Cos'è hadOOP

Programmazione

Hadoop è un framework open-source progettato per l’elaborazione distribuita di grandi quantità di dati su cluster di macchine commodity.

La sua architettura si basa sul principio del divide et impera: suddivide dataset massivi in blocchi, li distribuisce tra più nodi e li elabora in parallelo.

Questo approccio permette di ottenere scalabilità orizzontale, tolleranza ai guasti e prestazioni elevate senza la necessità di hardware costoso.

Cuore del sistema è l’HDFS (Hadoop Distributed File System), un file system distribuito che gestisce la replica dei dati, garantendo resilienza anche in caso di guasto di uno o più nodi del cluster.

Il modello di calcolo principale di Hadoop è MapReduce, che consente l'elaborazione massiva tramite due fasi distinte: map, in cui i dati vengono filtrati e trasformati, e reduce, in cui vengono aggregati i risultati intermedi.

Il JobTracker e i TaskTracker (o il ResourceManager e NodeManager in YARN) gestiscono la distribuzione dei task, l'allocazione delle risorse e il monitoraggio dell'esecuzione.

Hadoop si integra inoltre con un vasto ecosistema di strumenti come Hive, Pig, Spark e HBase, che estendono il framework con capacità SQL-like, analisi avanzate, stream processing e database NoSQL distribuiti, rendendolo uno dei pilastri fondamentali del Big Data computing.

Elementi principali di Hadoop

- **HDFS (Hadoop Distributed File System)**

File system distribuito progettato per archiviare grandi quantità di dati in modo ridondante. Divide i file in blocchi e li replica su più nodi per garantire affidabilità e tolleranza ai guasti.

- **YARN (Yet Another Resource Negotiator)**

Livello di gestione delle risorse e dei job. Coordina l'esecuzione delle applicazioni distribuendo CPU e memoria tra i nodi del cluster.

Elementi principali di Hadoop

- **MapReduce**

Modello di programmazione basato su due fasi: map (processamento parallelo dei dati) e reduce (aggregazione dei risultati). È il motore di calcolo tradizionale di Hadoop.

- **NameNode**

Nodo master di HDFS che gestisce la struttura del file system, la posizione dei blocchi e le operazioni sui file. È critico per il funzionamento del cluster.

- **HBase**

Database NoSQL distribuito, basato su colonne, progettato per operazioni in tempo quasi reale su dataset molto grandi.

Elementi principali di Hadoop

- **DataNode**

Nodi che memorizzano fisicamente i blocchi dei dati. Si occupano di lettura, scrittura e replica dei blocchi secondo le direttive del NameNode.

- **ResourceManager (YARN)**

Componente centrale che decide come distribuire le risorse tra le varie applicazioni eseguite nel cluster.

- **Pig**

Linguaggio di scripting ad alto livello (Pig Latin) per costruire pipeline di trasformazioni sui dati, particolarmente utile per ETL.

Elementi principali di Hadoop

- **NodeManager (YARN)**

Agenti installati sui singoli nodi che si occupano dell'esecuzione dei task e del reporting delle risorse usate.

- **JobHistory Server**

Server dedicato alla memorizzazione e consultazione dello storico dei job eseguiti, utile per debug e audit.

- **Hive**

Layer SQL-like che consente di interrogare i dati in HDFS usando un linguaggio simile a SQL, trasformando automaticamente le query in job MapReduce o Spark.

Elementi principali di Hadoop

- **Zookeeper**

Servizio di coordinamento distribuito che fornisce configurazione centralizzata, gestione dei lock e sincronizzazione per componenti Hadoop e sistemi correlati.

- **Sqoop**

Strumento per importare ed esportare dati tra database relazionali e HDFS.

- **Flume**

Sistema per l’ingestione massiva di log e flussi di dati verso HDFS.

1. map(KEYIN key, VALUEIN value, Context context) – Metodo del Mapper

È il cuore della fase Map. Viene eseguito per ogni riga/record in input.

Serve a trasformare il dato grezzo in una lista di coppie chiave → valore da aggiungere allo shuffle.

Cosa fa:

- **Legge una riga**
- **La pulisce/trasforma**
- **Emette una o più coppie con context.write()**

È il metodo che definisce “come leggere” il dataset.

2. reduce(KEYIN key, Iterable<VALUEIN> values, Context context) – Metodo del Reducer

Viene eseguito una volta per ogni chiave prodotta dal Map.
Serve ad aggregare, sommare, contare, fare statistiche, ecc.

Cosa fa:

- **Riceve una chiave**
- **Riceve tutti i valori ad essa associati**
- **Produce il risultato finale per quella chiave**

È il metodo dove si definisce “come aggregare” i dati.

3. setup(Context context) – Metodo opzionale di Mapper/Reducer

Viene eseguito una sola volta prima della fase Map o Reduce.

Cosa ci fai:

- **Inizializzazione di variabili**
- **Lettura parametri di configurazione**
- **Apertura file / connessioni**

Serve a evitare di ripetere operazioni costose dentro map() o reduce().

4. cleanup(Context context) – Metodo opzionale di Mapper/Reducer

Viene eseguito una sola volta dopo la fase Map o Reduce.

Cosa ci fai:

- Chiusura risorse
- Scrittura finale di valori aggregati
- Log, metriche, pulizie in generale

È l’opposto di setup(): gestisce tutto ciò che succede alla fine.

Le 4 import più comuni di Hadoop e cosa contengono

Queste sono le librerie Java Hadoop più utilizzate in ogni job MapReduce.

1. import org.apache.hadoop.mapreduce.*;

Contiene tutte le classi core per il MapReduce moderno:

- Mapper
- Reducer
- Job
- Context
- Partitioner, InputFormat, OutputFormat

È la libreria principale per scrivere job MapReduce.

2. import org.apache.hadoop.io.*;

Contiene i tipi di dato serializzabili usati da Hadoop:

- **Text**
- **IntWritable**
- **LongWritable**
- **FloatWritable**
- **NullWritable**
- **ecc.**

Questi rimpiazzano i tipi Java standard perché Hadoop richiede oggetti Writable.

3. import org.apache.hadoop.fs.*;

Contiene le API per lavorare con HDFS:

- **FileSystem**
- **Path**
- **FSDataInputStream, FSDataOutputStream**

Serve per leggere e scrivere file sul file system distribuito.

4. import org.apache.hadoop.conf.*;

Contiene:

- **Configuration (una delle classi più usate in assoluto)**
- **Parametri e configurazioni dei job**
- **Accesso alle variabili di Hadoop**

È usata in ogni Driver per configurare il Job.

Il programma “Hello World” in Hadoop definisce un semplice job MapReduce composto da un Mapper e un Reducer, entrambi basati sui tipi Text di Hadoop.

Il Mapper, ignorando completamente l’input, emette per ogni riga del file una coppia chiave-valore fissa "Hello" → "World", mostrando la struttura minimale di una fase di mappatura.

Il Reducer riceve quindi la chiave "Hello" assieme alla lista dei valori generati dal Mapper (tutti "World"), ma ne produce uno solo come output finale, scrivendo la coppia "Hello World".

Il metodo main() configura il job: imposta le classi Mapper e Reducer, i tipi di output, e definisce i percorsi di input e output su HDFS, per poi avviare l’esecuzione del job tramite waitForCompletion().

Insieme, questi elementi mostrano il flusso essenziale di Hadoop: input distribuito, emissione di coppie dal Mapper, aggregazione delle chiavi comuni nel Reducer e generazione del risultato finale.

```
1.import java.io.IOException;
2.import org.apache.hadoop.conf.Configuration;
3.import org.apache.hadoop.fs.Path;
4.import org.apache.hadoop.io.Text;
5.import org.apache.hadoop.mapreduce.Job;
6.import org.apache.hadoop.mapreduce.Mapper;
7.import org.apache.hadoop.mapreduce.Reducer;
8.import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
10.
11.public class HelloWorld {
12.
13.   public static class HelloMapper
14.         extends Mapper<Object, Text, Text, Text> {
15.
16.      @Override
17.      protected void map(Object key, Text value, Context context)
18.            throws IOException, InterruptedException {
19.         context.write(new Text("Hello"), new Text("World"));
20.      }
21.   }
22.
23.   public static class HelloReducer
24.         extends Reducer<Text, Text, Text, Text> {
25.
26.      @Override
27.      protected void reduce(Text key, Iterable<Text> values, Context context)
28.            throws IOException, InterruptedException {
29.         context.write(key, new Text("World"));
30.      }
31.   }
32.
33.   public static void main(String[] args) throws Exception {
34.      Configuration conf = new Configuration();
35.      Job job = Job.getInstance(conf, "hello world");
36.
37.      job.setJarByClass(HelloWorld.class);
38.      job.setMapperClass(HelloMapper.class);
39.      job.setReducerClass(HelloReducer.class);
40.
41.      job.setOutputKeyClass(Text.class);
42.      job.setOutputValueClass(Text.class);
43.
44.      FileInputFormat.addInputPath(job, new Path(args[0]));
45.      FileOutputFormat.setOutputPath(job, new Path(args[1]));
46.
47.      System.exit(job.waitForCompletion(true) ? 0 : 1);
48.   }
49.}
```


Buon Davante a tutti