

Definizione ed Implementazione di un Sistema di Raccomandazione Distribuito per film e Modellazione di Eventi Complessi

Prof. Ing. Tommaso di Noia
Prof.ssa Marina Mongiello
Mauro Losciale
Pietro Tedeschi



**Logica e Intelligenza Artificiale
Ingegneria del Software Avanzata
Laurea Magistrale in Ingegneria Informatica
Politecnico di Bari
A.A 2015 - 2016**

Indice

1	Introduzione	1
2	Stato dell'arte	1
2.1	Introduzione ai sistemi CEP	1
2.2	Sistemi di Raccomandazione	2
2.2.1	Metriche di Valutazione	5
2.3	Introduzione alla Matrix Factorization	6
2.4	Introduzione al Data Stream Processing	6
2.5	Il paradigma Publish-Subscribe	7
2.6	Il pattern Facade	7
2.7	Il pattern Singleton	7
2.8	Il pattern Model-View-Controller (MVC)	7
2.9	La tecnologia WebSocket	7
3	Analisi del progetto	7
4	Soluzione proposta	7
4.1	La libreria Spark	7
4.1.1	Spark Streaming	8
4.1.2	Spark MLlib	9
4.1.3	Spark SQL	9
4.2	Apache Kafka	9
4.2.1	Panoramica	9
4.2.2	Integrazione con Spark Streaming	9
4.3	Il framework Node.js	9
4.3.1	Panoramica	9
4.3.2	Kafka Client per Node.js	9
4.3.3	Il framework Angular.js	9
4.4	La libreria socket.IO	9
5	Conclusioni e sviluppi futuri	9
	Bibliografia	10

1 Introduzione

2 Stato dell'arte

2.1 Introduzione ai sistemi CEP

L'incremento dei dispositivi interconnessi e delle applicazioni distribuite, richiede un'elaborazione continua del flusso dati. Esempi di tali applicazioni, vanno dal traffico generato dalle Wireless Sensor Networks (WSN) al flusso dati relativo agli indici finanziari, dal monitoraggio stradale alla Clickstream Analysis.

Un sistema ad eventi complessi, meglio conosciuto come *Complex Event Processing* (CEP), modella il flusso informativo dei dati, visualizzando gli elementi come notifiche di ciò che sta accadendo nel mondo esterno. I dati vengono rilevati e filtrati utilizzando dei pattern (oppure le *processing rules*), i quali hanno il compito di rappresentare il modello di riferimento con l'informazione da rilevare, per poi farla pervenire alle rispettive parti (ad esempio, i dispositivi che effettuano una sottoscrizione ad un determinato topic nel paradigma *publish-subscribe*). L'obiettivo di un sistema CEP consiste nell'identificare eventi significanti e rispondere ad essi nel più breve tempo possibile. Un pattern o una regola, può essere definita mediante un linguaggio basato su query, il cosiddetto **Event Query Language**.

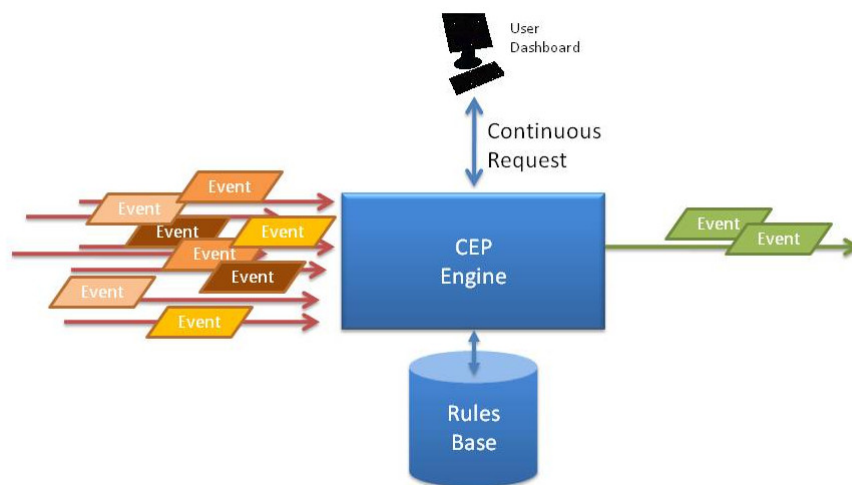


Figura 1: Architettura di un Sistema CEP [5]

Gli **Event Query Languages** possono essere raggruppati in tre categorie: Composition Operators, Data Stream Query Languages e Production Rules. I Composition Operators identificano gli eventi complessi partendo dalla composizione dei singoli eventi, utilizzando operatori quali congiunzione, negazione o di sequenza per la costruzione delle espressioni. Esempi rilevanti sono IBM Active Middleware Technology e ruleCore.

I **Data Stream Query Languages** sono basati sul linguaggio SQL; gli stream di dati sono semplicemente tuple convertite per database relazionali, in modo che si possano eseguire query SQL su di esse. E' utile citare i seguenti approcci: CQL, Coral8, StreamBase, Aleri, Esper e così via.

Le **Production Rules** specificano le azioni che devono essere eseguite quando il sistema si trova in determinati stati; non è un linguaggio ad eventi, ma costituisce un approccio importante nei sistemi CEP. Un esempio pratico è TIBCO Business Events.

Un altro fattore importante è il tempo. Sono due le parti da considerare quando si parla del tempo, il tempo della finestra ed il tempo dell'evento informativo. Il tempo della finestra mostra gli eventi che vengono esaminati in un determinato intervallo. Il tempo dell'evento invece, porta con sé informazioni relative alla data, ora di rilevazione, tempo di transizione, ed intervallo di elaborazione.

I contributi relativi ai sistemi CEP, arrivano da diverse comunità, a partire da quelle che si occupano di sistemi distribuiti, automazione industriale, sistemi di controllo, monitoraggio delle reti, Internet of Things, e middleware in generale.

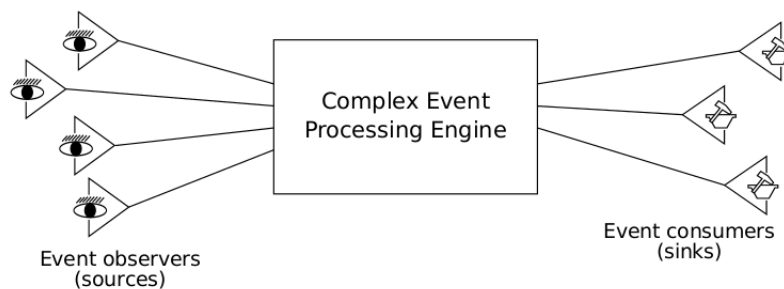


Figura 2: Funzionamento di un Sistema CEP (High-Level) [3]

Come possiamo vedere dalla Figura 2, viene associata una semantica dettagliata agli elementi informativi da processare. Da una parte abbiamo gli **Event Observer**, i quali rappresentano la sorgente dei dati e degli eventi da notificare; in seguito abbiamo il **CEP Engine**, responsabile del filtraggio e della notifica degli eventi ai nodi *sink*, identificati come **Event Consumers** [3, 4].

2.2 Sistemi di Raccomandazione

I sistemi di raccomandazione, raccolgono informazioni sulle preferenze di utente in corrispondenza di un insieme di elementi (ad esempio, film, musica, libri, giochi, viaggi, siti web, applicazioni, gadget). L'informazione può essere acquisita in maniera esplicita (tipicamente ciò viene fatto acquisendo il voto di un utente) o implicitamente (analizzando il comportamento dell'utente, ad esempio musica ascoltata, applicazioni scaricate, siti web visitati, libri letti). Inoltre i sistemi di raccomandazione, possono tener conto anche delle caratteristiche demografiche dell'utente (ad esempio età, nazionalità, sesso); dei contenuti informativi presenti nel mondo del Web 2.0, ad esempio all'interno delle piattaforme di Social Networking, quali follower, followed, twit, like, post; dei dati provenienti dai dispositivi caratterizzanti l'Internet of Things (ad esempio, coordinate GPS, RFID, segnali medici inviati in real-time).

I sistemi di raccomandazione utilizzano diverse sorgenti informative al fine di fornire all'utente finale una migliore Quality of Experience relativa alla predizione ed alla raccomandazione degli elementi che potrebbero interessargli. La tecnica del Collaborative Filtering (CF), ha un ruolo fondamentale nella raccomandazione, sebbene viene spesso usata anche con altre tecniche di filtraggio basate sul contenuto o basate sulla conoscenza. Il CF si basa sulla cronologia decisionale dell'utente: oltre alle nostre esperienze, facciamo le nostre decisioni anche in base alla conoscenza che ci circonda.

Il processo con cui un sistema di raccomandazione generi una raccomandazione, è basato sulla combinazione delle seguenti considerazioni:

1. La tipologia di dati disponibili nel database (votazioni, informazioni di registrazione dell'utente, caratteristiche peculiari del contenuto informativo, relazioni sociali)
2. L'algoritmo di filtraggio usato (demografico, basato sul contenuto, collaborativo, basato sulle relazioni sociali, dipendente dal contesto e ibrido).
3. Il modello scelto (basato sull'uso diretto dei dati: 'memory-based', oppure un modello generato usando tali dati: 'model-based').
4. Le tecniche impiegate: approccio probabilistico, reti Bayesiane, algoritmi di tipo nearest neighbors, algoritmi genetici, reti neurali, logica fuzzy.
5. Livello di dispersione del database e scalabilità desiderata.
6. Capacità di elaborazione del sistema (tempo di elaborazione e consumi di memoria).
7. L'obiettivo da raggiungere (predizioni e raccomandazioni)
8. La qualità del risultato desiderata (ad esempio la precisione).

La ricerca nell'ambito dei sistemi di raccomandazione, richiede che i dati siano di dominio pubblico, al fine di semplificare la ricerca sulle tecniche innovative relative all'analisi dei dati. Esempi di dataset pubblici presenti in letteratura, sono Last.Fm, Delicious, Netflix, MovieLens. Le funzionalità interne per i sistemi di raccomandazione, sono caratterizzate dagli algoritmi di filtraggio. Gli algoritmi di filtraggio vengono classificati nel modo seguente:

- Collaborative Filtering
- Demographic Filtering
- Content-Based Filtering
- Hybrid Filtering

Il **Content-Based Filtering** consente di creare raccomandazioni basate sulle scelte fatte in passato da un utente (ad esempio, in un sito E-Commerce, se l'utente ha acquistato una fiction cinematografica, probabilmente il sistema di raccomandazione gli consiglierà una fiction recente, che non ha ancora acquistato sul sito). La tecnica consente inoltre di generare la raccomandazione utilizzando il contenuto dell'oggetto, ad esempio il testo, le immagini, l'audio.

Il **Demographic Filtering** si basa sul principio che gli individui con caratteristiche personali comuni, quali età, sesso, luogo di residenza e così via, avranno le stesse preferenze.

Il **Collaborative Filtering** consente agli utenti di attribuire un voto ad un insieme di elementi (filmati, canzoni, film, libri, all'interno di una piattaforma web) salvando le proprie preferenze all'interno di un database, e consentendo di creare una raccomandazione specifica per ogni utente. I voti degli utenti possono essere anche acquisiti in maniera implicita (ad esempio il numero delle volte che viene ascoltata una canzone, il numero delle consultazioni relative ad una risorsa). L'algoritmo utilizzato maggiormente per il Collaborative Filtering è il k Nearest Neighbors (k NN).

Nella versione "user to user", il KNN esegue i seguenti task per generare la raccomandazione:

1. Determinare i k utenti vicini all'utente corrente.
2. Implementare un approccio che tenga conto degli elementi "vicini" non ancora votati dall'utente corrente.

3. Estrarre le predizioni dal passo 2 e selezionare le N raccomandazioni.

La similarità può essere distinta in **item-item**: due elementi sono simili se tendono ad ottenere lo stesso rate dagli utenti; oppure **user-user**: due utenti sono simili se tendono a dare lo stesso rate agli elementi.

Uno dei problemi più noti del filtro collaborativo è il "*Cold Start*", che si potrebbe verificare nel caso in cui si considerano elementi che non sono stati votati in precedenza.

L'**Hybrid Filtering** è una combinazione di Collaborative Filtering e Demographic Filtering, oppure una combinazione tra Collaborative Filtering e Content-Based Filtering che sfrutta i pregi di ciascuna di queste tecniche. Il metodo è basato su metodi probabilistici come gli algoritmi genetici, genetica fuzzy, reti neurali, reti Bayesiane, clustering.

Inoltre possiamo suddividere i metodi in *Memory-Based* e *Model-Based*:

I **Memory Based** conservano in memoria le informazioni associate ad ogni utente, item o voto all'interno del sistema. Queste informazioni costituiscono la *Knowledge Base* sulla quale lavora l'algoritmo di predizione. Idealmente i sistemi di tipo memory based devono essere in grado di generare l'insieme di predizioni in maniera efficiente, processando tutte le informazioni contenute nella matrice *Utenti/Item*.

La matrice *Utenti/Item* definita all'interno del sistema di raccomandazione contiene entry i, j che rappresentano un voto dell'utente i -esimo per l'elemento j -esimo; nel caso in cui dovesse mancare una preferenza per un determinato item, il valore viene posto a 0.

$$M = \begin{matrix} & i_1 & i_2 & \dots & i_j & \dots & i_m \\ \begin{matrix} u_1 \\ u_2 \\ \dots \\ u_i \\ \dots \\ u_n \end{matrix} & \begin{bmatrix} \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & r_{i,j} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix} \end{matrix}$$

Poiché i sistemi di raccomandazione sono caratterizzati da una elevata dimensione dello spazio degli utenti e degli item, il funzionamento degli algoritmi *memory-based* è stato definito sull'ipotesi di determinare un grado di similarità tra gli utenti, che permetta di estrapolare dalla matrice dei voti, le informazioni associate ai soli utenti simili all'utente attivo.

I **Model Based**, utilizzano l'insieme dei voti espressi dagli utenti per costruire un modello statistico di preferenze su cui generare le predizioni.

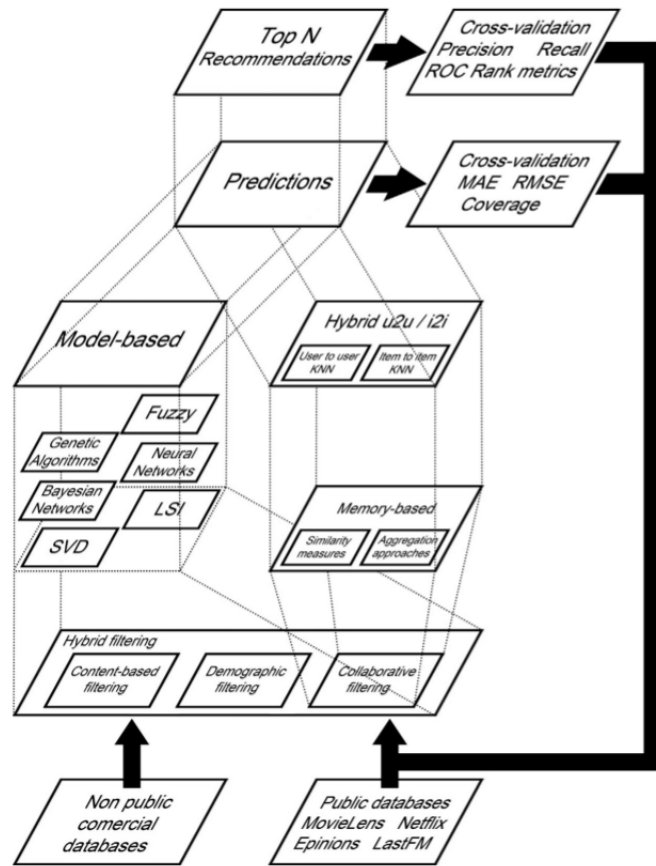


Figura 3: Modelli di Raccomandazione [2]

E' utile citare i metodi di riduzione basati sulla Matrix Factorization, la tecnica model-based Latent Semantic Index (LSI), il metodo di riduzione Singular Value Decomposition (SVD) e le tecniche di clustering.

Nella Figura 3 viene illustrato un diagramma con le tecniche principali e gli algoritmi di raccomandazione maggiormente utilizzati in letteratura [2].

2.2.1 Metriche di Valutazione

La qualità di un sistema di raccomandazione può essere valutata dai risultati forniti in output. Le tipologie di metriche usate, dipendono dal tipo di CF usato. Le metriche possono essere categorizzate in: Predictive Accuracy, come ad esempio il Mean Absolute Error (MAE) e sue varianti; Classification Accuracy metrics, come la precision, recall, F1-measure, e sensibilità ROC; Rank Accuracy, come il Mean Average Precision (MAP), e così via. Desideriamo concentrarci sulla metrica **Root Mean Squared Error** (RMSE). Infatti la radice quadrata dell'errore quadratico medio, è una metrica molto utilizzata per la raccomandazione di film.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i,j} (p_{i,j} - r_{i,j})^2}$$

dove n è il numero totale dei voti assegnati dagli utenti, $p_{i,j}$ è il voto predetto per l'utente i sull'elemento j , ed $r_{i,j}$ è il voto attuale. L'RMSE consente di valutare in maniera precisa i contributi degli errori assoluti tra i valori predetti ed i valori reali [6].

2.3 Introduzione alla Matrix Factorization

I modelli di Matrix Factorization possono essere utilizzati per ricercare quali sono i fattori latenti che caratterizzano le interazioni tra due tipologie di entità, tale che le interazioni user-item siano modellate come prodotti interni nello spazio. Ad esempio, due utenti potrebbero votare un determinato film con un voto alto, se ad entrambi piacciono gli attori, o il genere del film, ecc. Quindi, se si riescono a scoprire quali sono i fattori latenti dietro la scelta di un utente, si riesce anche a predire un voto o una possibile preferenza di un altro film rispetto all'utente, perchè le feature associate all'utente dovrebbero convergere alle feature associate all'elemento.

Analizziamo la tecnica dal punto di vista matematico. Sia U l'insieme degli utenti, e sia D l'insieme degli elementi. Sia R la matrice dei voti assegnati dagli utenti agli elementi, di dimensione $|U| \times |D|$. Supponiamo di voler cercare i K fattori latenti. Innanzi tutto bisogna cercare le due matrici, P (una matrice $|U| \times K$) e Q (una matrice $|D| \times K$) tale che il loro prodotto sia una stima del tipo: $R \approx P \times Q^T = \hat{R}$. Ogni riga di P rappresenta il grado di associazione tra un utente e le feature, mentre ogni riga di Q tra un elemento e le feature. Al fine di poter calcolare la predizione di un rating su un elemento d_j votato dall'utente u_i , definiamo il prodotto interno tra i due vettori corrispondenti come:

$$r_{i,j} = p_i^T q_j = \sum_1^k p_{ik} q_{kj}$$

Quindi un modo per ovviare a questo problema consiste nell'inizializzare le due matrici con alcuni valori, e calcolare la differenza tra i valori stimati ed i valori reali, e infine minimizzare questa differenza in maniera iterativa. Un metodo per minimizzare l'errore calcolato può essere l'algoritmo di discesa del gradiente stocastico oppure l'algoritmo **Alternating Least Squares** (ALS) [7].

In generale, mentre l'algoritmo della discesa del gradiente stocastico è più semplice e più veloce di ALS, ALS risulta essere preferibile in molteplici casi, tra cui la parallelizzazione computazionale. In ALS infatti, il sistema calcola ogni vettore d_j indipendentemente dagli altri elementi, così come accade per u_i . Ciò comporta un incremento della parallelismo a livello computazionale.

2.4 Introduzione al Data Stream Processing

Di recente, è nata una nuova classe di applicazioni *data-intensive*, nei quali i dati vengono modellati come relazioni transienti piuttosto che persistenti. Definiamo con il termine Data Stream, una sequenza di elementi codificati in digitale rappresentanti un'informazione. Esempi di applicazioni **Data Stream** vanno dal campo finanziario, monitoraggio del traffico di rete, applicazioni web, sicurezza, telecomunicazioni, gestione dati e reti di sensori. Nel modello Data Stream, i dati possono essere tuple relazionali, come ad esempio misurazioni di rete, registro chiamate, pagine web visitate, dati provenienti da sensori, e così via. Tuttavia, questi dati costituiscono un flusso multiplo e tempo variabile, tali da richiedere una nuova tecnologia al fine di poter esser gestiti.

Nei vari scenari applicativi sopra citati, non è efficiente e ne tanto meno semplice gestire i dati in arrivo all'interno di un Database Management System (DBMS) tradizionale. Infatti, i DBMS tradizionali non sono progettati per trattare in maniera rapida un flusso di dati contiguo. Di conseguenza sono nati i Data Stream Management System (DSMS) in grado di gestire un flusso dati continuo. I DSMS sono in grado di offrire un'elevata flessibilità nell'eseguire query continue sullo stream dati. Poichè i DSMS sono data-driven, ogni qualvolta viene eseguita una query, essa produrrà nuovi risultati non appena arrivano nuovi dati da processare.

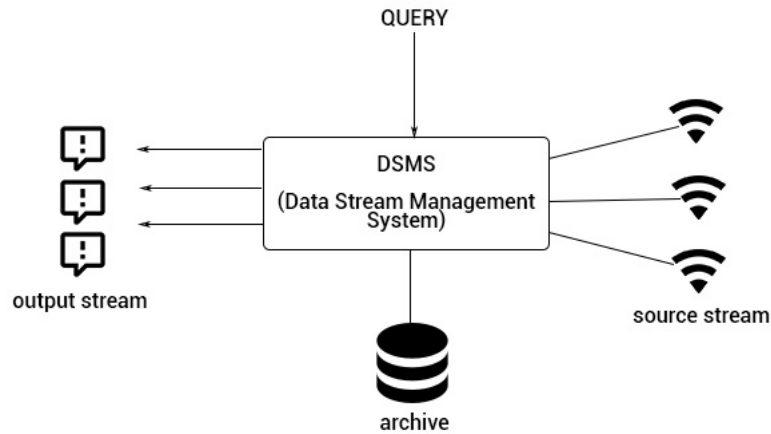


Figura 4: Architettura DSMS

I dati presenti all'interno di uno stream, arrivano online. Il sistema che elabora lo stream, non gestisce l'ordine di arrivo degli elementi da processare. Gli stream inoltre hanno la peculiarità di non avere una dimensione massima prefissata, ma sono altamente variabili. Dopo che il dato è stato processato, o viene scartato, oppure viene archiviato all'interno di uno spazio di storage [1].

2.5 Il paradigma Publish-Subscribe

2.6 Il pattern Facade

2.7 Il pattern Singleton

2.8 Il pattern Model-View-Controller (MVC)

2.9 La tecnologia WebSocket

3 Analisi del progetto

4 Soluzione proposta

4.1 La libreria Spark

Apache Spark è un sistema di cluster computing di tipo general-purpose, scalabile e veloce. Dispone di API di alto livello in **Java**, **Scala**, **Python** ed **R**, e un engine ottimizzato che supporta grafi di esecuzione generici. Supporta inoltre un ampio set di tool come **Spark SQL**, per structured data processing, **MLlib** per il machine learning e **Spark Streaming**, descritti nelle sezioni successive. Spark è eseguibile sia su sistemi Windows che UNIX-like (Linux, Mac OS).

Una delle possibili configurazioni di un sistema Spark è la modalità *cluster*, mostrata in Figura 5.

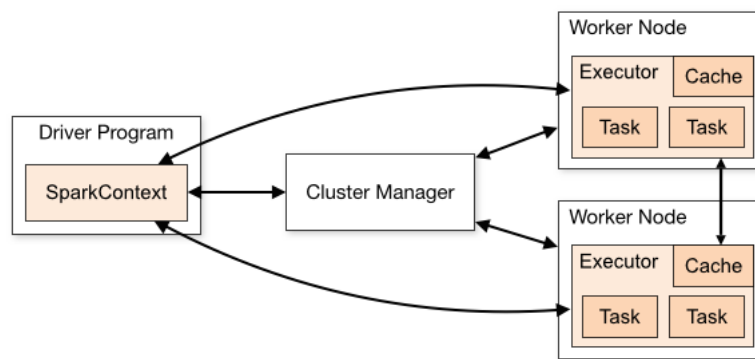


Figura 5: Configurazione in Spark di tipo Cluster Mode

Le applicazioni Spark sono eseguite come un set di processi indipendenti sul cluster, coordinati dall'oggetto *SparkContext* del programma sorgente (detto **driver program**). Precisamente, il programma driver può connettersi su diversi tipi di *cluster managers* (ad esempio un cluster di tipo Standalone, Mesos o YARN), il quale alloca le risorse a disposizione delle applicazioni. Una volta connesso, Spark scansiona i nodi del cluster alla ricerca degli **executor** (detti anche *worker node*), i quali eseguono effettivamente i task e il data storage delle applicazioni. A questo punto il driver invia il codice dell'applicazione agli executor (tipicamente un file JAR o file Python incluso nello *SparkContext*) e schedula i task per l'esecuzione parallela.

Alcune considerazioni riguardo tale architettura sono:

- Ogni applicazione gestisce i propri workers, i quali restano attivi durante tutto il ciclo di vita ed eseguono task multipli in thread multipli. Questo implica un isolamento tra le applicazioni, sia lato scheduling (ogni driver schedula i propri tasks) sia lato executor (tasks relativi ad applicazioni differenti risiedono in JVM differenti). Tuttavia ciò implica che non è possibile condividere nativamente i dati tra applicazioni diverse, a meno di utilizzare uno storage system esterno.
- Il driver deve poter gestire le connessioni con i workers durante l'intero ciclo di vita dell'applicazione. Per questo motivo dev'essere sempre garantita la visibilità a livello di rete tra driver e workers durante l'esecuzione.
- È necessario che driver e worker abbiano, a livello di rete, una distanza relativamente breve, preferibilmente nella stessa LAN, affinché lo scheduling sia rapidamente eseguito.

Il principio di funzionamento di Spark si basa sostanzialmente sul concetto di *Resilient Distributed Dataset (RDD)*. Un RDD è una collezione di dati su cui è possibile operare parallelamente, ed è distribuita su tutti i nodi del cluster come file system Hadoop oppure è generata da una collezione esistente in Java o Scala.

Una seconda astrazione è rappresentata dalle variabili condivise (*shared variables*), utilizzate nelle computazioni parallele. Di default Spark tiene traccia delle variabili istanziate nei vari task, e consente se necessario di condividerle fra task o fra task e driver. Le variabili condivise possono essere di due tipi: di tipo *broadcast*, il cui valore viene salvato nella cache per ogni nodo, e di tipo *accumulatore*, per esempio contatori o sommatore.

4.1.1 Spark Streaming

Spark Streaming è un'estensione delle Core API di Spark per lo **stream processing** di live data streams ad alto throughput. Supporta molteplici sorgenti di data stream come **Kafka**, Flume,

Twitter, ZeroMQ, Kinesis o socket TCP, i quali possono essere processati tramite direttive come *map*, *join*, *reduce* e *window*. Nel post processing è possibile salvare i data stream in un file system, in un database o visualizzarli in una live dashboard. Come ulteriore fase nella pipeline di operazione rientra anche il machine learning ed il graph processing. In Figura 6 viene riassunta l'architettura descritta.



Figura 6: Architettura di Spark Streaming

Nello specifico, i data streams ricevuti vengono suddivisi in frammenti (*batches*), processati da Spark per generare lo stream finale risultante in batches, come mostrato in Figura 7.



Figura 7: Spark Streaming Data Stream Processing

A livello alto il flusso continuo di dati è rappresentato da una struttura astratta detta *discretized stream* o *DStream*, il cui contenuto è rappresentato da tutte le sorgenti collegate eventualmente con Spark, o da stream risultanti da altri DStream. Internamente, un DStream è rappresentato tramite una sequenza di RDD.

4.1.2 Spark MLlib

4.1.3 Spark SQL

4.2 Apache Kafka

4.2.1 Panoramica

4.2.2 Integrazione con Spark Streaming

4.3 Il framework Node.js

4.3.1 Panoramica

4.3.2 Kafka Client per Node.js

4.3.3 Il framework Angular.js

4.4 La libreria socket.IO

5 Conclusioni e sviluppi futuri

Riferimenti bibliografici

- [1] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.
- [2] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Know.-Based Syst.*, 46:109–132, July 2013.
- [3] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, June 2012.
- [4] Lajos Jeno Fülöp, Gabriella Tóth, Róbert Rácz, János Pánczél, Tamás Gergely, Arpád Beszédes, and Lóránt Farkas. Survey on complex event processing and predictive analytics. *Nokia Siemens Networks*, 2010.
- [5] Patrick Gantet Patrick Gantet. Complex event processing.
- [6] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.
- [7] Gábor Takács, István Pilászy, Botyán Németh, and Domonkos Tikk. Matrix factorization and neighbor based algorithms for the netflix prize problem. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 267–274, New York, NY, USA, 2008. ACM.