti integrati sono diventati economici, è stato possibile aggiungerli ad ogni oggetto per dotarlo di una componente di intelligenza. Perfino le reti ferroviarie sono provviste di sensori, che permettono di monitorare traffico, condizioni meteorologiche, stato delle spedizioni e condizioni dei binari, in modo da prevedere ed evitare incidenti. Con questi dispositivi vengono costantemente raccolti voluminosi dati ambientali, finanziari, medici e di sorveglianza in tutto il mondo, ma contribuiscono a produrre molte informazioni anche giornali, riviste, sms, telefonate e sensori integrati nei dispositivi, come gli smartphone.

Una delle più grandi fonti di informazioni, tuttavia, si è costituita solo negli ultimi anni ed è di interesse specifico per questo lavoro: i social network. L'espansione capillare di questi siti web ha portato ad un aumento sensibile della mole di dati prodotti dall'uomo. A fronte dell'importanza di tali siti web per questo studio, si ritiene opportuno darne una descrizione più approfondita.

1.3.1 Social network

I social network sono siti che permettono agli utenti di costituire una rete di connessioni virtuali con altri individui, che siano amici o sconosciuti, e di restare in contatto con loro. Ciò che invoglia le persone ad utilizzarli è la facilità con cui essi potranno condividere i propri contenuti con i membri della loro rete. Tali contenuti sono spesso messaggi pubblici o privati, video,

foto e notizie personali: nuove informazioni che, nel complesso, costituiranno big data.

I tanti social network attualmente online si differenziano principalmente per il target di utenti cui si rivolgono e per la tipologia di contenuti che tendenzialmente raccolgono. Tutti condividono, però, alcune caratteristiche peculiari. Dinamiche condivise sono la registrazione di un profilo con le informazioni personali, la condivisione di nuovi elementi, l'instaurazione di nuovi legami e la visualizzazione di contenuti appartenenti ad altri individui.

Elencare i più famosi social network e descriverne le caratteristiche principali non è necessario ai fini di questo lavoro. Risulta anche inutile riportare tante statistiche riguardanti questi siti, poiché la maggior parte di esse non sarà più valida nel momento in cui qualcuno le leggerà. Per tale motivo ci si limiterà esclusivamente a descrivere Facebook e Twitter, due dei principali social network contemporanei, che sono stati oggetti di studio. [[Per merito della loro amplissima diffusione è stato possibile portare a termine diversi studi interessanti, analizzati nel corso del capitolo X, che hanno utilizzato informazioni raccolte dai profili degli iscritti a questi due siti.]]

Facebook, fondato nel 2004, rappresenta il più fulgido esempio di social network e si può affermare che esso stesso abbia contribuito alla definizione dei canoni propri di questi siti. Ogni iscritto a Facebook possiede un diario personale, con visibilità pubblica o limitata, sul quale può pubblicare aggiornamenti di stato, foto, video, note e link a pagine web. Chi vuole ha facoltà di aggiungere, tra le sue informazioni personali, dettagli inerenti luogo e data di nascita, preferenze musicali, situazione sentimentale, orientamento politico o religioso, ecc. Il lato social di Facebook consiste nella possibilità di aggiungere nuovi amici con cui condividere tutti o alcuni dei suddetti contenuti. Negli anni successivi sono state introdotte altre novità che hanno decretato il successo del social network, ad esempio la possibilità di utilizzare applicazioni, chattare con gli amici o creare gruppi di interesse, pagine (spesso utilizzate per attività commerciali) ed eventi. Facebook ha conosciuto una rapida espansione fino a diventare il secondo sito più visitato al mondo, secondo solo a Google[http://www.alexa.com/topsites]. Gli utenti attivi mensilmente su Facebook ammontavano a un milione nel 2004, a 145 milioni nel 2008 ed a 1,23 miliardi nel 2013[http://www.theguardian.com/news/datablog/2014/feb/04/facebook-in-numbers-statistics] (approssimativamente 1/7 degli abitanti del pianeta). Si stima che, nel 2012, ogni minuto venissero inviati 510000 commenti, 293000 aggiornamenti di stato e 136000 foto[http://thesocialskinny.com/100-social-media-statistics-for-2012/].

Più modesti, ma non per questo meno straordinari, sono i numeri che gravitano attorno a Twitter, nato nel 2006 ed attualmente in settima posizione nella classifica dei siti più visitati al mondo, subito dopo Wikipe-

dia e prima di Amazon[http://www.alexa.com/topsites]. Le dinamiche di interazione con il social network sono più limitate rispetto a quelle di Facebook. Gli utenti di Twitter - sono stimati 255 milioni di utenti attivi mensilmente[http://blog.digitalinsights.in/social-media-users-2014-stats-numbers/05205287.html] - utilizzano il sito principalmente per inviare messaggi, detti tweet, aventi una lunghezza massima di 140 caratteri. Sebbene sia possibile configurare la visibilità dei propri messaggi, Twitter viene usato solitamente senza filtri di sorta, lasciando visibili al mondo intero i propri tweet, diversamente da quanto avviene normalmente con Facebook, ove la visibilità è generalmente limitata agli amici. Nei tweet spesso vengono menzionati altri utenti o vengono inseriti dei tag, detti hashtag, che specificano il contenuto del messaggio o lo arricchiscono con informazioni o stati d'animo. Gli hashtag vengono usati per permettere al sistema di individuare e mostrare le tendenze che si delineano tra i circa 500 milioni di messaggi giornalieri [http://blog.digitalinsights.in/social-media-users-2014-stats-numbers/05205287.html].

Il risultato di queste incessanti attività è che ogni giorno Facebook produce oltre dieci TB di nuovi dati e Twitter altri sette. Se a questi si aggiungono i dati generati anche sugli altri social network allora risulta evidente quanto sia abbondante la sorgente di nuove informazioni. Con questi ritmi non sembrano esagerate le stime del volume di dati memorizzati nel mondo: 800000 PB nel 2000, 161 EB nel 2006, 35 ZB stimati per il 2020[IML1429USEN - IBM][http://www.emc.com/collateral/analyst-reports/expanding-digital-idc-white-paper.pdf]

1.4 Opportunità di business

Per le sfide che impongono, potrebbe sembrare che i big data siano un problema per chi si trova a doverli trattare. In realtà, se sapientemente elaborati, essi costituiscono un'occasione per considerevoli guadagni. Non a caso, infatti, le aziende che dispongono di più informazioni personali detengono più potere e ricchezza (Google, Amazon, eBay, Facebook, etc)[http://www.incomediary.com/30-richest-internet-entrepreneurs] e non deve stupire il fatto che sempre più imprese provino a comprare da loro dati con cui profilare gli individui.

Sembra ormai assodata l'uguaglianza tra big data e big power. In verità big data è sinonimo di big latent power, poiché questi dati richiedono un processo accurato di elaborazione, al fine di estrarre conoscenza utile per un dominio applicativo o esigenze di mercato.

Il motivo per cui i dati rappresentano una così importante fonte di guadagno si può facilmente immaginare: conoscere abitudini e preferenze delle persone significa poter proporre loro pubblicità mirate oppure prevedere

il loro comportamento ed agire di conseguenza. Emblematico è il caso di un discount che nel 2002 ha assunto uno statistico per provare a predire quali consumatrici fossero incinte, al fine di pubblicizzare prodotti specifici per i neonati. I proprietari erano convinti, infatti, che identificando una donna nel secondo trimestre della gravidanza avrebbero avuto modo di fidelizzarla per anni, prima vendendole pannolini, poi cereali, libri e DVD [http://www.nytimes.com/2012/02/19/magazine/shopping-habits.html].

La conoscenza degli utenti, comunque, è utilizzabile in diversi altri modi oltre al marketing personalizzato, ad esempio per dotare i sistemi di interfacce grafiche personalizzate per ogni individuo, per guidare scelte operative (quante scorte avere in magazzino di un determinato prodotto), campagne politiche, ecc.

Ciò che succede è che sempre più organizzazioni hanno accesso ad un gran numero di informazioni (pagine web, file di log, click-stream data, documenti, sensori), ma non sanno come valorizzarle. Una indagine dell'IBM [IML1429 - IBM] ha riscontrato come oltre metà dei business leader sentono di non avere gli strumenti adeguati per comprendere ciò che è necessario per la loro attività. Per tali ragioni, sempre più spesso, non sanno se conviene investire denaro e conservare questi dati (ammesso che abbiano i mezzi per farlo), nonostante siano sempre più numerosi gli strumenti che permettono di analizzarli per ottenere una visione più chiara del proprio business, dei clienti e del mercato.

Un altro problema che gli imprenditori devono affrontare è quello della sempre minore longevità dei dati. Ad oggi, infatti, non siamo più in grado di leggere gli 8-track tape, i floppy disk ed i VHS, rispettivamente di 30, 20 e 10 anni fa. La durata media della vita dei dispositivi di memorizzazione è sempre più breve e gli strumenti correlati diventano obsoleti. Un hard drive è progettato per durare 5 anni, un nastro magnetico 10 e CD/DVD approssimativamente 20, sebbene già ora inizino ad essere impiegati sempre meno, in favore dei Blu-ray Disc. Ci troviamo di fronte ad un paradosso per il quale le capacità di produrre e memorizzare dati aumentano, ma la capacità di conservarli nel tempo diminuisce. Moltissime aziende, infatti, hanno dovuto trascrivere i loro dati su nuovi supporti con una cadenza di 10-20 anni. Il fenomeno a cui si assiste è la crescita, per dimensione e complessità, dei dati a disposizione di una azienda a discapito della capacità di poterli analizzare. Col passare del tempo questo gap sembra aumentare in modo incontrollato: gli imprenditori che si doteranno di strumenti per colmare questo divario sono quelli che controlleranno in futuro il mercato.

Analizzare i big data prodotti dai propri utenti, tuttavia, non è l'unico modo che gli entrepreneur hanno a disposizione per monetizzare. Diverse sono le aziende, infatti, specializzate nello sviluppo di tecnologie per l'anali-

si di questi dati che vengono poi noleggiate a terzi per ottenere importanti guadagni. Anche senza possedere big data, pertanto, gli imprenditori possono sviluppare dei servizi da vendere a chi ne dispone. Tali servizi possono riguardare infrastrutture per la loro memorizzazione o tecnologie per poterli elaborare efficientemente, finanche in tempo reale. Diverse sono le redditizie piattaforme sviluppate da colossi dell'IT ed offerte alle imprese; alcune delle principali sono di proprietà di Amazon, Google, IBM e Microsoft. Tralasciando queste tecnologie, diverse aziende vendono strumenti per l'analisi ed infrastrutture.

1.5 Criticità nella gestione

La natura dei big data, come è stato anticipato, solleva diverse problematiche risolubili solo con particolari tecnologie. Ogni azienda possiede un numero di informazioni variabile ed è interessata ad elaborare i propri dati per conseguire uno specifico obiettivo, pertanto i processi che essi subiranno cambieranno in ogni contesto. È possibile, tuttavia, individuare due criticità comuni a tutti i casi d'uso: l'elaborazione e la memorizzazione dei dati.

Nel momento in cui si deve progettare un sistema avente come obiettivo l'analisi di grandi moli di informazioni bisogna decidere con quali modalità sarà eseguita la computazione e in che modo saranno memorizzati i dati. I criteri che dovrebbero orientare il progettista verso una scelta consona sono due: la natura delle operazioni che si desidera effettuare e la loro frequenza.

In base alle esigenze proprie del dominio, bisognerà trovare perciò la soluzione più adatta a risolvere i quattro problemi tipici della gestione di big data: come avviene l'elaborazione e come la memorizzazione, dove avviene l'elaborazione e dove la memorizzazione.

Nel corso dei capitoli seguenti saranno presentati nel dettaglio dei framework e degli strumenti progettati per risolvere questi problemi e che vengono tuttora frequentemente impiegati in molteplici scenari; adesso ci si limiterà a presentare sommariamente, per ogni problema, i due principali approcci tra cui scegliere.

Il primo problema riguarda come effettuare le operazioni sui dati. Questa scelta ha anche un impatto diretto sul paradigma di programmazione adottato in fase di scrittura del codice del programma preposto all'elaborazione dei dati. È possibile seguire un approccio classico ed elaborare i dati in modo centralizzato, su un unico server, oppure cercare una soluzione più scalabile, optando per una modalità di elaborazione distribuita tra più nodi di una rete. Questo secondo approccio è quello più adottato e, per tale motivo, i framework che consentono una ripartizione automatica del carico di lavoro tra

computer interconnessi hanno conosciuto una rapida evoluzione. Un esempio è offerto da Apache Hadoop, che sarà approfondito nel capitolo successivo.

Per quanto concerne le modalità di memorizzazione delle informazioni, è possibile scegliere tra la consolidata soluzione offerta dai database relazionali oppure scegliere sistemi innovativi per conservare i dati. Nel corso del capitolo 3 sarà effettuato uno studio comparativo dei database NoSQL, una nuova tipologia di database che meglio si adatta a gestire big data.

Le soluzioni al problema riguardante l'ubicazione del processo di elaborazione sono riconducibili a due tipologie principali: effettuare la computazione in locale su uno o più server propri oppure percorrere la via del cloud computing, delegando l'elaborazione a server remoti. Come detto nel paragrafo precedente, diverse aziende come Amazon e Google offrono dei servizi con cui è possibile elaborare efficientemente grandi moli di dati. In seguito verrà dimostrato come essi costituiscano un'opzione valida ed economicamente vantaggiosa rispetto all'effettuare il processo di elaborazione su macchine in locale.

Le opzioni appena illustrate per risolvere il problema dell'elaborazione sono anche valide per quello della memorizzazione. È possibile conservare i dati su server propri in locale oppure optare per il cloud storage, trasferendo i propri file in rete. Anche per questo scopo è possibile trovare online servizi di noleggio di spazio virtuale ove caricare i propri dati e renderli disponibili in qualunque momento e da qualunque parte del globo.

Tutte le varie possibilità presentate offrono ai progettisti ampia libertà di scelta. Essi potranno propendere per una architettura interamente cloud oppure per soluzioni ibride, che impiegano servizi remoti solo per soddisfare alcune necessità, delegando ai propri server il resto delle operazioni.

Progettare correttamente la struttura del sistema risulta fondamentale quando vengono elaborati big data, poiché è alta la probabilità di incappare in bottleneck che potrebbero far degradare le performance del sistema. Ulteriore attenzione va posta se si adoperano soluzioni cloud, poiché vanno studiati attentamente i costi di questi servizi al fine di valutare l'effettiva economicità dell'opzione.

Capitolo 2

Elaborazione distribuita: Apache Hadoop

2.1 Descrizione del framework

Un punto di svolta fondamentale nello sviluppo di tecnologie orientate ai big data è stata la pubblicazione, ad opera di Google, di alcuni articoli in cui venivano illustrati i principi di funzionamento di alcuni strumenti sviluppati ed utilizzati dal colosso per far fronte alle proprie gravose esigenze. Il paper sul Google File System del 2003[] e quello su MapReduce del 2004[], più degli altri, hanno radicalmente cambiato il modo di pensare ai dati ed hanno avviato lo sviluppo di molte nuove tecnologie, tra cui Hadoop.

Hadoop è un progetto top-level della Apache Software Foundation, scritto in Java da una community globale di contributori. Lo sviluppo del framework è cominciato nel 2004 come sotto-progetto di Lucene, una libreria per l'information retrieval, da cui poi si è distaccato per divenire un progetto a sé stante. Hadoop ha avuto un'importante evoluzione dal 2005, sotto la spinta di Yahoo!, interessato a realizzare una tecnologia che potesse competere con quella sviluppata da Google. Nell'articolo del 2004 viene descritto MapReduce, un modello di programmazione ed un omonimo framework scritto in C++ progettati per eseguire in parallelo programmi su cluster di computer. Il framework di Google, di cui Hadoop è una riproduzione open-source, si preoccupa di distribuire la computazione tra i nodi del cluster, gestire i malfunzionamenti delle macchine, bilanciare i carichi di lavoro ed accorpare i risultati prodotti da ciascun nodo prima di restituirli in output all'utente. In questa maniera uno sviluppatore può concentrarsi sulla scrittura del codice del proprio programma, anche senza avere esperienza di computazione parallela, poiché essa sarà gestita automaticamente dal framework. Il codice

scritto, che deve aderire al modello di programmazione MapReduce (descritto nel dettaglio nel paragrafo 3.3), risulta lineare, poiché non contiene sezioni orientate a gestire meccanismi di parallelizzazione o recupero da situazioni di errore.

In questa maniera Hadoop diventa lo strumento ideale, talvolta l'unica opzione possibile, per elaborare grandi moli di dati, perché permette di ripartire pesanti carichi di lavoro tra tante macchine di un cluster, ciascuna delle quali processa solo una ridotta frazione dei dati di input. Per tal motivo, moltissime aziende[http://wiki.apache.org/hadoop/PoweredBy] che trattano big data, quali Yahoo!, Facebook, Twitter, eBay, LinkedIn e Spotify, impiegano cluster con Hadoop per effettuare l'elaborazione dei propri dati. Intorno ad Hadoop si sono poi sviluppati numerosi altri progetti, tra cui HBase, Pig, Hive e Spark, che vengono per convenzione inclusi nella "piattaforma Hadoop". Questi progetti, generalmente, utilizzano Hadoop nel loro core per aggiungergli delle caratteristiche o realizzare tecnologie più complesse, come database o strumenti di analisi.

Nella sua versione iniziale Hadoop era progettato per l'elaborazione batch dei dati, ma ora ci sono diversi servizi che consentono di adoperarlo in applicazioni real-time (stream-processing, real-time query, ecc.).

I moduli principali che compongono il framework dell'ultima versione (2.5.0) sono: Hadoop Common, una raccolta di strumenti impiegati da diversi moduli, Hadoop Distributed File System (HDFS), il file system presentato nel dettaglio in seguito, Hadoop YARN, un framework per il job scheduling e la gestione delle risorse del cluster e Hadoop MapReduce, il sistema basato su YARN per elaborare in parallelo grandi dataset.

Nonostante sempre più strumenti vengano sviluppati per elaborare in parallelo i dati, Hadoop rimane una tecnologia largamente impiegata nelle imprese dell'IT. A conferma di quanto detto, diverse sono le importanti aziende che offrono software o servizi basati sul framework: alcuni esempi sono Cloudera, MapR e Hortonworks, aziende specializzate nello sviluppo di piattaforme e strumenti per l'analisi di big data.

Nei paragrafi seguenti saranno descritti l'Hadoop Distributed File System ed il modello di programmazione MapReduce, essenziali per comprendere i principi di funzionamento del framework.

2.2 Hadoop Distributed File System

Una componente essenziale del framework è il file system che tutti i nodi del cluster utilizzano: l'Hadoop Distributed File System (HDFS).

La necessità di sviluppare un file system specifico per Hadoop derivava da due esigenze avvertite, sin dalle fasi iniziali dello sviluppo, dai contributori del progetto. Il primo problema da risolvere era causato dalla natura distribuita del framework: ogni nodo del cluster, infatti, per eseguire i compiti assegnati, doveva poter accedere ai dati di input del programma. La seconda problematica, invece, era legata alla tipologia di questi dati di input: big data, non memorizzabili fisicamente su una solo macchina, coerentemente con la definizione data nel paragrafo 2.1. Replicare i dataset su ogni nodo, pertanto, risultava impossibile.

L'Hadoop Distributed File System risolve questi ed altri problemi adoperando soluzioni architetturali già presenti in altri file system distribuiti ed ignorando alcuni vincoli POSIX per ragioni di efficienza, similmente al Google File System (GFS). Una assunzione che viene fatta, ad esempio, è che i dati del cluster siano soggetti a batch processing e pertanto il modello adottato è di tipo write-once-read-many. Per tale motivo, una volta creati, i file non possono essere modificati (nel GFS, invece, è possibile eseguire operazioni atomiche di append): questo, sebbene rappresenti un grande limite, consente di ottenere un elevato throughput.

L'HDFS gestisce i file in modo gerarchico, come un file system tradizionale, utilizzando cartelle e path, con la differenza che i dati non vengono memorizzati fisicamente in un unico luogo, ma vengono ripartiti tra i nodi di un cluster. I file, infatti, vengono suddivisi in chunk di 64 MB (l'ultimo chunk può avere dimensione inferiore) e distribuiti tra i nodi, che, in questo modo, si dividono l'onere di conservare grandi moli di dati. Il file system è progettato per essere avviato su grandi cluster con commodity hardware, dove i malfunzionamenti non sono un caso, quanto la norma (crash delle macchine, rottura dei drive, errori di rete, ecc.). Per tale motivo, al fine di garantire sempre la disponibilità dei dati, i chunk vengono replicati (il fattore di replicazione di default è 3, ma è configurabile).

A livello architetturale l'HDFS presenta una struttura master/slave, illustrata in figura 3.X. Un unico nodo master, il NameNode, si preoccupa di gestire il file system, conservando l'elenco dei file memorizzati e tenendo traccia della loro ubicazione nel cluster attraverso metadati. Tutti gli altri nodi slave del cluster, invece, ricoprono il ruolo di DataNode e conservano i chunk che sono loro assegnati dal NameNode. Quest'ultimo, conoscendo la locazione dei chunk, può indirizzare le richieste del client ai DataNode opportuni, che si occuperanno di leggere i dati ed inviarli al programma che li consumerà. Il NameNode si preoccupa, inoltre, di eseguire operazioni quali la rinomina o la cancellazione dei file, ma anche di verificare che i chunk siano integri (attraverso il checksum) e che siano sufficientemente replicati (in caso negativo avvia la loro copia su nuovi DataNode).

Diversi accorgimenti permettono al NameNode di mantenere un indice dei chunk sempre aggiornato per fornire dati consistenti a chi legge dal file system. Esso conserva in memoria, innanzitutto, un file contenente le proprietà del sistema ed il mapping tra i blocchi dei file ed i nodi del cluster. Tale file, detto FsImage, viene periodicamente serializzato ed occupa poco spazio (un NameNode con 4 GB di RAM supporta un numero enorme di file e directory). Al suo avvio, il NameNode carica l'ultima FsImage ed interroga tutti i DataNode del cluster che gli forniscono una risposta, detta BlockReport, contenente l'elenco dei chunk da loro attualmente conservati; combinando le informazioni ottenute, il nodo master ottiene una rappresentazione generale del sistema e provvede a verificare che ogni file sia aggiornato e sufficientemente replicato. Tutte le azioni eseguite dal NameNode sul file system, inoltre, vengono memorizzate e replicate in un file di log, detto EditLog, che consente, in caso di fallimento del nodo master, di ripristinare al suo riavvio l'ultima configurazione del file system. Il nodo master, infatti, ricarica l'ultima FsImage e riesegue su di essa le ultime operazioni andate perdute. Per tale motivo, una volta serializzata una nuova FsImage, l'EditLog può essere svuotato, poiché le ultime modifiche sono conservate nell'ultima immagine del sistema.

L'HDFS non supporta gli snapshot per memorizzare una copia dei dati ad un certo istante temporale - il GFS, invece, beneficia di questa funzionalità – e non dispone di un meccanismo per riavviare automaticamente il NameNode in caso di malfunzionamenti. Quest'ultimo, inoltre, rappresenta un single-point-of-failure del sistema, poiché è l'unico a poter eseguire operazioni sul file system. A fronte di questi svantaggi, tuttavia, l'HDFS presenta altri punti di forza, tra cui una indiscutibile semplicità di accesso. Esso può essere utilizzato dalle applicazioni in diverse maniere; espone infatti una API Java ed un wrapper in C di tale API, consente la navigazione tramite browser mediante il protocollo HTTP e la shell presenta la maggior parte dei comandi tradizionali riguardanti i file con semantica intuitiva. Un'altra interessante caratteristica del framework riguarda la capacità di suddividere in modo ottimale le repliche dei chunk tra i DataNode per ottimizzare le performance in fase di lettura. L'HDFS, infatti, in linea col principio secondo cui "moving computation is cheaper than moving data", cerca di minimizzare lo spostamento di dati sulla rete, distribuendo in modo omogeneo i blocchi di file tra i vari rack del cluster, per consentire ai client di avere sempre a brevi distanze dei nodi contenenti le informazioni a loro necessarie.

2.3 Modello di programmazione MapReduce

In questo paragrafo vengono descritti i principi fondamentali del modello di programmazione MapReduce, alla base sia del framework di Google, che di quello di Apache; per tale motivo, durante la descrizione, ci si riferirà indistintamente ai due prodotti, poiché le differenze che sussistono riguardano principalmente dettagli realizzativi e non strutturali.

Per semplificare il processo di elaborazione dei dati avvantaggiandosi del framework, i programmi da eseguire sul cluster devono essere costituiti da due operatori: map e reduce. Nonostante le tipologie di computazione da effettuare sui dati siano di natura diversa, la maggior parte di esse può essere realizzata impiegando queste due sole funzioni, come spiegato anche nell'introduzione di Google su MapReduce:

We designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the map and reduce primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a map operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a reduce operation to all the values that shared the same key, in order to combine the derived data appropriately.

Un programma che implementa correttamente le funzioni map e reduce è in grado di essere eseguito dal framework in modo distribuito: il programmatore può, perciò, concentrare la sua attenzione esclusivamente sulla scrittura di tali operatori.

La funzione map riceve in input una coppia chiave/valore (K, V) e produce in output un insieme di coppie intermedie chiave/valore (I, P). Il framework raggruppa automaticamente le coppie intermedie aventi stessa chiave I e inserisce tutti i corrispondenti valori P in una lista. La funzione reduce riceve in input una di queste coppie (I, [P1, P2, P3, ...]) generate dal framework e produce un insieme, possibilmente ridotto, di tali valori. È possibile schematizzare le funzioni nel modo seguente:

map (K, V) -> list(I, P)

reduce (I, list(P)) -> list(P)

I tipi delle chiavi e dei valori intermedi sono solitamente quelli supportati dal linguaggio di programmazione del framework; generalmente sono sufficienti interi, double, long e stringhe per la maggior parte dei casi d'uso.

Il framework assegna ai nodi del cluster l'esecuzione di una o più funzioni map/reduce e ne raccoglie i risultati (per i dettagli sul funzionamento si rimanda al paragrafo successivo). Ciascun nodo possiede il codice di entrambi

gli operatori e, su richiesta del framework, esegue la funzione che gli viene richiesta sui dati di input che riceve.

Per chiarire quanto appena spiegato viene presentato lo pseudo-codice delle funzioni map e reduce che si potrebbero scrivere per indicizzare un documento:

map(int lineNumber, String line): for each word w in line: EmitIntermediate(w, 1);

reduce(String w, List<Integer> values): Emit(w, length(values));

Il programma conta, per ogni parola, il numero di occorrenze nel documento. La funzione map riceve in input una coppia avente per chiave un numero naturale rappresentante la riga i-esima del documento e per valore una stringa uguale alla riga i-esima. L'operatore itera su ogni parola w della riga e produce in output una coppia (w, 1), a significare che la parola w è stata incontrata una volta. Al termine dell'esecuzione di tutte le funzioni map, il framework si occuperà di accorpare le coppie aventi stessa chiave prodotte dai vari nodi del cluster: per ogni parola w del documento verrà creata una coppia avente come chiave w e come valore una lista dei valori intermedi a lei abbinati (in questo caso una lista di numeri 1, provenienti dalle funzioni map eseguite su diversi nodi). Le coppie di input per la funzione reduce saranno pertanto della forma (w, list(1, 1, ..., 1)), che si limiterà a controllare la dimensione della lista per restituire in output coppie del tipo (w, #occorrenze).

Le funzioni potrebbero essere schematizzate così:

map (int, String) -> list(String, int)

reduce (String, list(int)) -> (String, int)

Come è possibile notare, il modello di programmazione MapReduce permette di scrivere semplici funzioni, delegando al framework l'onere di gestire il calcolo in parallelo tra i nodi. Le funzioni map e reduce illustrate permettono di indicizzare agevolmente dataset dell'ordine di PB, a patto di avere cluster di dimensione appropriata.

2.4 Principi di funzionamento del framework

Nel corso degli anni Hadoop è stato protagonista di una grande evoluzione, che ha causato, talvolta, modifiche sostanziali al framework nel passaggio da una versione all'altra. Sebbene il core sia rimasto immutato, diverse altre componenti sono cambiate radicalmente. Per tale motivo, nell'illustrare i principi di funzionamento di Hadoop, sarà scelta una versione di riferimento, la 0.18. I componenti fondamentali di questa release sono rimasti pressoché

immutati fino alla versione 2 del framework, quando sono state introdotte molte novità, tra cui YARN.

Come si è potuto evincere, un cluster di Hadoop è composto da diversi nodi, ciascuno dei quali ricopre uno specifico compito. Tali ruoli vengono assegnati alle macchine in fase di configurazione del cluster dal manager del sistema, che deve avere pertanto una conoscenza profonda dell'architettura del framework: una corretta configurazione, infatti, è essenziale per ottenere elevate prestazioni. Sebbene scrivere programmi secondo il modello MapReduce sia relativamente semplice, configurare correttamente il cluster può risultare laborioso per chi si approccia al framework per la prima volta. Per questo motivo sono stati sviluppati degli strumenti per effettuare automaticamente il tuning dei parametri di configurazione (ad esempio Starfish) e script personalizzabili per eseguire il deploy del cluster (ad esempio quello offerto dalla Google Cloud Platform).

Eseguire un cluster di Hadoop significa avviare su ciascun nodo uno o più daemon, che specificano il ruolo del nodo nella rete. In generale, è possibile suddividere i daemon in due classi: quelli di computazione e quelli di memorizzazione. I daemon di computazione sono il JobTracker ed il TaskTracker, mentre quelli di memorizzazione sono il NameNode, il Secondary NameNode ed il DataNode.

Il JobTracker è il punto di collegamento tra l'applicazione scritta dall'utente ed il framework. Una volta inviato il proprio programma ad Hadoop, il JobTracker fissa un piano di esecuzione, assegna ai nodi disponibili dei task da completare e si preoccupa di raccogliere e ridirezionare i risultati intermedi della computazione. Il daemon, inoltre, controlla se i nodi lavoratori stiano correttamente funzionando interrogandoli periodicamente; se non riceve risposte da un nodo, il JobTracker assume che sia incappato in una situazione di errore e riassegna il task ad un nuovo nodo inattivo. Nel cluster vi è solo un JobTracker ed è tipicamente il master node del cluster. Esso rappresenta un single-point-of-failure (con YARN, invece, il ResourceManager non è affetto da questa criticità). Generalmente non si eseguono TaskTracker sul nodo dove è in esecuzione il JobTracker al fine di non sovraccaricare ulteriormente la macchina e non incorrere in degradi di prestazioni.

Il TaskTracker è uno slave node e si occupa di eseguire i task map e/o reduce assegnatigli dal JobTracker, cui spedisce i risultati elaborati. Sebbene ci sia al più un solo daemon TaskTracker in esecuzione su ogni nodo, ciascuno di essi può avviare istanze multiple della Java Virtual Machine, in modo da iniziare diversi task in parallelo. Il TaskTracker, mentre svolge i compiti, informa periodicamente il JobTracker della sua attività.

Tra i daemon di memorizzazione, NameNode e DataNode sono stati già esaustivamente descritti nel paragrafo sull'HDFS. Resta da analizzare il Se-

condary NameNode, sebbene il suo nome sia eloquente. Il ruolo di questo daemon, infatti, è di assistenza al NameNode, che è un single-point-of-failure, per monitorare lo stato del cluster. Come per il NameNode, ogni cluster ha un solo Secondary NameNode, che risiede generalmente su una macchina dedicata, senza altri daemon in esecuzione. Esso non riceve in tempo reale aggiornamenti sulla situazione dell'HDFS, ma comunica ad intervalli configurabili con il NameNode per effettuare snapshot dei suoi metadati. Le informazioni che il Secondary NameNode memorizza possono aiutare a minimizzare i tempi di downtime dovuti a fallimenti del NameNode principale, poiché è possibile, previo intervento manuale, promuovere il Secondary NameNode a NameNode per riattivare il cluster.

Su ciascun nodo slave sono solitamente in esecuzione sia il DataNode che il TaskTracker per ripartire sia il carico di lavoro che quello di memorizzazione. Per ottenere buone prestazioni, Hadoop è in grado di assegnare i task di map e reduce in modo da sfruttare la località dei dati ed evitare inutili trasferimenti di informazioni nella rete. Il framework, perciò, assegna i job ai TaskTracker in esecuzione sulle macchine più vicine ai DataNode contenenti i dati di input necessari.

2.4.1 Modalità di esecuzione

Hadoop può essere eseguito in tre modalità differenti: locale, pseudo-distribuita e distribuita.

La modalità locale, impiegata in default da Hadoop, necessita di una sola macchina e non di un cluster. In questa modalità non vengono avviati né i daemon di Hadoop precedentemente illustrati né l'HDFS, poiché non ci sono forme di comunicazione tra i nodi: i dati ed il programma sono residenti sulla stessa macchina, che esegue un programma in modo tradizionale. Questa modalità è spesso impiegata in fase di sviluppo e di debug, poiché permette di verificare la correttezza della logica di un'applicazione MapReduce, senza trattare con la complessità addizionale dovuta all'interazione con i daemon.

Anche la modalità pseudo-distribuita richiede una sola macchina ed equivale ad eseguire un cluster con un solo nodo. Su questo nodo vengono eseguiti tutti i daemon, inclusi quelli dell'HDFS, perciò vi è comunicazione tra le istanze della Java Virtual Machine. Come per la modalità locale, anche quella pseudo-distribuita ha finalità di debug. In modo particolare è utile per monitorare l'utilizzo della memoria, i problemi nei flussi di input/output dell'HDFS e, soprattutto, l'interazione tra i daemon.

Vi è infine la modalità distribuita, impiegata quando la correttezza del programma è stata appurata mediante opportuni test con le altre modalità.

In questa modalità diversi nodi formano un cluster e su ciascuno vengono avviati i diversi daemon, che interagiscono tra loro tramite SSH.

2.4.2 Ottimizzazione delle performance

Le informazioni finora fornite dovrebbero permettere una comprensione generale dell'architettura del framework, ma ci sono alcuni altri elementi che vale la pena di approfondire.

Un punto cruciale nell'esecuzione di applicazioni MapReduce è il passaggio dal completamento dei job di map all'avvio di quelli di reduce. Da quanto finora detto, sembrerebbe che sia Hadoop a gestire autonomamente questo momento della elaborazione. In effetti questo è il comportamento di default del framework, che, in modo trasparente all'utente, raccoglie in una lista i valori intermedi appartenenti ad una stessa chiave ed avvia funzioni di reduce sulle coppie generate. In realtà lo sviluppatore può alterare le azioni che saranno eseguite al termine dei job di map per ottenere significativi aumenti nelle performance. Oltre alle classi Mapper e Reducer, che contengono, rispettivamente, le funzioni map e reduce e che devono tassativamente essere scritte, lo sviluppatore può definire un Combiner ed un Partitioner personalizzati. Se queste due classi non vengono configurate allora il framework adopera quelle di default, che impiegano funzioni di hash per ripartire i dati in modo omogeno tra i nodi.

Il Combiner ha un ruolo simile al Reducer, poiché aggrega i valori delle coppie intermedie. Mentre il Reducer compatta le coppie provenienti da nodi differenti, il Combiner agisce su un singolo TaskTracker, prima che invii nel cluster i risultati intermedi prodotti dalla funzione map. Il Combiner ed il Reducer, in generale, possono aggregare dati in modo differente, secondo le esigenze specifiche del caso d'uso. Aggregare i dati su ciascun nodo, prima che vengano trasmessi nel cluster può ridurre le operazioni di input/output migliorando il throughput generale. A dimostrazione di ciò, viene ora arricchito il precedente esempio per l'indicizzazione di un documento introducendo un Combiner adatto, che sfrutta la proprietà associativa dell'addizione.

Il Mapper precedente resta invariato, mentre il Combiner aggiunto si limitar a sommare localmente il numero di occorrenze per ciascuna chiave (parola) prima di restituirlo in output. L'output del Mapper, perciò, non saranno più tante coppie (wn, 1), ma coppie del tipo (wn, #occorrenze_nella_riga). A fronte di questo cambiamento, il codice del Reducer deve essere lievemente modificato, poiché i valori associati ad una chiave non saranno più liste di 1, ma liste di interi probabilmente diversi tra loro. Il numero di occorrenze di una parola non sarà più dato dalla lunghezza della lista (un 1 per ogni match), ma dalla somma dei valori nella lista:

reduce(String w, List<Integer> values): int count = 0; for each value v in values: count = count + v; Emit(w, count);

Aggregando parzialmente le occorrenze in ogni TaskTracker, il numero di coppie immesse nella rete diminuisce drasticamente. Senza un Combiner, ogni riga del documento data in input ai Mapper produceva tante coppie quante le parole che conteneva; mediante il Combiner, invece, viene emessa una coppia in meno per ogni parola duplicata. Considerando che un documento contiene moltissime parole duplicate, il numero di coppie trasferite diminuisce sensibilmente aumentando le performance generali.

Resta da approfondire il Partitioner, che ha il compito di ripartire le coppie prodotte dai Mapper - ed eventualmente aggregate dai Combiner - tra i nodi del cluster che eseguiranno funzioni di reduce. Il Partitioner di default è l'HashPartitioner ed ha un ruolo fondamentale, poiché serve ad evitare bottleneck nella fase conclusiva dell'applicazione: se tutte le coppie intermedie, infatti, confluissero in un unico TaskTracker esso impiegherebbe una grande quantità di tempo per completare la fase di reduce, nullificando i benefici guadagnati dell'elaborazione in parallelo nella fase di map. Un Partioner preposto alla suddivisione di chiavi di tipo stringa, ad esempio, potrebbe suddividerle in base alla loro lunghezza o al loro primo carattere.

Altre due caratteristiche peculiari di Hadoop meritano un approfondimento: la Streaming API e la DistributedCache.

Hadoop può interagire con altri linguaggi attraverso una API generica, chiamata Streaming. Questa è particolarmente utile per la scrittura di semplici e brevi programmi MapReduce che possono essere sviluppati più facilmente in linguaggi di scripting che non hanno bisogno di librerie Java. Hadoop Streaming interagisce con gli altri programmi usando il paradigma Unix: l'input arriva dallo STDIN ed è direzionato in uscita verso lo STDOUT. I comandi Unix funzionano con il framework, che permette di impiegarli come funzioni map o reduce. In questo modo è possibile utilizzare, ad esempio, cut, uniq, wc oppure script in Python o PHP come parametri nella configurazione dei task di Hadoop.

[ESEMPIO IN PYTHON...]

Una ultima caratteristica di Hadoop meritevole di essere illustrata è DistributedCache, un semplice ma utile meccanismo che consente di replicare automaticamente alcuni file tra tutti i nodi del cluster. DistributedCache viene solitamente impiegato per fornire, spesso all'accensione, tutte le macchine del cluster di file contenenti "dati di background", spesso richiesti dai mapper. Può essere utilizzato, ad esempio, per distribuire file di configurazione, script o documenti contenenti metadati. Nell'ormai noto esempio di indicizzazione del testo, ad esempio, la funzione di map potrebbe necessitare di un tokenizer per dividere una stringa in parole. Se il tokenizer scelto fosse

incluso in una libreria esterna, ad esempio, essa potrebbe essere distribuita tra tutti i nodi del cluster tramite la funzionalità di DistributedCache. Tutte le funzioni map potrebbero, pertanto, utilizzare quella libreria, perché certamente presente sul nodo corrente.

Capitolo 3

Memorizzazione flessibile: database NoSQL

3.1 Origini delle tecnologie

Memorizzare negligentemente big data è il presupposto fondamentale per poterli in seguito elaborare efficientemente. Come detto nel secondo capitolo, il problema che sorge in fase di memorizzazione riguarda non soltanto lo spazio fisico che essi richiedono - col passare del tempo gli hard disk diventano sempre più capienti ed economici - ma anche le modalità con cui essi vengono archiviati. Una scelta progettuale sbagliata, infatti, può comportare una incapacità di recuperare questi dati velocemente, causando problemi di entità variabile col contesto d'uso (si pensi alla criticità nei sistemi real-time o a bottleneck in fase di lettura). Quando bisogna trattate big data è richiesto un consistente sforzo ingegneristico per sviluppare ex novo soluzioni che ripartiscano l'onere di memorizzazione tra diverse macchine, in un modo simile a quanto illustrato con l'Hadoop File System nel capitolo 2.2. Infatti, sebbene sia facile aumentare la capacità di memorizzare dati comprando nuovi drive, risulta comunque impensabile concentrare tutto questo potenziale su un unico server.

L'esigenza di ripartire il carico di lavoro senza degradare nelle performance ha reso inadatti i database tradizionali, che utilizzano il modello relazionale per la strutturazione dei dati. Quando questi database sono stati progettati negli anni '70, infatti, Internet era ancora in uno stato embrionale: non esistevano social network, smartphone e big data e, soprattutto, non si avvertiva l'esigenza di sviluppare sistemi che potessero scalare. Ma il Web 2.0 ha generato nuovi bisogni che i database NoSQL hanno provato a soddisfare.

Il termine "NoSQL" include un'ampia gamma di tecnologie ed architettu-

re per i dati, sviluppate in risposta all'aumento del volume delle informazioni e della frequenza con cui queste sono letti. È stato usato per la prima volta da Carlo Strozzi nel 1998[] per definire il suo database che non esponeva la tipica interfaccia di interrogazione tramite Structured Query Language (SQL). Come egli stesso affermò, sarebbe stato più appropriato definire il database NoREL, dal momento che la differenza principale del suo sistema rispetto ai precedenti era nel modello di memorizzazione dei dati e non tanto nel linguaggio di interrogazione, l'SQL. Col passare degli anni, tuttavia, il termine NoSQL si è affermato e viene ora impiegato per etichettare i database che non adoperano il modello relazione, a prescindere dal linguaggio delle query. È possibile trovare, pertanto, database NoSQL interrogabili sia tramite API proprie sia tramite SQL; un discorso analogo vale per i database relazionali. Nella grande maggioranza dei casi, tuttavia, i database NoSQL espongono API proprie, mentre quelli relazionali supportano l'SQL.

Per questi motivi l'acronimo viene spesso risolto come "Not Only SQL", ad indicare, più che una tecnologia in particolare, una scuola di pensiero che propone un nuovo approccio alla gestione di grandi dati e alla progettazione di database.

3.2 Confronto con database relazionali e NewSQL

I database NoSQL non sono stati progettati per sostituire definitivamente quelli relazionali, ma per offrire ai progettisti una valida alternativa da considerare durante la progettazione di sistemi che coinvolgono grandi moli di dati.

I database relazionali, infatti, sono tuttora adatti per molte applicazioni e spesso incontrano le attuali esigenze di business. Inoltre sono semplici da utilizzare e sono supportati da un vasto ecosistema di strumenti creati appositamente per loro, come tool di analisi e visualizzazione, di ottimizzazione e di gestione. Esistendo in commercio da molti anni, inoltre, hanno potuto beneficiare di una più lunga evoluzione ed il risultato sono sistemi robusti e performanti, a lungo collaudati da milioni di utilizzatori nel corso del tempo. Per lo stesso motivo sono anche conosciuti da un più ampio gruppo di persone, che ne padroneggia le best practice, rendendo semplice la ricerca di personale qualificato nelle aziende. I database NoSQL, invece, sono ancora poco diffusi[grafico diffusione NoSQL] ed il numero di persone con skill in questo campo è ancora ridotto, nonostante la domanda del mercato sia in continua crescita[].

Confrontare i database relazionali con quelli NoSQL dal punto di vista economico non è rilevante: per entrambi è possibile trovare soluzioni a pagamento ed altre gratuite. Allo stesso modo è possibile individuare opzioni open-source o proprietarie.

Tutti i vantaggi dei database relazionali elencati, tuttavia, risultano di poco conto quando l'applicazione richiede di scalare e di maneggiare dataset in evoluzione, che non aderiscono a schemi prefissati. I database NoSQL, nonostante si siano cominciati a diffondere solo dagli anni 2000, sono dotati di un architettura che garantisce, oltre a buone performance, flessibilità e scalabilità, a fronte di compromessi accettabili.

3.2.1 Flessibilità del modello dei dati

I database relazionali sono progettati per trattare dati strutturati, ovvero dati aderenti ad uno schema predefinito, in cui le informazioni sono suddivise in campi, ciascuno dotato di un tipo. Questi dati sono perciò facilmente memorizzabili in tabelle, contenenti anche milioni di righe.

I database NoSQL, invece, non richiedono la definizione a priori di uno schema fisso per una tabella, comune a tutti i dati che vi apparteranno; per questo motivo vengono definiti schema-less. Questa caratteristica è ideale per uno sviluppo agile, con rapide iterazioni e aggiornamenti frequenti del codice e del sistema. Come sarà spiegato nel dettaglio in seguito, ogni oggetto memorizzato nel database può avere dei propri campi, non necessariamente condivisi con gli altri. Durante il ciclo di vita del database, pertanto, a ciascun oggetto potranno essere aggiunti o rimossi campi all'occorrenza con grande facilità.

A differenza dei database relazionali, inoltre, consentono di memorizzare, oltre ai dati strutturati, anche dati semi-strutturati (come i file XML o JSON) e non strutturati (file multimediali o testi). I database NoSQL, infatti, non utilizzano un modello di dati statico che prevede la memorizzazione di ogni record come una tupla di una tabella, ma modelli dinamici e variabili, approfonditi nel capitolo 4.X.

Nei database relazionali la struttura delle tabelle può essere modificata, ma l'operazione è sconsigliata poiché particolarmente esosa, specialmente quando applicata su tabelle con milioni di righe. Questa operazione, inoltre, può essere effettuata solo mentre il database è offline, comportando ovvi problemi.

3.2.2 Scalabilità delle architetture

La maggiore differenza tra i database NoSQL e quelli relazionali, però, riguarda la loro capacità di scalare, ovvero la capacità di gestire efficientemente carichi di lavoro crescenti senza degradare nelle performance, adattando dinamicamente la struttura del sistema alla crescita di richieste. Prima di analizzare i due tipi di database in relazione alla loro capacità di scalare, però, risulta conveniente illustrare i due tipi di scalabilità di cui un generico sistema può beneficiare: scalabilità verticale e scalabilità orizzontale.

Un sistema composto da uno o più nodi interconnessi è scalabile verticalmente quando permette l'aggiunta di risorse ad un suo nodo con semplicità. In questa maniera il nodo può sopportare un carico di lavoro maggiore incrementando le performance generali del sistema. Per far scalare verticalmente un sistema è quindi sufficiente dotare un nodo di componenti più potenti, per esempio un processore o la memoria RAM.

Viceversa, un sistema è scalabile orizzontalmente quando è possibile aggiungergli con semplicità nuovi nodi, sui quali sarà ripartizionato il carico di lavoro. Aumentando il numero di nodi interconnessi e bilanciando le richieste tra di essi, infatti, il sistema aumenta la sua capacità di far fronte ad un numero maggiore di richieste. Le macchine che saranno aggiunte come nuovi nodi della rete possono essere uguali a quelle preesistenti oppure possedere componenti diversi, in base alle caratteristiche del sistema.

I due modelli di scalabilità illustrati presentano dei tradeoff. La scalabilità verticale è semplice da ottenere e non richiede la configurazione specifica di nuovi nodi, ma incontra dei limiti fisici e può risultare tutt'altro che economica. Non sempre, infatti, è possibile migliorare i componenti di un nodo e molto spesso farlo richiede l'acquisto di unità molto costose. I componenti sostituiti, inoltre, diventano inutili e risulta necessario cercare di contenere le spese provando ad utilizzarli in altri contesti.

La scalabilità orizzontale, viceversa, richiede uno sforzo iniziale di configurazione, poiché bisogna collegare la nuova macchina al sistema in attività per permetterle di rispondere a nuove richieste (si pensi all'aggiunta di nodi per Hadoop). Risulta però meno dispendiosa perché consente il riutilizzo di molte macchine economiche, non necessariamente di alta fascia. Sistemi scalabili orizzontalmente sono infatti spesso costituiti da commodity hardware, macchine di modeste performance, recuperate da precedenti impieghi. La scalabilità orizzontale è un requisito fondamentale per la realizzazione di supercomputer, cluster di migliaia di nodi, con capacità di elaborazione pari a decine di petaFLOPS.

I database relazionali scalano verticalmente con grande semplicità, mentre per ottenere scalabilità orizzontalmente è richiesto un significativo sforzo

ingegneristico aggiuntivo. I database relazionali, infatti, non sono progettati per effettuare query in modo distribuito: coordinare l'esecuzione di interrogazioni tra più macchine e garantire consistenza è perciò un onere dello sviluppatore.

I database NoSQL invece possono scalare orizzontalmente con grande facilità, perché quando sono stati sviluppati a questa capacità è stata data molta priorità, in previsione della grande mole di informazioni che essi avrebbero dovuto memorizzare. L'architettura risultante consente di beneficiare automaticamente di nuove macchine (commodity server, istanze cloud, ecc.) suddividendo i dati tra i nodi e coordinando l'esecuzione di interrogazioni tra di essi, in modo trasparente all'utente. La scalabilità orizzontale permette anche una maggiore replicazione dei dati, utile come protezione da malfunzionamenti e guasti.

3.2.3 Limiti dei database NoSQL

La flessibilità e la scalabilità dei database NoSQL è raggiunta accettando dei compromessi di cui quelli relazionali ne sono scevri.

Una prima fondamentale differenza riguarda le garanzie che la base di dati può dare durante l'esecuzione di transazioni, sequenze di operazioni successive che possono concludersi correttamente (se e solo se tutto hanno buon esito) oppure no. In caso di successo (segnalato con l'istruzione di commit), le modifiche fatte allo stato del database durante la transazione devono diventare permanenti o persistenti; in caso contrario, invece, il sistema deve essere in grado di ripristinare lo stato in cui era prima dell'avvio della transazione (esecuzione del rollback). Un database dovrebbe essere in grado di garantire l'esecuzione di transazioni dotate di un certo insieme di proprietà, comunemente noto come ACID (Atomicity, Consistency, Isolation, Durability). In modo particolare le transazioni dovrebbero essere consistenti, ovvero garantire che i vincoli del database non vengano violati durante l'esecuzione delle operazioni e riuscire a vedere correttamente i risultati delle precedenti operazioni conclusesi correttamente. I database tradizionali possono essere configurati per offrire forte consistenza, mentre quelli NoSQL generalmente offrono solo eventuale consistenza. Ogni prodotto, in modo particolare, offre delle proprie parziali garanzie, da studiare con attenzione in fase di scelta del database NoSQL. Ci sono, infatti, contesti in cui il completamento simultaneo di transazioni su tutti i nodi del sistema è fondamentale ed altri in cui è assolutamente irrilevante, come nel caso della applicazioni riguardanti i social media, dove le attività degli utenti non sono quasi mai collegate e non impongono particolari vincoli.

Le tecnologie distribuite come i database NoSQL, inoltre, possono soddisfare per loro natura al massimo due delle seguenti proprietà contemporaneamente in virtù del teorema di Brewer (teorema CAP):

coerenza (tutti i nodi vedono gli stessi dati nello stesso momento);

disponibilità (la garanzia che ogni richiesta riceva una risposta su ciò che sia riuscito o fallito);

tolleranza di partizione (il sistema continua a funzionare nonostante arbitrarie perdite di messaggi).

Come nel caso delle garanzie ACID, ogni prodotto ha una propria politica di gestione dei dati e decide di fare a meno di una delle tre proprietà elencate. In generale, però, queste tecnologie offrono le garanzie sintetizzate nell'acronimo BASE (Basically Available, Soft State, Eventual consistency), rilassando i vincoli sulla consistenza. Anche questo è un fattore da considerare in fase di scelta del database NoSQL.

Per i database relazionali stand-alone, invece, non vale il teorema di Brewer. Questo permette loro di garantire contemporaneamente tutte le tre proprietà e fornisce loro un altro punto di forza.

Per superare i limiti illustrati una nuova classe di database management system (DBMS) è in corso di sviluppo. Si tratta dei database NewSQL, che hanno come obiettivo quello di preservare la capacità di scalare dei database NoSQL mantenendo le garanzie ACID dei database relazionali. Questi nuovi sistemi solitamente supportano il modello di dati relazionali ed il linguaggio SQL per le interrogazioni, ma possono essere realizzati in modi molti diversi. Un esempio di database NewSQL è Spanner, progettato da Google e ideato per sopperire alla mancanza di transazioni in BigTable, il suo predecessore. Google F1 è il DBMS sviluppato su Spanner ed è impiegato internamente nell'infrastruttura della Google Cloud Platform (vedasi capitolo X). Le architetture dei database NewSQL sono molto variegate, infatti è possibile trovare semplici motori di database ottimizzati per scalare o cluster di nodi che non condividono alcun dato, con propri meccanismi di interrogazione e flussi di controllo.

3.3 Tassonomia

Come detto nel primo paragrafo del capitolo, i database NoSQL non condividono lo stesso modello di dati, ma è possibile individuare quattro macrocategorie per provare a classificarli: database con coppie chiave-valore, orientati ai documenti, a colonna e a grafo. Ciascuna categoria verrà analizzata da due prospettive: il modello di memorizzazione impiegato (strutture dati logiche o fisiche) ed il modello di interrogazione offerto all'utilizzatore (come

e a che prezzo i dati sono recuperati). Questo tipo di analisi va condotta attentamente ogni volta che si vuole utilizzare una soluzione di tipo NoSQL nel core di un sistema.

3.3.1 Database con coppie chiave-valore

I database con coppie chiave/valore funzionano similmente ad una grande tabella hash, in cui ad una chiave viene assegnato un valore. Poiché questi valori abbinati alle chiavi possono essere di diversi tipi (stringhe, JSON, BLOB, ecc.) il database risulta molto flessibile.

Le chiavi possono essere specificate esplicitamente dall'utente oppure generate automaticamente dal sistema. Ad ognuna di esse corrisponde un puntatore ad un particolare oggetto e spesso vengono raccolte in bucket, gruppi logici di chiavi, che però non hanno conseguenze sull'ordinamento fisico dei dati.

Le performance di questi sistemi, in virtù della loro semplice architettura, sono facilmente incrementate attraverso l'utilizzo di meccanismi di cache del mapping tra le chiavi ed i rispettivi valori. Con riferimento al teorema di Brewer, spesso questi database mancano di consistenza ed il loro modello estremamente semplice non offre molte delle funzionalità tipiche dei database tradizionali (atomicità della transazioni o consistenza quando transazioni multiple vengono eseguite contemporaneamente) che vanno invece implementate nell'applicazione.

Con la crescita costante del volume dei dati bisogna inoltre sviluppare un meccanismo per poter generare sempre nuove chiavi in modo coerente. Questo requisito non va sottovalutato poiché gli oggetti sono interrogabili unicamente attraverso le loro chiavi: i valori ad esse abbinati sono oscuri al sistema.

Alcuni dei più importanti database di questo tipo sono Riak, DynamoDB e Redis.

3.3.2 Database orientati ai documenti

I database orientati ai documenti raggruppano le informazioni semanticamente collegate tra loro in documenti, oggetti simili a quelli cui si è abituati nel paradigma di programmazione object-oriented. Per certi versi è possibile immaginare i documenti come una raccolta di coppie chiave-valore dei database precedentemente illustrati. I documenti, spesso codificati in file XML, JSON o BSON (codifica binaria di un oggetto JSON), contengono uno o più campi di un certo tipo, come stringhe, date, array, sotto-documenti. I record

non sono frammentati in tante colonne, ma conservati insieme. Ogni documento può contenere, però, campi diversi, mantenendo la proprietà di schemi dinamici, tipica dei database NoSQL; questa flessibilità è utile per modellare dati non-strutturati e polimorfici o per poter far evolvere agevolmente un'applicazione.

Una differenza tra i database con coppie chiave-valore e quelli orientati ai documenti è che quest'ultimi spesso aggiungono dei metadati ai valori inseriti nei campi per permettere query sul contenuto degli attributi stessi. In generale queste tecnologie offrono la possibilità di eseguire query su un qualunque campo di un documento e di aggiornarne direttamente i valori, ma non di effettuare join, che dovranno essere gestiti dall'applicazione. Alcuni prodotti, infine, offrono funzionalità di indicizzazione per ottimizzare le query.

I prodotti più consolidati per questa tipologia di base di dati sono MongoDB e CouchDB.

3.3.3 Database a colonna

I database a colonna sono stati sviluppati cercando di emulare l'architettura di BigTable[] e perciò organizzano le informazioni in un modo simile a quello adoperato nella tecnologia di Google. Il risultato è che il modello di dati assomiglia ad un dizionario multidimensionale, sparso, distribuito e persistente.

Ad una entità del database, individuabile dalla sua chiave, sono abbinati dei valori memorizzati in colonne specifiche, identificate da un proprio qualificatore. Le entità non devono necessariamente possedere un valore per ogni qualificatore e possono liberamente aggiungere nuove colonne. Anche questa tipologia di database risulta molto flessibile, perché i dati non devono aderire necessariamente ad uno stesso modello prefissato e possono modificarlo a runtime.

I database a colonna memorizzano in celle adiacenti i valori appartenenti alla stessa colonna, diversamente da quelli relazionali che raggruppano i dati per righe. Più colonne possono essere poi raccolte in una column family (da qui la multidimensionalità del dizionario). Generalmente le colonne comprese in una stessa colomun family sono fisicamente memorizzate in celle vicine di memoria, per ottimizzare la scansione dei valori conservati nella base di dati.

La lettura e la scrittura dei dati vengono eseguite iterando su una colonna di una determinata column family per un insieme di chiavi, piuttosto che su una riga; per questo motivo i database risultano molto efficienti quando viene richiesto loro di recuperare i valori di uno stesso qualificatore, indipendentemente dal numero di righe nel database, perché questi valori sono adiacenti.

I database di questo tipo più impiegati sono Apache HBase e Apache Cassandra.

3.3.4 Database a grafo

I database a grafo sono meno diffusi, ma particolarmente utili quando le informazioni da memorizzare sono collegate tra loro in virtù di alcune caratteristiche. Se i dati, infatti, presentano una struttura a grafo, con nodi ed archi, questa tipologia di database risulta adatta a contenerli.

L'utilizzo di una rete di relazioni, sebbene possa sembrare controintuitiva, può essere utile in molte applicazioni. Con un po' di ragionamento, infatti, è possibile impiegare questi database per applicazioni diverse dai social network, cui si è soliti pensare a primo acchito.

Sia i nodi sia le relazioni possono essere arricchite con delle proprietà e gli archi possono essere orientati o bidirezionali.

L'attrattiva di questa soluzione risiede nella semplicità con cui possono essere rappresentate le relazioni tra entità di una applicazione. Per questo motivo spesso sono impiegati in siti di e-commerce per costruire un grafo di utenti e prodotti acquistati. Tramite questo grafo è possibile riuscire a predire con dei recommender system[][][] quali nuovi prodotti un utente potrebbe essere interessato a comprare per fornire suggerimenti personalizzati.

I database a grafo possono essere interrogati per effettuare inferenze dirette o indirette sui dati memorizzati: verificare le relazioni tra i nodi, infatti, non richiede il completamento di costose operazioni di join. Le analisi sui tipi di relazioni sono molto efficienti, a differenza degli altri tipi, molto meno ottimizzati. Aggiungere o rimuovere relazioni dal grafo risulta banale.

Prodotti attualmente impiegati in molti casi d'uso sono Neo4j, Titan e InfiniteGraph.

Capitolo 4

Cloud computing

4.1 Caratteristiche e modelli di servizi offerti

Nel paragrafo 1.5 sono stati evidenziati i problemi che si devono solitamente affrontare durante la gestione di big data, ovvero quelli riguardanti le modalità e l'ubicazione dei processi di elaborazione e memorizzazione. Nel capitolo 2 è stato presentato Apache Hadoop, un framework progettato per elaborare grandi dataset in modo distribuito, mentre nel capitolo 3 sono stati approfonditi i database NoSQL, tecnologie scalabili in grado di memorizzare e recuperare efficientemente molte informazioni. Questi due strumenti forniscono modalità adatte per la gestione di grandi moli di dati, ma ancora nessuna soluzione è stata proposta per risolvere il problema dell'ubicazione di questi processi. Nel corso di questo capitolo, pertanto, verranno analizzate le piattaforme di cloud computing, degli strumenti per processare dati in remoto, che spesso si rivelano più adeguati dei server locali.

Questi servizi si sono sviluppati soprattutto negli ultimi dieci anni, in risposta alle esigenze di molte imprese di decentralizzare la fase di elaborazione dei dati. A fronte del notevole aumento del volume di dati raccolti, infatti, numerose aziende hanno realizzato di possedere infrastrutture inadatte o troppo costose (spese di manutenzione, upgrade, personale, energia elettrica, ecc.) per soddisfare le richieste dei propri sistemi. Diversi colossi informatici hanno ritenuto questa situazione una buona occasione per impiegare con maggiore produttività i loro data center e per aggiungere una nuova cospicua fonte di guadagno al loro business. Sono nate così le piattaforme di cloud computing, gruppi di servizi noleggiabili da aziende o privati secondo dettagliate politiche di prezzi ed utilizzabili nella forma di web service. Usufruendo delle potenti infrastrutture di aziende come Amazon, Google, IBM, Microsoft, ecc. le imprese ricevono la garanzia di elaborare e memorizzare le

informazioni con grande velocità ed in modo sicuro (per via dei processi di criptografia e replicazione dei dati).

Integrare questi servizi nelle applicazioni preesistenti risulta spesso semplice, per via delle API in diversi linguaggi di programmazione che vengono fornite agli sviluppatori. Beneficiando delle infrastrutture di queste imprese, i sistemi risultano scalabili e godono di disponibilità pressoché continua. Le interfacce web, inoltre, rendono la gestione di questi servizi ancora più intuitiva.

Le piattaforme di cloud computing differiscono per i servizi contenuti, le politiche dei prezzi ed altre caratteristiche minori, ma, generalmente, condividono i modelli di servizi offerti: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) e Software as a Service (SaaS).

Col modello IaaS il cloud provider offre macchine fisiche o, più spesso, virtuali con hardware configurabile oppure altre risorse, quali load balancer, indirizzi IP, VLAN, server per lo storage di file, ecc. Queste risorse remote possono essere utilizzate per avviare macchine con sistema operativo personalizzato - tramite immagini di dischi preconfigurate - con cui eseguire qualunque tipo di computazione (ad esempio, è possibile utilizzarle per un cluster di Hadoop). In questo modo gli utenti possono disporre di un numero di computer virtualmente illimitato con cui far fronte a qualunque tipo di esigenza. Collegandosi alle proprie macchine, spesso tramite il protocollo di rete SSH, è possibile infatti installare ed avviare su di esse dei propri programmi e raccogliere i risultati delle analisi effettuate nei data center del cloud provider.

Solitamente i costi di questi servizi sono proporzionali alla quantità di risorse utilizzate nel tempo e, pertanto, è consigliabile condurre analisi sull'effettiva economicità della soluzione. Per tale motivo spesso le macchine vengono avviate, impiegate per qualche elaborazione e subito spente, al fine di non pagare intervalli di inattività: questo approccio risulta praticabile per via dei ridotti tempi di accensione e spegnimento delle istanze remote.

Nel modello di PaaS i fornitori offrono delle risorse di elaborazione già configurate e pronte all'uso. Tipicamente vengono messe a disposizione macchine con un determinato sistema operativo e diversi software preinstallati: ambienti d'esecuzione per linguaggi di programmazione (ad esempio la Java Virtual Machine), database, web server e strumenti di varia natura. Queste macchine possono essere utilizzate nel core di un proprio sistema e, se adeguatamente selezionate, si rivelano molto adatte a sopportare grandi carichi di lavoro. Un servizio PaaS, ad esempio, potrebbe gestire le richieste fatte ad un sito web, ad una applicazione per smartphone, ad un servizio di telefonia, ecc.

Diversi provider, come Google e Microsoft offrono anche delle funzionalità di load balancing: all'aumentare delle richieste il provider alloca dinamicamente nuove risorse per non far degradare il sistema nelle performance, senza che l'utilizzatore del servizio debba intervenire manualmente. I sistemi sono quindi capaci di scalare automaticamente e, pertanto, questi servizi di adattano bene ad ambienti real-time.

Infine, col modello SaaS, i provider noleggiano i propri software, nascondendo agli utilizzatori i dettagli sulle infrastrutture sottostanti e sollevandoli dall'onere di configurare ed avviare delle macchine remote. Il modello SaaS viene talvolta definito come "software on-demand", proprio perché gli utenti possono fare uso delle applicazioni offerte nella piattaforma in relazione alle loro esigenze. È compito del cloud provider assicurarsi che i prodotti siano sempre disponibili e funzionanti, bilanciando i carichi di lavoro e manutenendo ed evolvendo il software (i cambiamenti sono trasparenti agli utenti).

Il modello SaaS permette agli investitori di utilizzare soluzioni già in commercio e risparmiare denaro evitando l'acquisto di hardware ed i costi di manutenzione: in questo modo è possibile orientarsi verso altri obiettivi.

Tutti i modelli, tuttavia, presentano lo svantaggio di dover memorizzare i dati degli utenti nei server del cloud provider, esponendoli al rischio di accessi non autorizzati. Per ovviare a questi problemi i fornitori dei servizi solitamente offrono diverse garanzie (i file vengono divisi in più parti, criptati e sparsi in data center diversi), ma gli utenti possono anche usufruire di servizi di criptazione di terze parti per assicurarsi maggiore protezione.

La piattaforma per l'analisi di big data sviluppata per questo studio necessitava di servizi di cloud computing per potere adoperare framework come Hadoop e per beneficiare della scalabilità garantita dal cloud provider. Per tale motivo essa presenta una architettura ibrida, con diverse componenti cloud. È stato comunque dimostrato[http://www.accenture.com/us-en/Pages/insight-cloud-based-hadoop-future-big-data.aspx] che le soluzioni cloud costituiscono, dal punto di vista economico, una valida alternativa a quelle bare-metal per molti casi d'uso e sono pertanto suggerite anche in assenza di necessità simili a quelle di questo caso di studio.

La Google Cloud Platform è stata preferita alle altre piattaforme per la varietà di tecnologie nel suo portfolio, per la superiorità nelle prestazioni rispetto ai concorrenti[http://www.infoworld.com/d/cloud-computing/ultimate-cloud-speed-tests-amazon-vs-google-vs-windows-azure-237169?page=0,4] [http://yourstory.com/2013/compute-engine-better-than-aws/], per la sua semplicità di utilizzo, per la molteplicità di strumenti offerti agli sviluppatori (plugin per IDE, script di configurazione e deploy, ecc.) e per la sua effettiva economicità (per pro-

vare i servizi ed eseguire tutti gli esperimenti della piattaforma sono stati consumati in totale solo X.X$).

4.2 Google Cloud Platform

La Google Cloud Platform (GCP) è una raccolta di alcune delle tecnologie sviluppate da Google ed attualmente utilizzate internamente dall'azienda per garantire il corretto funzionamento dei suoi prodotti, quali Gmail, YouTube, il motore di ricerca ed altri.

La piattaforma è stata resa disponibile al pubblico a partire dal 2008 con l'obiettivo di noleggiare ad aziende e privati gli stessi strumenti adoperati da Google nella forma di web service. Con essa Google è entrata nel panorama dei cloud provider, in diretta competizione con Amazon Web Services (AWS), Azure di Microsoft, SoftLayer e BlueMix di IBM, ecc.

Nel corso degli anni la piattaforma di Google è stata evoluta, migliorando i suoi prodotti, introducendo nuovi servizi, fornendo ulteriori API agli sviluppatori ed aggiornando le politiche dei prezzi, conformemente ai competitori.

Secondo le modalità tipiche dei servizi di cloud computing, gli interessati alle tecnologie offerte nel portfolio possono utilizzare i prodotti di cui hanno bisogno dietro un opportuno compenso. Noleggiando le proprie infrastrutture ed i propri software, pertanto, Google si assicura una cospicua fonte di guadagno (come anticipato nel paragrafo 1.4) e permette agli sviluppatori di adoperare potenti strumenti, adatti ad elaborare efficientemente big data.

I servizi inclusi nella piattaforma sono di natura eterogenea e rientrano nei modelli di IaaS, PaaS e SaaS. Tutti beneficiano di diverse qualità come scalabilità e disponibilità, assicurano protezione e replicazione dei dati trattati e garantiscono elevate performance, indipendentemente dal numero di richieste cui devono rispondere. Le tecnologie offerte dalla Google Cloud Platform sono, in generale, orientate all'elaborazione o alla memorizzazione delle informazioni; talvolta costituiscono servizi specifici per le applicazioni. Di seguito viene presentato un elenco di alcuni dei prodotti principali inclusi nella piattaforma:

Compute Engine: IaaS che permette la creazione di una o più macchine virtuali con hardware configurabile dall'utente. Può essere utilizzata per creare cluster di computer, che saranno collegati tra loro attraverso la fibra ottica privata di Google;

App Engine: PaaS utilizzata per sviluppare ed ospitare applicazioni nei data center di Google. L'SDK supporta diversi linguaggi di programmazione ed è compatibile con i maggiori framework. Google si occupa di gestire le ri-

chieste fatte alle applicazioni utilizzanti App Engine, amministrare i database e bilanciare i carichi di lavoro;

Cloud SQL: database relazionale MySQL che gestisce, in modo trasparente all'utente, la replicazione ed il backup dei dati, l'aggiornamento del sistema ed il recupero automatico in caso di errori. È capace di processare efficientemente miliardi di tuple e può essere usato dai tradizionali strumenti di gestione di database relazionali;

Cloud Storage: servizio per la memorizzazione di dati, simile ai tradizionali servizi di file hosting;

Cloud Datastore: database NoSQL schema-less, ideale per memorizzare dati non strutturati. Supporta transazioni ACID e query SQL-like. Google provvede allo sharding ed alla replicazione dei dati;

Big Query: IaaS che permette di analizzare grandi dataset in pochi secondi eseguendo query SQL-like, ideale come strumento per acquisire in tempo reale comprensione su big data.

Tra i prodotti presentati alcuni sono stati utilizzati per la piattaforma oggetto di questo lavoro e saranno, pertanto, presentati nel dettaglio.

4.2.1 Google Compute Engine

Google Compute Engine (GCE) è uno dei servizi principali della GCP che permette agli utenti di avviare e controllare remotamente macchine virtuali (VM) in esecuzione nei data center di Google.

Gli utenti hanno la possibilità di configurare le macchine che saranno avviate e, una volta attive, utilizzare queste istanze per effettuare qualsiasi tipo di operazione. Nello specifico, essi devono specificare il sistema operativo ed il tipo di macchina. GCE è compatibile con molti sistemi operativi e distribuzioni, alcuni gratuiti, altri a pagamento. Alcuni tra quelli nativamente supportati sono Debian, CentOS, openSUSE, Red Hat, FreeBSD e Windows Server, ma viene anche offerta la possibilità di caricare immagini personalizzate, che meglio rispondono alle esigenze degli utilizzatori. Il tipo di macchina, invece, determina le specifiche fisiche della VM, come il quantitativo di memoria disponibile o il numero di core virtuali. Il tipo deve essere scelto tra quelli offerti da Google, che, per comodità, raggruppa l'offerta in quattro categorie di macchine: standard, high CPU, high memory, small. Gli utenti possono pertanto scegliere, in relazione all'utilizzo che sarà fatto delle istanze, se avviare macchine con molta memoria o preferire una maggiore potenza di elaborazione oppure propendere per soluzioni economiche e con poche risorse. La tabella X.X riassume l'attuale proposta di Google e descrive i tipi di VM.

Quando vengono spente dall'utente, le istanze di GCE sono distrutte e le risorse a loro dedicate vengono riutilizzate per nuove VM; con questo processo tutti i dati memorizzati sul disco dall'utente vengono perduti. Per risolvere questo problema, Google offre la possibilità di utilizzare dischi persistenti, risorse assimilabili agli hard drive comunemente collegati ai personal computer. Essi si presentano come la soluzione primaria per la memorizzazione di dati in modo persistente. Un disco può essere collegato e scollegato ad una VM secondo le preferenze dell'utente e può essere programmato per distruggersi o rimanere in funzione quando l'istanza cui è collegato viene spenta. Nel secondo caso i dati memorizzati al suo interno non vengono persi e l'utente può collegare il disco ad una nuova istanza. In fase di configurazione di un disco l'utente può scegliere se creare un disco standard oppure un SSD: come per i tipi di macchina, anche i dischi hanno performance e prezzi differenti.

Le macchine virtuali possono essere utilizzate indipendentemente oppure possono essere collegate tra loro per formare un cluster. Google mette a disposizione, ad esempio, degli script per effettuare il deploy di un cluster Hadoop avente caratteristiche impostate dall'utente (numero e tipologia di nodi, file da copiare sulle macchine all'avvio, file system, ecc.). Le macchine sono collegate tra loro attraverso la fibra ottica di Google e, pertanto, le connessioni registrano basse latenze: il trasferimento di dati tra i nodi del cluster risulta molto veloce e GCE rappresenta una buona infrastruttura per il calcolo distribuito.

Tutte le istanze sono collegate ad Internet e posso essere collegate in una rete interna, eventualmente protetta con un firewall. Ciascuna di esse, inoltre, possiede un proprio indirizzo IP e può essere identificata tramite una risoluzione DNS.

Le risorse di GCE possono essere facilmente gestite attraverso lo strumento da riga di comando o tramite la web console degli sviluppatori della piattaforma (in entrambi i casi il protocollo SSH viene utilizzato per collegarsi alle istanze remote). I software possono poi interagire con le macchine utilizzando le RESTful API per diversi linguaggi offerte agli sviluppatori (Java, Python, Ruby, .NET, Go, PHP, Objective C, ecc.). Esse solitamente utilizzano OAuth 2.0 per completare l'autenticazione e per integrare GCE con altri prodotti della piattaforma come Cloud Storage.

Google non garantisce, ad ogni modo, che le istanze siano in esecuzione per il 100% del tempo, pertanto è compito degli utenti assicurarsi che il proprio servizio possa ripristinare il proprio stato a seguito di un guasto imprevisto; diversi accorgimenti vengono comunque suggeriti per progettare sistemi robusti.

4.2.2 Google Cloud Storage

Google Cloud Storage (GCS) rappresenta una delle tre soluzioni offerte nella piattaforma per la memorizzazione dei dati, insieme a Cloud SQL e Cloud Datastore. Mentre questi ultimi due prodotti costituiscono dei veri e propri database, GCS è un più semplice servizio di file storage nel cloud. Con esso gli utenti possono arrivare a memorizzare diversi terabyte di dati, pagando in relazione all'ammontare di dati trasferiti ed allo spazio fisico occupato (il prezzo è di circa 0.026$/GB/mese). I dati possono essere memorizzati in uno o più data center di Google (Stati Uniti, Unione Europea ed Asia), che garantisce agli utilizzatori elevate velocità di lettura, continua disponibilità e, soprattutto, scalabilità. Il servizio risulta, pertanto, anche adatto a distribuire in download diretto file molto grandi o molto richiesti.

GCS è una tecnologia di storage rivolta agli sviluppatori, poiché i file memorizzati possono essere utilizzati da altri prodotti della piattaforma di Google. Per questo motivo viene spesso utilizzato insieme ad altri servizi, ad esempio Compute Engine, App Engine e BigQuery. Gli utenti che non sono interessati a sviluppare software possono, invece, utilizzare il più noto Google Drive, un più semplice servizio di hosting. Ad ogni modo, utilizzando il Google Drive SDK, è possibile integrare i due servizi, per offrire agli utenti finali i file caricati su Cloud Storage.

In GCS tutti i dati vengono contenuti in un progetto, che consiste di un insieme di utenti abilitati a gestire i file, un insieme di API attivate ed impostazioni di fatturazione ed autenticazione. Ogni utente può gestire uno o più progetti e può creare, all'interno di un progetto, un numero arbitrario di bucket. Essi rappresentano i contenitori dei file e sono assimilabili a delle cartelle: tutti gli oggetti caricati in GCS devono essere inseriti in un bucket. A differenze delle cartelle, però, i bucket non possono essere annidati e non possono essere condivisi tra più progetti. Ogni bucket è caratterizzato da un identificativo univoco e permette di configurare i privilegi da assegnare agli utenti che vogliono accedervi. Gli oggetti di GCS non possono essere condivisi tra più bucket e possiedono due componenti: dati e metadati. I dati rappresentano il file da memorizzare, mentre i metadati sono delle coppie chiave-valore che descrivono delle qualità dell'oggetto.

GCS offre una forte read-after-write consistency per tutte le operazioni di upload e cancellazione. Questo significa che un oggetto appena caricato può subito essere scaricato, rimosso o ispezionato nei suoi metadati. Allo stesso modo risulta impossibile accedere ad un oggetto che è stato appena cancellato. Dal punto di vista della disponibilità, le operazioni su GCS sono atomiche: gli oggetti caricati sono accessibili solo se perfettamente integri. I file corrotti o parziali non possono essere letti, così come quelli ancora in fase

di upload.

Come per Compute Engine, anche le risorse di Cloud Storage possono essere gestite in diversi modi: interfaccia da riga di comando, web console ed API (XML, JSON, ecc.). I software possono interagire con il servizio attraverso una delle librerie offerte da Google, disponibili per tanti linguaggi di programmazione. Anche GCS utilizza OAuth 2.0 per gestire i meccanismi di autenticazione e autorizzazione.

Capitolo 5

Piattaforma cloud per l'analisi di big data

5.1 Contesto del lavoro

Le tecnologie presentate nei precedenti capitoli sono state impiegate per lo sviluppo di una piattaforma cloud per l'analisi di big data provenienti da social network, il cui funzionamento sarà illustrato nel corso di questo capitolo.

L'origine di tale strumento risiede nell'esigenza avvertita da Vox, Osservatorio Italiano sui Diritti, di monitorare le zone italiane dove omofobia, razzismo, discriminazione verso i disabili, misoginia ed antisemitismo sono maggiormente diffusi. L'ambizioso obiettivo del progetto, infatti, è lo sviluppo di una Mappa dell'Intolleranza, un mezzo capace di evidenziare i luoghi affetti da questi gravi problemi di discriminazione, attraverso l'analisi di tweet in lingua italiana geolocalizzati.

L'idea di questo strumento non è nuova: gli studenti della Humboldt State University in California hanno sviluppato nel 2013 la Hate Map (in figura X), una mappa di calore interattiva in grado di mostrare le principali zone degli Stati Uniti da cui provenivano tweet razzisti, omofobi e contro le persone diversamente abili. Il loro lavoro ha consentito importanti osservazioni di carattere sociale: è stato riscontrato, ad esempio, che la maggior parte dei messaggi d'odio avevano origine in piccole città o aree rurali piuttosto che in grandi metropoli.

La Mappa dell'Intolleranza di Vox è pensata per offrire alle amministrazioni locali italiane uno strumento per scoprire le aree più a rischio, al fine di agire concretamente sul territorio per risolvere i problemi di discriminazione. Recenti statistiche dimostrano quanto importante sia operare attivamente in

tale direzione: solo nel 2013, il 25% degli omosessuali è stato vittima di violenza, 6743000 donne hanno subito abusi fisici o sessuali ed il 45% dei giovani si è dichiarato xenofobo o diffidente degli stranieri.

Il ruolo dei social network in questo contesto è essenziale, poiché essi permettono di veicolare ed alimentare sentimenti di odio verso il prossimo con grande facilità. Uno studio sistematico dei messaggi discriminatori pubblicati dagli utenti potrebbe, però, permettere di prevenire l'insorgere di episodi violenti e situazioni spiacevoli.

Il progetto avviato da Vox risulta essere tanto importante, quanto complesso. Fa emergere, infatti, diverse problematiche di natura linguistica e tecnologica alle quali diverse facoltà italiane stanno facendo fronte collaborando, al fine di produrre uno strumento affidabile e preciso.

La piattaforma che sarà ora descritta permette un importante passo avanti nella realizzazione del progetto della Mappa dell'Intolleranza, poiché si prefigge di essere l'infrastruttura portante del sistema ed uno strumento scalabile per la raccolta e l'analisi di grandi moli di dati.

5.2 Architettura della piattaforma

Nello sviluppo della piattaforma è stato seguito un approccio modulare, in linea con le best practice dell'ingegneria del software. Questa scelta progettuale ha portato allo sviluppo di un sistema dotato di moduli indipendenti tra loro, facilmente manutenibili e concepiti per portare a termine determinati compiti in modo autonomo. Il software beneficia, nel complesso, di diverse importanti qualità, tra cui spiccano evolvibilità e scalabilità. La prima risulta essenziale per una piattaforma costituente l'infrastruttura di un sistema complesso, che, verosimilmente, cambierà nel corso del tempo per rispondere a nuove esigenze. Nel suo core, infatti, la piattaforma è predisposta ad accettare nuovi moduli, con cui potrà effettuare operazioni attualmente non supportate, anche molto differenti tra loro. Il requisito di scalabilità, d'altronde, è imprescindibile per un software big-data-oriented, nonostante la dimensione dei tweet finora raccolti permettesse l'utilizzo di tecnologie tradizionali per lo storage ed il processing. In fase di sviluppo, invece, si è supposto di dover elaborare più informazioni di quelle possedute e sono stati presi tutti gli accorgimenti del caso, impiegando tecnologie ed algoritmi adatti per i big data. Il sistema è stato pertanto contestualizzato in una situazione futura, simulando lunghi periodi di attività e ipotizzando di aver raccolto molti più dati di quelli attualmente a disposizione.

Combinando i moduli che costituiscono il sistema è possibile eseguire delle attività più complesse, dette task. Nella versione iniziale della piattaforma

l'attenzione è stata posta sui task orientati alla raccolta e all'elaborazione di informazioni, tralasciando i task di data visualization. Tra i task implementati, particolare attenzione è stata posta su quelli destinati a trattare in modo diretto grandi moli di dati, detti big-data-critic. Questi task hanno richiesto, infatti, l'utilizzo di tecnologie ed algoritmi scalabili e le loro performance sono state oggetto dello studio riportato nel paragrafo X.

L'architettura della piattaforma è illustrata in figura X: sono mostrati i moduli che costituiscono il sistema ed i task che è in grado di eseguire usufruendo di essi [[evidenziare con colore diverso i task bdc]].

Il software è stato sviluppato in Java, lo stesso linguaggio utilizzato dalle librerie che esso utilizza come dipendenze. Le tecnologie big-data-oriented impiegate, inoltre, espongono API in questo linguaggio, lasciando, di fatto, poco margine di scelta in fase realizzativa. A fronte del gran numero di software di terze parti e librerie necessari per completare i task della piattaforma, è stato utilizzato il framework Maven per gestire in modo agevole le dipendenze e rendere il programma facilmente evolvibile.

5.3 Metodologia ed obiettivi dell'analisi

La piattaforma realizzata, nonostante sia impiegabile in diversi modi, è stata utilizzata per un caso di studio specifico, che ha indirizzato lo sviluppo verso determinati moduli del sistema piuttosto che altri.

L'obiettivo del lavoro è stato quello di individuare delle caratteristiche linguistiche salienti tra i tweet potenzialmente discriminatori verso alcune categorie di persone, analizzando i dati raccolti dal punto di vista statistico. Ignorando la semantica dei messaggi si è ipotizzato di riuscire a trovare delle caratteristiche comuni tra i messaggi di odio indirizzati a determinate categorie di persone. Se tali caratteristiche emergessero, sarebbe possibile utilizzarle, ad esempio, per scoprire nuovi messaggi, ignorati nelle precedenti sessioni di campionamento.

Nella fase iniziale del lavoro è stato raccolto da Twitter un numero significativo di messaggi potenzialmente discriminatori, raggruppandoli in base alla categoria di individui vittime di forme di intolleranza. Le categorie più ricche di messaggi sono state in seguito indicizzate per ottenere una significativa distribuzione di probabilità dei termini utilizzati nei tweet che vi appartenevano. Tali distribuzioni saranno più informative all'aumentare dei messaggi raccolti, pertanto i risultati ottenuti potrebbero migliorare a seguito di sessioni più lunghe di raccolta di dati.

Per far emergere da questi indici le parole più informative si è reso necessario confrontare queste distribuzioni di probabilità con una di riferimento

che approssimasse quanto meglio possibile dei testi generici in lingua italiana. Infatti, i termini frequenti nei tweete e non d'uso comune nella lingua italiana appartengono all'insieme di parole più discriminanti per la categoria in esame e sono di interesse per lo studio. Un termine molto diffuso sia in tweet razzisti che in altri contesti, ad esempio, non può essere utile ad identificare un messaggio razzista; viceversa, un termine molto frequente in messaggi razzisti, ma generalmente impiegato di rado in testi in italiano potrebbe essere d'aiuto per individuare messaggi discriminatori.

Una distribuzione che approssimasse bene i testi in lingua italiana è stata ottenuta indicizzando itWaC, una ricca raccolta di testi in lingua italiana, ottenuta tramite un processo di crawling nel 2006. La collezione fa parte di una raccolta più ampia di corpora annotati (con part-of-speech tagging e lemmatizzazione) chiamata WaCky, che comprende anche risorse per altre lingue, come l'inglese (ukWaC) ed il tedesco (deWaC). Con oltre 4 milioni di documenti su contenuti diversi, filtrati ed annotati, itWaC costituisce uno dei più grandi corpus pubblici in lingua italiana e rappresenta una fondamentale e versatile risorsa linguistica, utilizzabile per molti scopi scientifici. In figura X sono riportate le caratteristiche del corpus, tratte dalla documentazione ufficiale[paper wacky].

I termini più significativi per i tweet discriminatori sono stati individuati calcolando la divergenza di Kullback-Leibler[] (KLD) tra la distribuzione di probabilità dei messaggi e quella di itWaC. La KLD è una fondamentale equazione della teoria dell'informazione che quantifica la prossimità tra due distribuzioni di probabilità. La sua formula ha origine dalla teoria della verosimiglianza e mira a quantificare quanta informazione si perde approssimando una distribuzione con un'altra. Effettuando un ranking dei valori calcolati con la KLD per ogni categoria di messaggi è stato possibile rilevare le parole più significative per ciascun contesto.

Per condurre lo studio appena illustrato è stato necessario implementare diversi moduli e combinarli per completare diversi task. Di seguito vengono analizzate nel dettaglio le fasi del lavoro ed i componenti della piattaforma.

5.4 Crawler

Uno dei moduli più importanti della piattaforma è il crawler. Esso permette di raccogliere dati da una generica sorgente informativa per effettuare delle elaborazioni in tempo reale o in un momento successivo.

Il sistema può eseguire diversi tipi di crawler. Uno specifico per Twitter è stato già implementato e costituisce un componente essenziale per il progetto della Mappa dell'Intolleranza. Esso fa utilizzo della Streaming API

di Twitter per ricevere i tweet provenienti dal flusso globale di messaggi che soddisfano dei criteri fissati dallo sviluppatore. Twitter offre diversi streaming endpoint, ciascuno adatto per particolari casi d'uso. L'endpoint scelto per questo crawler è quello pubblico, adatto quando si vogliono raccogliere dati pubblici su utenti o argomenti specifici, poiché è in grado di filtrare i messaggi di interesse per il client tra quelli presenti nel flusso di tweet. Gli altri endpoint sono quello specifico per gli utenti, che raccoglie tutti i dati - anche quelli privati - riguardanti un determinato utente (utilizzato, ad esempio, per i client mobili per Twitter di terze parti) e quello per i siti (la versione multi-utente del precedente).

Pe ricevere informazioni da Twitter, il client deve autenticarsi come sviluppatore ed effettuare una richiesta al social network tramite l'API, specificando i contenuti che è interessato a ricevere. Pervenuta la richiesta, Twitter apre una connessione HTTP permanente con il client e gli invia dati in tempo reale, nella forma di documenti JSON, attraverso questo canale. È compito del client leggere in modo progressivo le informazioni che riceve da questa sorgente informativa e restare in ascolto di nuovi dati. Questo meccanismo di interazione si contrappone a quello dalla REST API, con cui il client può chiedere una particolare informazione al social network tramite un'interrogazione che utilizza una connessione temporanea per veicolare il dato. La natura della Streaming API, pertanto, richiede una particolare attenzione in fase di sviluppo del client, che dovrà esser capace di consumare dati in tempo reale, senza causare bottleneck. Per convertire velocemente i documenti JSON ricevuti con l'API in oggetti Java è stato perciò impiegato Jackson, un JSON-processor capace di estrarre dai documenti degli attributi specificati dall'utente (corpo del messaggio, autore, id, provenienza, ecc.) ed ignorare tutti gli altri.

Il crawler per Twitter è stato progettato per raccogliere messaggi in una determinata lingua contenti specifiche keyword. Per realizzare la Mappa dell'Intolleranza si rende necessario raccogliere tweet potenzialmente omofobi, razzisti, discriminanti verso disabili, misogini ed antisemiti. Per comodità questi messaggi sono stati suddivisi in classi numerate progressivamente (1: omofobia, 2: razzismo, 3: disabilità, 4: misoginia, 5: antisemitismo). Per raccogliere questi tweet, un gruppo di psicologi della Università degli Studi di Roma La Sapienza ha fornito, per ogni classe, un insieme di termini, detti seed, utilizzati con accezione spesso discriminatoria in contesti quotidiani. Un lavoro successivo, inoltre, ha permesso di ampliare questo lessico tramite l'uso di sinonimi, variazioni morfologiche (nel genere e nel numero) e risorse esterne, quali BabelNet e Morph-it!. Il risultato è stato un aumento del numero di seed per ogni classe, mostrato in figura X. Tutti i seed raccolti sono stati utilizzati per raccogliere i tweet appartenenti a ciascuna classe.

Per un corretto funzionamento del crawler è richiesto un file di configurazione apposito. Al suo interno l'utente deve specificare alcuni parametri: le credenziali da sviluppatore di Twitter, la lingua dei messaggi che è interessato a raccogliere (per il caso di studio è stata scelta quella italiana), le classi di interesse (coi rispettivi seed) ed la durata delle sessione di crawling per ciascuna classe. Nel file di configurazione, inoltre, l'utente può specificare dei filtri da applicare ai tweet in arrivo e degli handler per gestire i tweet che hanno superato questi controlli.

I filtri sono stati implementati per dare la possibilità all'utente di bloccare determinati messaggi in base a delle loro caratteristiche (contenuto, provenienza, autore, ecc.) e dotare il modulo di un meccanismo di protezione dallo spam. Effettuando lunghe sessioni di crawling è emerso, infatti, che diversi spam bot impiegavano alcuni dei seed delle classi per promuovere siti web di carattere pornografico (cosa consentita dai termini di utilizzo di Twitter) o per campagne pubblicitarie di varia natura, producendo messaggi inutili per il caso di studio. Mediante i filtri sviluppati questi ed altri messaggi sono stati bloccati, al fine di non alterare le distribuzioni di probabilità dei termini delle classi ed i risultati stessi delle analisi. L'utente è libero di non impiegare nessun filtro, utilizzarne alcuni già implementati o scriverne di nuovi.

Gli handler, invece, sono i moduli preposti alla gestione dei tweet che hanno superato i controlli dei filtri che sono stati attivati. Anche in questo caso l'utente può aggiungere delle classi al sistema per eseguire nuove operazioni, oppure impiegare una già presente nella piattaforma. Gli handler possono eseguire azioni di qualsiasi natura: memorizzare le informazioni inerenti un tweet in un database (locale o in cloud), condurre analisi di vario genere, serializzare il contenuto dei messaggi in un file di testo, ecc. Quest'ultima operazione era quella necessaria al caso di studio e, pertanto, è stato sviluppato un apposito handler per serializzare i tweet. L'handler in questione, insieme ad altri componenti della piattaforma sviluppata, fa utilizzo di un chunks builder, un modulo utile per scrivere molti dati su più file numerati progressivamente e aventi stessa dimensione.

Lunghe sessioni di crawling sono state avviate in una fase iniziale su tutte le classi. In un momento successivo, però, il crawler è stato mantenuto in esecuzione solo su quelle che hanno dimostrato riguardare più messaggi su Twitter: omofobia, disabilità e misoginia. Per ottenere distribuzioni di probabilità sufficientemente significative, infatti, risultava necessario raccogliere quanti più tweet possibili per una determinata classe: si è preferito, così, concentrare il crawler su specifiche categorie ed ottenere campioni più grandi piuttosto che ottenere dati poco significativi su tutte le classi. Le analisi condotte, in ogni caso, potranno essere ripetute in futuro su tutte le classi, qualora il numero di tweet raccolto dovesse crescere.

Alcune semplici operazioni sono state effettuate al termine della raccolta di tweet per migliorare i campioni ottenuti. Per ciascuna classe sono stati rimossi i messaggi duplicati e buona parte di quelli in lingue straniere, che avevano erroneamente superato il filtro di Twitter. Tale controllo è stato completato combinando i risultati di due strumenti per il riconoscimento della lingua: language-detection e language detection with infinity-gram, aventi precisione superiore al 99% per la lingua italiana.

I risultati ottenuti al termine di questa fase di crawling sono illustrati in figura X.X.

class 1 – 22564 – 1738 KB

class 3 – 29793 – 2242 KB

class 4 – 249425 – 17935 KB

5.5 Parser del corpus itWaC

Come già illustrato nel paragrafo 5.3, per ottenere una distribuzione di probabilità di termini che rappresentasse la lingua italiana si è ritenuto opportuno indicizzare il corpus itWaC. Esso, tuttavia, si presenta in una forma non direttamente processabile, poiché è costituito da archivi compressi, contenenti ciascuno parte dei documenti del corpus, per giunta in formato XML. Tali parti, oltretutto, non sono costituite da semplice testo, ma contengono le annotazioni prodotte dalla fase di part-of-speech-tagging eseguita dai curatori dei corpora.

Per questo motivo la piattaforma è stata dotata di un parser capace di elaborare il corpus e produrre dei file di testo indicizzabili. Il modulo sviluppato, pertanto, processa i file XML contenuti negli archivi senza decomprimerli, ignorando i tag ed inoltrando il testo letto ad un chunks builder, che serializza il corpus in file di più piccole dimensioni.

La grande dimensione del corpus rende il task appena descritto di natura big-data-critic. Per risolvere questo problema, tuttavia, non è stato necessario ricorrere a tecnologie cloud, poiché è stato sufficiente elaborare i file XML secondo il noto standard StAX, che prevede una lettura progressiva dei dati. Con questo approccio i file XML del corpus vengono assimilati a flussi da cui leggere gradualmente i dati: in questa maniera risulta indifferente elaborare file di grandi o piccole dimensioni, poiché essi sono comunque processati una riga per volta.

Il parser sviluppato converte il corpus producendo file aventi dimensione media di 500 MB, per un totale di oltre 11 GB di testo, senza tag di alcun tipo e direttamente indicizzabile.

5.6 Indicizzazione con MapReduce

Per indicizzare il corpus itWaC ed i tweet appartenenti alle tre classi selezionate è stato scritto un unico job di MapReduce da eseguire su un cluster di Hadoop. Le dimensioni dei dataset di input, destinate ad aumentare col passare del tempo, hanno reso il task big-data-critic, suggerendo l'utilizzo di soluzioni cloud. Il cluster di Hadoop (versione 1.2.1) è stato avviato, pertanto, su macchine remote noleggiate attraverso Google Compute Engine. Tali macchine sono state configurate per utilizzare come file system condiviso non il tradizionale Hadoop Distributed File System, ma un bucket di Google Cloud Storage. Questo risultato è stato ottenuto per mezzo del Google Cloud Storage Connector for Hadoop, uno strumento sviluppato da Google ed utilizzabile gratuitamente. Il connettore offre una serie di vantaggi, tra i quali la possibilità di gestire facilmente i file consumati da Hadoop (anche quando il cluster è spento), migliori performance durante l'esecuzione di job MapReduce per via dell'infrastruttura di GCS, minori tempi di avvio del cluster, assenza delle routine di gestione dell'HDFS, ecc. I dataset di input per Hadoop, per questo motivo, sono stati caricati su GCS in un bucket accessibili da tutti i nodi in un momento antecedente all'esecuzione dei job MapReduce.

Il job implementato utilizza un Mapper, un Combiner ed un Reducer per contare le occorrenze di una parola nel dataset di input, eseguendo il minor numero di operazioni possibili (piccoli miglioramenti nel codice producono una sensibile diminuzione dei tempi di esecuzione per via dell'elevato numero di chiamate alle funzioni da parte dei nodi del cluster). Le funzioni scritte condividono i principi di funzionamento alla base dei job esemplificativi illustrati nei paragrafi 2.3 e 2.4.1, con particolari accortezze aggiuntive per la natura dei grandi dati di input (ad esempio l'utilizzo del tipo long invece che int per tutte le variabili preposte a contare le occorrenze di un termine o il numero di righe di un file). Dominio e codominio delle funzioni sono riportati di seguito:

map (long, String) -> list(String, long)
combine (String, list(long)) -> (String, long)
reduce (String, list(long)) -> (String, long)

La funzione map riceve in input una coppia costituita dal numero di riga del file di input e dal testo presente su quella riga, che può essere un tweet oppure parte di un documento di itWaC. Successivamente invoca un tokenizer opportuno sul testo per dividerlo nelle sue parole costituenti e ritorna tante coppie (token, 1). Due tipi di tokenizer sono attualmente presenti nella piattaforma: uno specializzato nell'elaborazione di tweet e l'altro più appropriato per semplice testo. Il primo è stato sviluppato dal gruppo di ricerca

Noah's ARK della Carnegie Mellon University (CMU) ed è perfezionato nel riconoscimento di emoticons, hashtag, menzioni, url e retweet. Lo strumento è in grado di fare anche part-of-speech tagging, ma per il caso d'uso è stata utilizzata solo la libreria che effettua la suddivisione in token. Il secondo tokenizer della piattaforma, invece, proviene da Apache Lucene, libreria per l'information retrieval ed è il UAX29URLEmailAnalyzer. Tale tokenizer è stato preferito agli altri più noti poiché produce dei risultati più consistenti con lo strumento della CMU (non frammenta url e indirizzi email, medesima gestione della punteggiatura, ecc.). Questa peculiarità è risultata fondamentale dal momento che le frequenze delle parole nei dataset di input sarebbero dovute essere confrontate con la divergenza di Kullback-Leibler: produrre, con strumenti diversi, token uguali a partire da testi uguali rappresentava un presupposto imprescindibile per l'analisi.

La funzione combine, applicata localmente su ogni nodo, riceve in input un token ed una lista di 1, avente dimensione pari al numero di occorrenze del token in quella riga. Pertanto essa si limita a restituire una coppia (token, dimensione_della_lista).

La funzione reduce, infine, riceve in input un token ed una lista delle sue occorrenze nelle righe del dataset. Di conseguenza non deve che ritornare una coppia costituita dal token stesso e dalla somma dei numeri contenuti nella lista di input, ovvero il numero totale di occorrenze nel testo in input.

Il task di indicizzazione appena presentato è stato ripetuto più volte, facendo variare il numero di nodi nel cluster e la tipologia di istanze noleggiate; i risultati sono presentati in dettaglio nel paragrafo 5.X.

5.7 Divergenza di Kullback-Leibler con MapReduce

Per individuare i termini più significativi per ciascuna classe (omofobia, disabilità, misoginia) è stata misurata la divergenza di Kullback-Leibler tra la distribuzione di probabilità dei termini appartenenti a ciascuna classe e quella del corpus itWaC.

La divergenza di Kullback-Leibler DKL tra due distribuzioni discrete P e Q è data dalla formula

formula immagine

dove i rappresenta il termine i-esimo appartenente alle distribuzioni, P(i) la sua probabilità nella prima distribuzione e Q(i) quella nella seconda distribuzione.

L'obiettivo dell'analisi, tuttavia, non era quello di quantificare la somiglianza tra due distribuzioni di probabilità, quanto quello di individuare i termini aventi probabilità molto diverse, ovvero gli addendi della formula con valore maggiore. Per tale motivo, invece che sommare i termini è stato effettuato un ranking, che ha permesso di evidenziare quelli più significativi.

La KLD è stata calcolata tre volte, confrontando singolarmente la distribuzione di probabilità di ciascuna classe (P) con quella di itWaC (Q).

Durante il calcolo della probabilità di un termine in un documento (corpus o classi dei tweet) è stato applicato il Laplace smoothing[], poiché alcune parole comparivano nel copus di itWaC ma non nei tweet e viceversa (soprattutto nel caso di URL). Pertanto, la formula applicata per il calcolo della probabilità R di un termine i in un documento D è stata:

R(i) = (fD,i + 1) / (fD + |D|)

dove fD,i rappresenta il numero di occorrenze del termine i nel documento D, fD il numero totale di occorrenze di parole in D e |D| il numero di parole diverse nel documento.

Il calcolo della KLD è stato eseguito con un job di MapReduce, utilizzante un Mapper ed un Reducer. Come per il task di indicizzazione, il cluster di Hadoop è stato avviato su macchine di Compute Engine utilizzanti Cloud Storage come file system. Prima di procedere all'esecuzione del task, pertanto, si è reso necessario caricare online gli indici dei documenti prodotti nella fase precedente.

Dominio e codominio delle funzioni sono riportati di seguito:

map (long, String) -> list(String, double)

reduce (String, list(double)) -> (String, double)

La funzione map riceve in input una coppia costituita dall'indice del file che sta leggendo e dalla sua annessa riga contenente il termine i-esimo ed il suo numero di occorrenze (risultato dell'anteriore task di indicizzazione). Calcola, successivamente, la probabilità R del termine nel documento utilizzando la formula illustrata in precedenza (anteponendo al termine un determinato prefisso, la funzione map è in grado di riconoscere su quale dei documenti - P o Q - calcolare la probabilità). Per motivi di efficienza, le dimensioni dei vocabolari dei due documenti in esame ed il numero totale di occorrenze di parole in essi vengono calcolati dal nodo master del cluster prima di avviare il job. Essi, infatti, rimangono costanti durante l'elaborazione e possono risultare computazionalmente esosi da misurare. Le funzioni map, pertanto, possono direttamente leggere questi valori senza bisogno di doverli determinare ogni volta. I Mapper completano la loro esecuzione ritornando una coppia costituita dal termine e dalla probabilità calcolata. La funzione map, quando riconosce di stare calcolando la probabilità di un termine nel documento Q, moltiplica il valore ottenuto per -1 prima di ritornarlo. In

questo modo essa produce coppie del tipo (termine, P(termine)) e (termine, -Q(termine)). Questo accorgimento è stato ideato per permettere alla funzione reduce di riconoscere, successivamente, se una data probabilità proveniva da P o Q (la KLD, infatti, non è simmetrica).

La funzione reduce riceve in input un termine ed una lista di probabilità. Nel caso specifico la lista contiene la probabilità del termine in P e/o la probabilità del termine in Q (se un termine compare solo in una distribuzione la lista conterrà un solo elemento). Controllando se la probabilità è positiva o negativa la funzione reduce è in grado di determinare la distribuzione d'origine ed applicare correttamente la formula

DKLi = P(i) * ln (P(i) / Q(i))

Se una delle due probabilità è mancante allora viene calcolata dalla funzione reduce seguendo il criterio del Laplace smoothing. La probabilità R per il termine i mancante nel documento D è data dalla formula

R(i) = 1 / (fD + |D|)

dove fD rappresenta il numero totale di occorrenze di parole in D e |D| il numero di parole diverse nel documento (entrambi già calcolati dal nodo master e accessibili direttamente dal Reducer).

La funzione reduce, quindi, ritorna un insieme di coppie (termine, DKL) non ordinate, che Hadoop serializza su file. Ordinando i valori calcolati in modo decrescente è possibile osservare i termini più significativi per la classe in esame.

Nel paragrafo 5.X sono presentati i risultati di questa analisi statistica.

5.8 Altri moduli della piattaforma

Oltre ai moduli presentati nei paragrafi precedenti, la piattaforma è dotata anche di altri semplici strumenti che offrono delle funzionalità utili, necessarie per task più complessi.

Il primo, già illustrato brevemente, è il chunks builder, impiegato in diversi contesti per serializzare dati in file aventi dimensione fissa e nomi fedeli a pattern specificati dall'utente. La sua capacità di scrivere dati progressivamente, a partire dall'ultimo chunk presente nella cartella di destinazione, lo rende adatto, in modo particolare, a gestire ripetute sessione di crawling effettuate ad intervalli di tempo irregolari: i nuovi tweet, infatti, vengono automaticamente aggiunti all'ultimo file preesistente che, raggiunto il limite prefissato, sarà chiuso in favore di un nuovo chunk. Quest'ultimo presenterà una numerazione consistente con i file già presenti nella cartella, risultati da precedenti attività del crawler.

La piattaforma, inoltre, è dotata di un modulo per interfacciare il sistema con Google Cloud Storage. Esso può essere utilizzato per ispezionare il contenuto di un bucket, elencare i file appartenenti ad una cartella, inviare oggetti dal file system locale o scaricarli dal cloud.

Infine, è presente uno strumento per effettuare l'external sort, un algoritmo di ordinamento di big data che non richiede il caricamento dell'intero dataset in memoria. Utilizzando un approccio che prevede ordinamenti parziali e serializzazioni successive su file, i dati di input vengono ordinati con impiego ordinario di memoria. Questo algoritmo è stato impiegato, ad esempio, per ordinare i valori della KLD al termine del job MapReduce ed evidenziare quelli più significativi.

Capitolo 6

Esperimenti con la piattaforma

6.1 Valutazione delle performance dei cluster

6.1.1 Dataset di input

6.1.2 Protocollo sperimentale

6.1.3 Risultati

Nel seguente paragrafo sono riportati i risultati delle analisi illustrate in precedenza; le considerazioni maturate dalla loro osservazione, invece, sono rimandate a quello successivo.

Vengono presentati, innanzitutto, i tempi di esecuzione dei task di indicizzazione. Come anticipato nel paragrafo 5.X, il procedimento è stato applicato sul corpus itWaC (dopo una sua opportuna conversione in testo semplice) e sulle classi che hanno mostrato ricevere più tweet. Per verificare l'effettiva utilità di tecnologie cloud per queste tipologie di elaborazioni è stato condotto uno studio comparativo delle performance dei cluster di Hadoop, facendo variare sia il numero di nodi interconnessi, sia la tipologia di macchine di Compute Engine.

Le configurazioni di cluster scelte sono state limitate da alcuni vincoli che Google impone agli sviluppatori in possesso di account senza determinati privilegi (per ottenere la rimozione di queste restrizioni è necessario compilare una richiesta formale al cloud provider e spiegare le motivazioni per cui si vogliono impiegare più istanze di Compute Engine). È stato possibile avviare, per questo motivo, al massimo 23 macchine diverse contemporaneamente e suddividere tra di esse 24 CPU virtuali, di seguito nominate solo CPU per comodità. Per poter osservare le variazioni nei tempi di indicizzazione sono state selezionate configurazioni di cluster aventi stesso numero di nodi con

macchine di tipi diversi e configurazioni con un numero di nodi variabile, ma tutti dello stesso tipo. Per la natura dei task da completare si è preferito scegliere macchine dotate di CPU performanti con un quantitativo di memoria RAM sufficiente per eseguire i job, ma comunque modesto se paragonato a quello massimo consentito da Compute Engine.

Tutti i job di Hadoop sono stati ripetuti tre volte, al fine di minimizzare, nei risultati, la componente di variabilità dovuta ad errori nelle macchine del cluster, problemi di I/O, lentezza della rete, ecc.: quelli che vengono presentati sono i tempi calcolati con media aritmetica. Solo una volta, durante l'esecuzione di un job, è stato ricevuto un backend error, che non ha comunque pregiudicato il completamento del processo di indicizzazione (è stato registrato un ritardo di 30 secondi).

Sono riportati, inoltre, i tempi medi di avvio e di spegnimento dei cluster di Hadoop. Questi risultati sono stati calcolati misurando il tempo impiegato da Compute Engine per completare l'esecuzione di alcuni script, che automatizzano l'accensione/spegnimento delle macchine, l'interscambio di chiavi per la connessione SSH, la configurazione di Hadoop e l'avvio/terminazione dei suoi daemon. Nessun file necessario ai test è stato caricato sulle macchine in fase di deploy del cluster, presumendo che i file della piattaforma siano presenti su dischi persistenti: i risultati, pertanto, non sono affetti da ritardi dovuti a tempi variabili di upload.

Le performance dei cluster sono, inoltre, messe a confronto con quelle ottenute da una macchina locale eseguente un cluster di Hadoop in modalità standalone sugli stessi dati in input. I job sono stati eseguiti tre volte anche su questa macchina e quelli presentati sono i tempi medi di esecuzione. La macchina locale possiede un processore Intel Core i5-3570K, 3.40GHz con 16GB di memoria RAM. Per le specifiche tecniche delle macchine di Compute Engine, invece, si rimanda al paragrafo 4.2.1.

La tabella X.X riporta i dati raccolti.

TABELLA 1

A conclusione del paragrafo vengono presentati i risultati dell'analisi statistica sui tweet attraverso la divergenza di Kullback-Leibler con il corpus itWaC. Il calcolo della KLD e l'ordinamento dei risultati vengono eseguiti in tempi molto brevi per via della modesta dimensione degli indici. Per tale motivo non si è ritenuto significativo inserirli.

La tabella X.X riporta, per ciascuna classe, le prime 25 parole più significative calcolate con i job di MapReduce (i seed usati in fase di crawling sono messi in evidenza). Nella tabella X.X, invece, sono stato omessi i seed e le stopwords, al fine di ottenere risultati più leggibili.

TABELLA 2

TABELLA 3

6.1.4 Discussione dei risultati

Alla luce dei risultati illustrati in precedenza è possibile presentare delle conclusioni sia sulle prestazioni ottenute dai cluster di Hadoop, sia sulle parole emerse calcolando la divergenza di Kullback-Leibler tra le distribuzioni di probabilità dei termini nei documenti.

Osservando la tabella X.X emergono, innanzitutto, alcuni risultati facilmente ipotizzabili già in fase di progettazione dei test. Risulta evidente che, mantenendo costante il numero di nodi nel cluster, le prestazioni migliorano con l'aumentare del numero complessivo di CPU a disposizione, ovvero scegliendo macchine progressivamente più potenti. Quanto detto è vero sia per grandi dataset (itWaC) che per altri più piccoli (i tweet).

Il ragionamento inverso non trova invece riscontri empirici. Mantenendo costante il tipo di macchine del cluster e facendo variare il numero di nodi si osservano risultati diversi in funzione della dimensione dei dati di input. Per grandi dataset si nota un miglioramento delle performance all'aumentare del numero di nodi interconnessi, verosimilmente perché maggiori risorse computazionali sono a disposizione del framework. Quando i dati di input sono di meno, invece, le prestazioni migliorano progressivamente fino ad un certo punto, per cominciare poi a degradare. Il motivo può essere riconducibile al fatto che, oltre una certa soglia, il vantaggio guadagnato con l'aumento di CPU viene perso nelle fasi di interscambio di messaggio e di dati tra i nodi del cluster. Questo risultato induce a pensare che, aumentando notevolmente il numero di nodi interconnessi, anche con grandi dataset si potrebbe verificare un fenomeno simile: accurati tuning del cluster sono, pertanto, sempre consigliati.

In generale è possibile osservare come le migliori prestazioni su grandi dataset siano state ottenute da cluster con moltissimi nodi, seppur costituiti da macchine del tipo formalmente meno potente. Su piccoli dataset, invece, le prestazioni migliori sono state registrate da cluster formati da due soli worker, ma del tipo migliore sulla carta.

Effettuando un confronto con la macchina locale eseguente Hadoop in modalità standalone, emergono indiscutibilmente i vantaggi di non dover effettuare scambi di dati tra nodi durante l'indicizzazione dei tweet: i task vengono completati in pochi secondi, con prestazioni nettamente migliori di quelle di qualunque cluster. Il risultato opposto si osserva con il dataset di itWaC: la macchina locale registra delle prestazioni nettamente peggiori di quelle del cluster meno performante (xs vs xs). Il confronto, poi, con la configurazione migliore del cluster () dovrebbe essere sufficiente a dimostrare la potenza del framework, che impiega il X% di tempo in meno di un computer di fascia medio-alta.

I tempi di accensione e spegnimento del cluster sono, tutto sommato, soddisfacenti. I cluster con un maggior numero di nodi impiegano, ragionevolmente, più tempo degli altri. Stesso discorso per i tempi di spegnimento, anche se qui le differenze si attenuano.

Si fa notare, inoltre, la grande somiglianza nelle performance del cluster costituito da quattro worker, ciascuno con quattro CPU, e di quello costituito da due worker dotati ciascuno di otto CPU (sedici CPU in totale per entrambe le configurazioni).

6.2 Valutazione delle caratteristiche lessicali dei tweet

6.2.1 Dataset di input

6.2.2 Risultati

6.2.3 Discussione dei risultati

Parte III

Conclusione

Lesson-learnt

Sviluppi futuri

Il caso di studio presentato si presta ad essere espanso in diverse direzioni. La piattaforma sviluppata può essere infatti impiegata in molti modi ed offre una collezione di strumenti per integrare facilmente nuovi moduli con i prodotti della Google Cloud Platform. Apache Hadoop si è rivelata essere una tecnologia adatta per elaborare grandi moli di informazioni ed il suo utilizzo è pertanto suggerito anche per nuovi task, che richiederanno, tuttavia, una progettazione ex novo delle funzioni map e reduce: queste, infatti, sono specifiche per determinati compiti e difficilmente riusabili.

I primi moduli da integrare nella piattaforma sono quelli destinati ai task di data visualization, già sviluppati nel corso di altri lavori. Questi si rendono necessari per visualizzare la Mappa dell'Intolleranza e le zone maggiormente affette dai problemi di discriminazione. È auspicabile che tali moduli facciano uso di tecnologie big-data-oriented nel loro core per beneficiare di grande scalabilità.

L'utilizzo di BigQuery, un altro prodotto della Google Cloud Platform, potrebbe essere impiegato per fornire analisi in tempo reale sulle situazioni di intolleranza in divenire nel nostro Paese. La sua capacità di processare molti terabyte di dati in pochi secondi potrebbe fornire alle amministrazioni locali uno strumento più potente per monitorare nascenti episodi di discriminazione, dal momento che le notizie su Twitter, come anche negli altri social network, si diffondono in tempi molto brevi.

L'analisi statistica dei tweet, come è stato dimostrato, non fornisce strumenti sufficientemente efficaci per combattere le discriminazioni: si rende necessario, dunque, uno strumento capace di condurre indagini sulla semantica dei messaggi. Annotando manualmente dei campioni piuttosto grandi di tweet, si potrebbero ottenere corpus di messaggi discriminatori, possibilmente uno per ciascuna delle categorie di individui da tutelare. Tali corpus potrebbero essere impiegati per la costruzione di un classificatore in grado di determinare se nuovi tweet presentano tracce di intolleranza.

Una prevenzione più efficace degli episodi di discriminazione si potrebbe ottenere ampliando il raggio d'azione delle analisi, non limitandosi al con-

tenuto dei messaggi, ma indagando sulla natura stessa del grafo del social network. Se, ad esempio, molti messaggi antisemiti provenissero dallo stesso utente si potrebbe ritenere opportuno analizzare anche gli altri suoi precedenti messaggi, per determinare se la persona, in effetti, prova e manifesta sistematicamente sentimenti di odio verso gli ebrei. Si potrebbe pensare di condurre le stesse analisi anche sulle persone a lei correlate (followers su Twitter, amici su Facebook, ecc.) per individuare tra di loro altri individui potenzialmente pericolosi. Così facendo potrebbe emergere, ad esempio, una rete di razzisti, la cui attività dovrebbe essere osservata con maggiore attenzione. Individuando più utenti affetti da forme di intolleranza si potrebbe, inoltre, migliorare il classificatore precedentemente ipotizzato arricchendo il numero di feature nel training set.

Diversi studi[][][][][], infine, hanno dimostrato come sia possibile predire, con buona affidabilità, le personalità degli individui analizzando i loro messaggi sui social network (in modo particolare su Facebook e Twitter). Gli strumenti sviluppati si sono rivelati in grado di quantificare i Big Five personality traits (openness to experience, conscientiousness, extraversion, agreeableness, neuroticism), cinque ampie dimensioni della personalità, solitamente non sovrapposte negli esseri umani, definite dagli psicologi e con diversi riscontri scientifici, poiché valide per soggetti di sesso, età e culture differenti. Supponendo di aver individuato individui realmente pericolosi, questi strumenti potrebbero essere impiegati per indagare sui tratti della personalità tipici di un determinato profilo di discriminatore. Potrebbe così emergere, per esempio, che i razzisti hanno generalmente livelli bassi di estroversione, oppure che l'odio contro le donne è spesso manifestato in individui caratterizzati da assenza di apertura mentale verso nuove esperienza, e così via.

Ringraziamenti

Ringrazio...

Parte IV

Bibliografia