Approfondimento sullo stack tecnologico di Spark

COURSE NAME

Big Data Analytics

Author:

Angelo Longano (196528)

September 12, 2024

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Contents

Abstract

Apache Spark è un framework open source per il calcolo distribuito, progettato per gestire grandi quantità di dati in modo più rapido e scalabile rispetto a Hadoop MapReduce.
Utilizza i Resilient Distributed Datasets (RDD) per elaborare i dati in parallelo, riducendo gli accessi al disco e migliorando le prestazioni.
Questo articolo presenta lo stack tecnologico di Spark, altamente modulare e scalabile. Il nucleo di Spark è lo Spark Core, che implementa il paradigma RDD e si integra con vari moduli per ottimizzare l'uso in diversi ambiti. Tra questi moduli troviamo Spark SQL, Spark Streaming, MLlib e GraphX, che supportano rispettivamente applicazioni SQL, streaming di dati, machine learning e analisi di grafi.

Infine, l'articolo illustra un esempio di implementazione di un classificatore multiclasse con RandomForest, utilizzando il linguaggio Python e un container Docker per eseguire Spark in modalità standalone.

1 Methodology

1.1 Introduzione

Apache Spark è un framework open source per il calcolo distribuito sviluppato dall'AMPlab della Università della California e successivamente donato alla Apache Software Foundation.
Gestire molti dati è una sfida ardua e le tecnologie sql tradizionali non sono abbastanza dinamici e scalabili.
Le tecnologie NoSQL sono state sviluppate per risolvere questo problema e Spark è una di queste.
In particolare Apache Spark implementa un paradigma di calcolo distributa basato sugli RDD (Resilient Distributed Dataset) che permette di scrivere programmi paralleli in modo semplice e veloce.
Rispetto ad altre tecnologie come Hadoop che utilizzano il paradigma MapReduce, Spark è molto più veloce e scalabile.

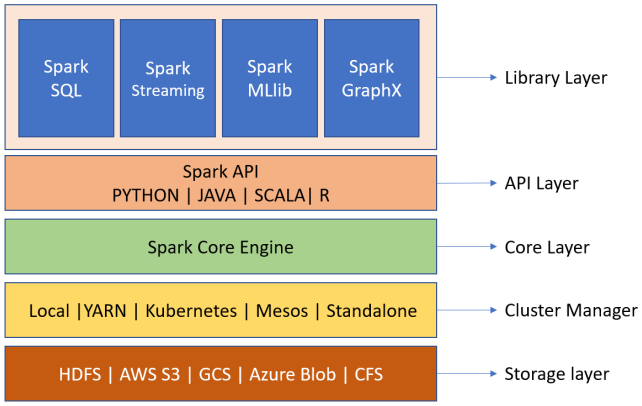

Figure 1: Immagine presa da un articolo di medium.com [1]

Lo stack della tecnologia Spark è altamente modulare e scalabile. Il cuore della tecnologia è nello Spark Core che implementa il paradigma RDD.
Grazie alle Spark API, la tecnologia può essere utilizzato con diversi linguaggi di programmazione come Scala, Java, Python e R.
Inoltre nello stack sono state implementate nelle versioni successive diverse librerie per utilizzare la tecnologia al meglio, ottimizzata e resa più user-friendly.
In particolare:

- Spark SQL: permette di utilizzare Spark con il linguaggio SQL
- Spark Streaming: permette di utilizzare Spark con lo streaming di dati
- MLlib: permette di utilizzare Spark per il machine learning
- GraphX: permette di utilizzare Spark per l'analisi di grafi

Per funzionare Spark necessita di un gestore di cluster e un sistema di archiviazione distribuita.
Il gestore di cluster è un software che permette di gestire le risorse di calcolo e di memoria di un cluster di macchine. In particolare Spark può essere utilizzato con diversi gestori di cluster come YARN, Mesos e Kubernetes.
Mentre il sistema di archiviazione distribuita è un sistema che permette di archiviare i dati in modo distribuito su un cluster di macchine. In particolare Spark può essere utilizzato con diversi sistemi di archiviazione distribuita come HDFS, S3 e Azure Blob Storage.
In questo articolo non verranno trattati i dettagli di archiviazione distribuita e la parte di sperimentazione utilizzerà Spark in modalità standalone.

1.2 Hadoop MapReduce vs Apache Spark

L'obiettivo di Spark era quello di creare un nuovo framework ottimizzato per l'elaborazione iterativa rapida come il machine learning e l'analisi interattiva dei dati, pur mantenendo la scalabilità e la tolleranza ai guasti di Hadoop MapReduce.
Questo modello di programmazione si ispira alla programmazione funzionale, secondo il principio "divide et impera": dividere il carico di lavoro in piccoli chunks (map) e aggregare insieme i risultati parziali ottenuti (reduce).
Lo sviluppatore deve solo preoccuparsi di implementare le funzioni map e reduce, mentre il framework si occupa di parallelizzare il calcolo e gestire la tolleranza ai guasti.
Questa soluzione è molto utile ed economica per gestire grandi quantità di dati, come ad esempio i dati gestiti da una data warehouse.
Non può essere utilizzata però per applicazioni che richiedono molte iterazioni sui dati, come ad esempio il machine learning.

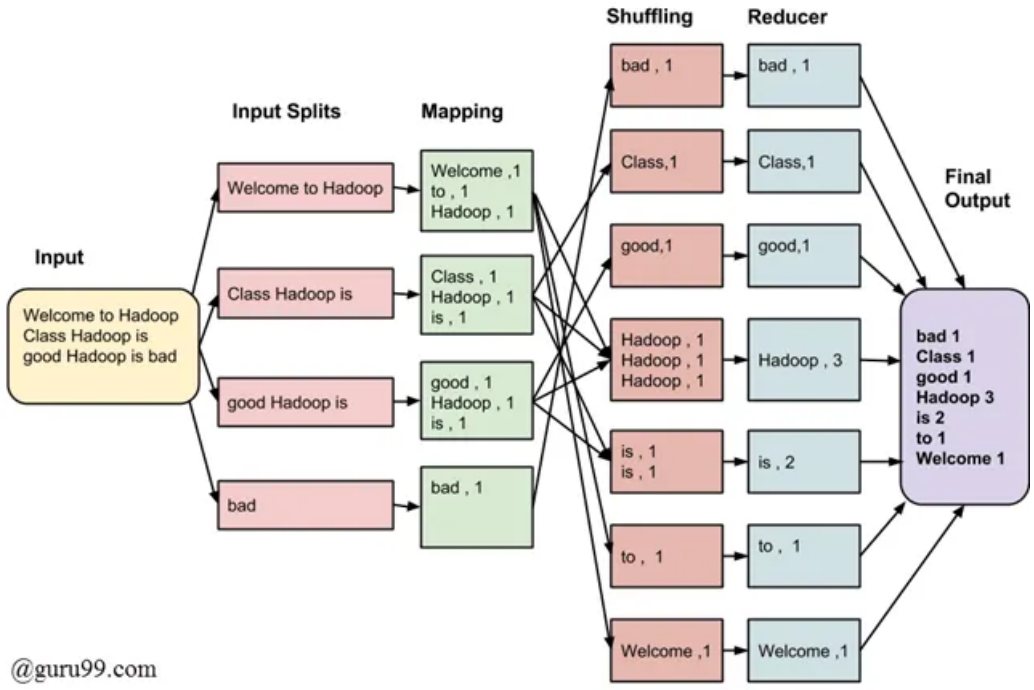

Figure 2: Immagine presa da un articolo di medium.com [2]

Come si può notare in figura 2, il modello MapReduce richiede 4 passaggi fondamentali:

- Splitting: il datasets grande viene diviso in piccoli chunks, che vengono processati in parallelo
- Mapping: i dati vengono trasformati in coppie chiave-valore
- Shuffling: i dati vengono ordinati e aggregati in base alla chiave
- Reducing: i dati vengono aggregati in base alla chiave

L'operazione di shuffling è molto costosa perché richiede tante operazioni di rete e di I/O.
Hadoop MapReduce utilizza il disco per memorizzare i dati e questo lo rende sia economico ma molto lento.
L'utilizzo ideale di questa tecnologia è per batch processing ripetitivo e sempre uguale nel tempo su grandi quantità di dati.

Il vantaggio di usare Spark rispetto a MapReduce è che Spark mantiene i dati in memoria, riducendo gli accessi al disco come si può notare in figura 3.
Inoltre Spark permette di effettuare più operazioni sui dati senza doverli riscrivere su disco, rendendo il calcolo iterativo molto più veloce.

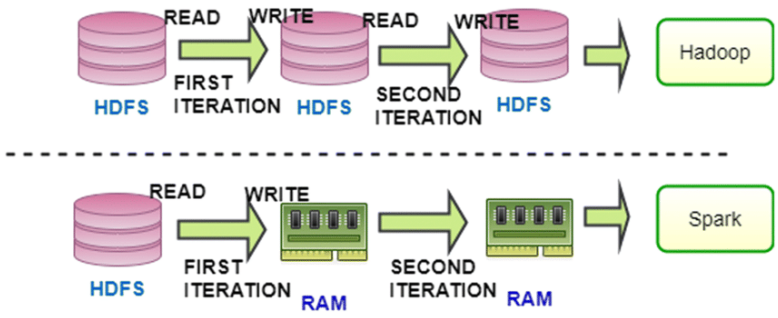

Figure 3: Confronto tra Spark e Hadoop sugli accessi alla memoria [2]

Apache Spark rappresenta un approccio più moderno e flessibile alla gestione dei BigData, fornisce delle API in più linguaggi di programmazione e permette di utilizzare diverse librerie per il machine learning, l'analisi di grafi e il processamento di streaming.
Inoltre Spark è molto più veloce di Hadoop MapReduce e permette di scrivere programmi più complessi in modo più semplice.
Rimane una soluzione più costosa di Hadoop MapReduce perché richiede più risorse di memoria e di calcolo.

1.3 Come funziona Apache Spark

Apache Spark si basa sul concetto di RDD (Resilient Distributed Dataset), ovvero un insieme di dati immutabile e distribuito che può essere processato in parallelo.
Ad alto livello, ogni applicazione Spark consiste in un programma driver che esegue la main function dell'utente e le varie operazioni parallele su un cluster.
Come si può vedere in figura 4, lo SparkContext in esecuzione nel nostro programma può chiedere di allocare e deallocare risorse con un cluster manager (es. YARN, Mesos) e può mandare task da fare eseguire ai nodi esecutori.

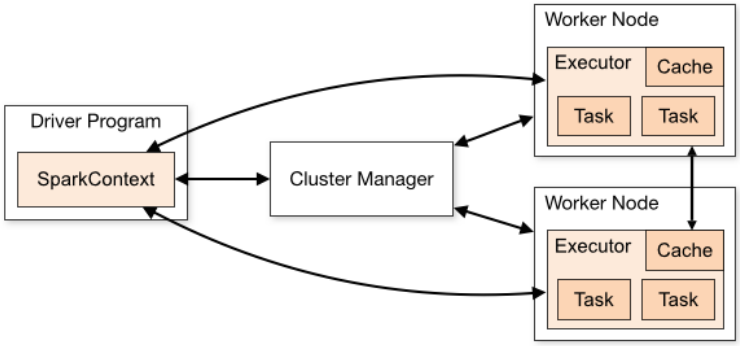

Figure 4: Architettura di esecuzione di Spark [3]

Come accennato precedentemente, la gestione dei cluster non verrà approfondita maggiormente e nell'esercitazione verrà utilizzato Spark in modalità standalone.

L'astrazione principale che Spark fornisce è un insieme di dati distribuiti resilienti (RDD), ovvero una raccolta

di elementi partizionati tra i nodi del cluster su cui è possibile operare in parallelo.
Gli RDD possono essere creati in due modi:

- Parallelizzando una collezione esistente nel programma driver
- Caricando un file in un file system distribuito (es. HDFS su Hadoop)

Inoltre un RDD può essere salvato in modo persistente in memoria, permettendogli di essere riutilizzato in modo efficiente attraverso operazioni parallele.
Un'altra caratteristica importante è che gli RDD si riprendono automaticamente dai guasti dei nodi.
Come si può vedere in figura 5, gli RDD supportano due tipi di operazioni:

- Trasformazioni: operazioni che producono un nuovo RDD a partire da uno esistente (es. map, filter, reduceByKey)
- Azioni: operazioni che restituiscono un valore al programma driver o salvano i dati su un file system (es. count, collect, save)

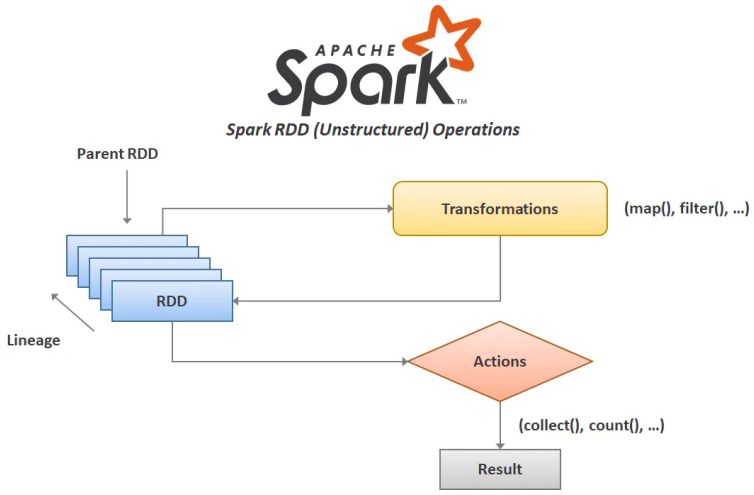

Figure 5: Flusso di operazioni sugli RDDs [4]

Tutte le trasformazioni in Spark sono operazioni "lazy", cioè vengono eseguite solo quando viene invocata un'azione che ha bisogno di fornire un risultato.
Di default ogni trasformazione deve essere ricompilata ad ogni esecuzione di un'azione, ma si può ottimizzare questo processo: salvando un RDD persistente su disco (o in cache).

Una seconda astrazione di Spark è rappresentata dalle variabili condivise che possono essere utilizzate nelle operazioni parallele.
Per impostazione predefinita, quando Spark esegue una funzione in parallelo come un insieme di task su nodi diversi, invia una copia di ogni variabile utilizzata nella funzione a ciascun task.
A volte, una variabile deve essere condivisa tra i task o tra i task e il programma driver.
Spark supporta due tipi di variabili condivise:

- Broadcast variables: permettono di mantenere una variabile in cache su tutti i nodi invece di inviarla con ogni task
- Accumulators: sono variabili che possono essere solo "aggiunte" attraverso un'operazione associativa e commutativa, come contatori o somme

1.4 Spark SQL, DataFrames e Datasets

Il concetto di RDD è centrale in Spark, ma nelle nuove versioni (>1.6) sono stati introdotti dei moduli specifici per utilizzare in modo più efficiente e semplice la tecnologia Spark.
Spark SQL è il modulo di Spark per l'elaborazione di dati strutturati.
A differenza dell'API RDD di basso livello, le interfacce fornite da Spark SQL forniscono a Spark maggiori informazioni sulla struttura dei dati e sul calcolo in corso.
Internamente, Spark SQL utilizza queste informazioni aggiuntive per eseguire ottimizzazioni supplementari.

A meno di specifiche esigenze, è consigliato dalla documentazione di Spark di utilizzare le nuove interfacce fornite da Spark SQL.
Quando si calcola un risultato, viene utilizzato lo stesso motore di esecuzione (Spark Core), indipendentemente dall'API o linguaggio utilizzato per esprimere il calcolo.
Questa unificazione significa che gli sviluppatori possono facilmente passare da un'API all'altra in base al modo più naturale di esprimere una determinata trasformazione.
Spark SQL può anche essere usato per leggere i dati da un'installazione Hive esistente.
Quando si esegue SQL da un linguaggio di programmazione, i risultati vengono restituiti come Dataset o DataFrame.
Un Dataset è una collezione distribuita di dati e utilizzabile con le Dataset API solo in Java e Scala.
Python non ha il supporto per queste API, ma data la sua natura dinamica, molti dei vantaggi dell'API Dataset sono già disponibili (ad esempio, è possibile accedere al campo di una riga per nome, naturalmente row.columnName).
Un DataFrame è un DataSet organizzato in colonne denominate. È concettualmente equivalente a una tabella in un database relazionale o a un dataframe di Pandas, ma con ottimizzazioni più ricche.
I DataFrame possono essere costruiti da un'ampia gamma di fonti, come: file di dati strutturati, tabelle in Hive, database esterni o RDD esistenti.
L'API DataFrame è disponibile in Scala, Java, Python e R.

1.5 Spark MLlib

MLlib è la libreria di Spark dedicata al machine learning con l'obiettivo renderlo scalabile e facile da implementare. Ad alto livello, fornisce strumenti quali:

- Algoritmi di ML: algoritmi di apprendimento comuni come classificazione, regressione, clustering e filtraggio collaborativo.
- Featurizzazione: estrazione, trasformazione, riduzione della dimensionalità e selezione delle caratteristiche.
- Pipeline: strumenti per la costruzione, la valutazione e la messa a punto di pipeline di ML.
- Persistenza: salvataggio e caricamento di algoritmi, modelli e Pipeline.
- Utilità: algebra lineare, statistica, gestione dei dati, ecc.

Con MLlib è possibile utilizzare sia le API RDD che i DataFrame per costruire e addestrare modelli di machine learning.
Nella sperimentazione verranno utilizzate le nuove API DataFrame, per implementare un semplice predittore multiclasse con il RandomForest.
Verranno definiti dei trasformer per la preparazione dei dati e insieme allo stimatore verrà costruito un pipeline per l'addestramento del modello.
Il dataset è stato preso dal sito del famoso ricercatore Chih-Jen Lin e contiene 60 mila esempi, ognuno con 126 attributi e classificati in 3 classi.
Per l'implementazione è stato usato un notebook Jupyter che esegue all'interno di un container Docker, il tutto verrà spiegato nella prossima sezione.

1.6 Utilizzo di Spark con PySpark e Docker

Per la sperimentazione allegata a questo articolo, verrà utilizzato PySpark, ovvero l'API di Spark per il linguaggio Python.
Per questioni di praticità e di portabilità, verrà utilizzato un'immagine Docker con tutti i moduli necessari per poter eseguire del codice python su un Jupiter Notebook connettendosi a Spark.
Tutte le indicazioni per l'installazione e l'utilizzo sono presenti nel README allegato alla repository.
Di seguito verranno spiegati solo i risultati ottenuti dall'esercitazione.

2 Results

Il modello di machine learning implementato è un classificatore multiclasse con il RandomForest.
Per la configurazione degli iperparametri sono stati selezionati il numero di alberi da costruire e la profondità

massima di ogni albero.
I valori non sono troppo alti per non saturare la memoria heap del container Docker, ma comunque ottengono una buona performance per il tempo impiegato.
In figura 6 si possono notare i risultati ottenuti dai test del modello configurato con i diversi iperparametri.

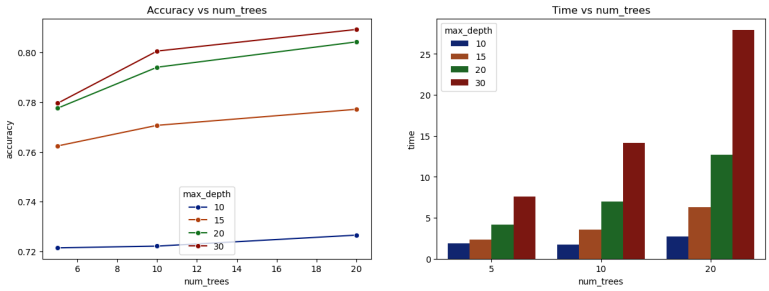

Figure 6: Risultati dell'addestramento del modello

Questa è una semplice implementazione di un classificatore multiclasse con il RandomForest che lavora su 60 mila esempi. Per sfruttare al meglio la tecnologia Spark occorrerebbe utilizzare dataset molto più grandi e magari non in modalità standalone.
La struttura configurata per l'addestramento su Spark rispecchia un'ambiente reale in produzione, dove si utilizzano pipeline per la preparazione dei dati e si addestrano modelli.

3 Conclusions

In questo articolo è stata presentata una breve introduzione alla tecnologia Apache Spark e sono stati spiegati i concetti principali per utilizzarla.
In particolare è stata spiegata la differenza tra Hadoop MapReduce e Spark e come Spark sia più veloce e scalabile rispetto a MapReduce.
Sono state spiegate le principali caratteristiche di Spark come gli RDD, le variabili condivise e le nuove API come Spark SQL e MLlib.
Infine è stata presentata una semplice implementazione di un classificatore multiclasse con il RandomForest utilizzando Spark e PySpark su un ambiente quasi in produzione.

References

[1] N. F. Tech. Apache Spark: Visual Intro. [Online]. Available: https://medium.com/nerd-for-tech/apache-spark-visual-intro-9eb3fd2709f9

[2] A. Muhandisin. MapReduce vs Spark: Choosing the Right Framework for Your Big Data Needs. [Online]. Available: https://arismuhandisin.medium.com/mapreduce-vs-spark-choosing-the-right-framework-for-your-big-data-needs-2b01e3a51dcd

[3] A. Spark. Cluster Overview. [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview.html

[4] A. Vidhya. Spark RDD Low-Level API Basics using PySpark. [Online]. Available: https://medium.com/analytics-vidhya/spark-rdd-low-level-api-basics-using-pyspark-a9a322b58f6

[5] Wikipedia. Apache Spark. [Online]. Available: https://it.wikipedia.org/wiki/Apache_Spark

[6] A. W. Services. What is Apache Spark? [Online]. Available: https://aws.amazon.com/it/what-is/apache-spark/

[7] IBM. Apache Spark. [Online]. Available: https://www.ibm.com/it-it/topics/apache-spark

[8] A. Spark. Apache Spark Documentation. [Online]. Available: https://spark.apache.org/docs/latest/

[9] ——. RDD Programming Guide. [Online]. Available: https://spark.apache.org/docs/latest/rdd-programming-guide.html

[10] ——. SQL Programming Guide. [Online]. Available: https://spark.apache.org/docs/latest/sql-programming-guide.html

[11] ——. MLlib Guide. [Online]. Available: https://spark.apache.org/docs/latest/ml-guide.html

[12] ——. ML Pipelines. [Online]. Available: https://spark.apache.org/docs/latest/ml-pipeline.html