# Introduzione ad Apache Spark

Andrea Nasato

28 gennaio 2017

# Outline

# Topic

# Apache Spark

Apache Spark is a fast and general engine for large-scale data processing.

# Breve storia di Spark

| | |
|---|---|
| 2009 | progetto di ricerca in UC Berkley (tesi dottorato Matey Zaharia) |
| 2010 | diventa open source |
| 2013 | viene donato all'Apache Software Foundation |
| 2014 | diventa un Apache Top-Level Project<br>vince la sortbenchmark competition (GraySort) |
| 2015 | versioni 1.4.0 -> 1.5.2<br>600 contributors |
| 2016 | versioni 1.6.0 -> 2.1.0<br>1000 contributors<br>vince la sortbenchmark competition (CloudSort) |

# sortbenchmark 2014

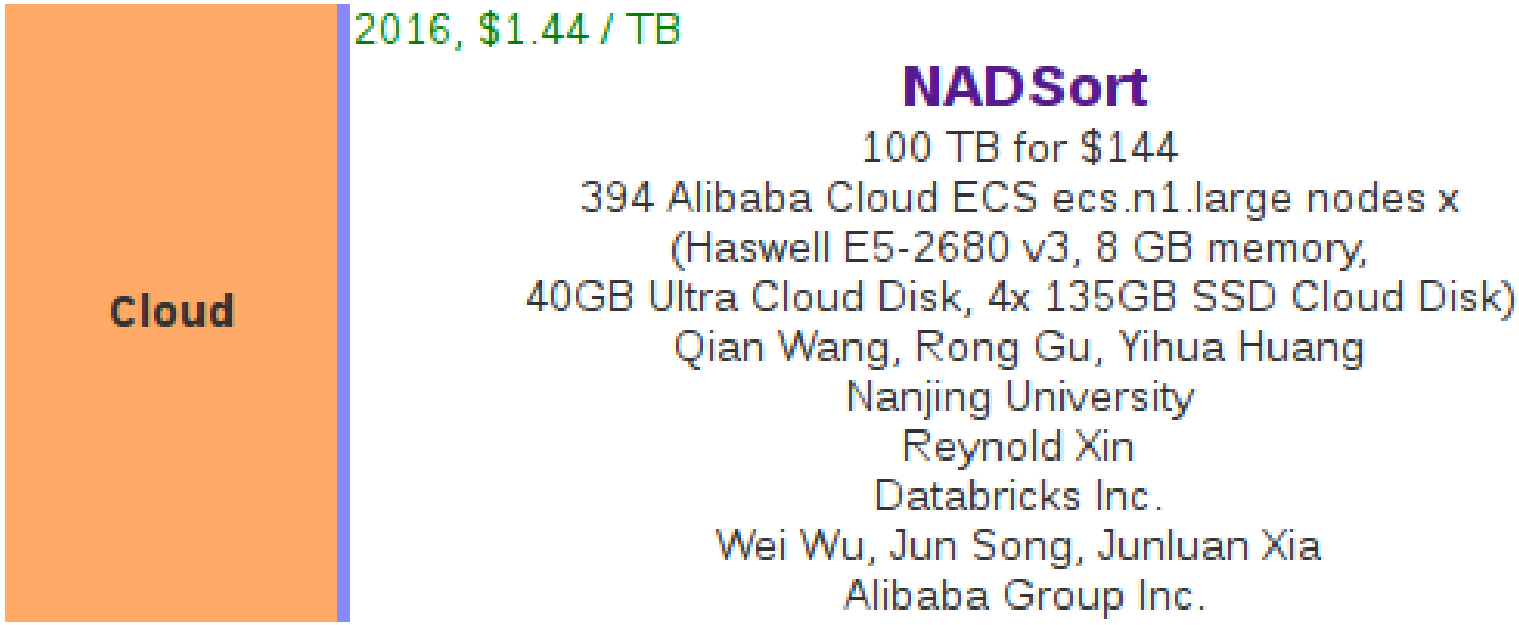
Sort rate (TBs / minute) achieved while sorting a very large amount of data (currently 100 TB minimum).

| | Hadoop MR Record | Spark Record |
|---|---|---|
| Data Size | 102.5 TB | 100 TB |
| Elapsed Time | 72 mins | 23 mins |
| # Nodes | 2100 | 206 |
| # Cores | 50400 physical | 6592 virtualized |
| Cluster disk throughput | 3150 GB/s(est.) | 618 GB/s |
| Network | dedicated data center 10Gbps | virtualized (EC2) |
| Sort rate | 1.42 TB/min | 4.27 TB/min |
| Sort rate/node | 0.67 GB/min | 20.7 GB/min |

(fonte
`https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html`)

# sortbenchmark 2016

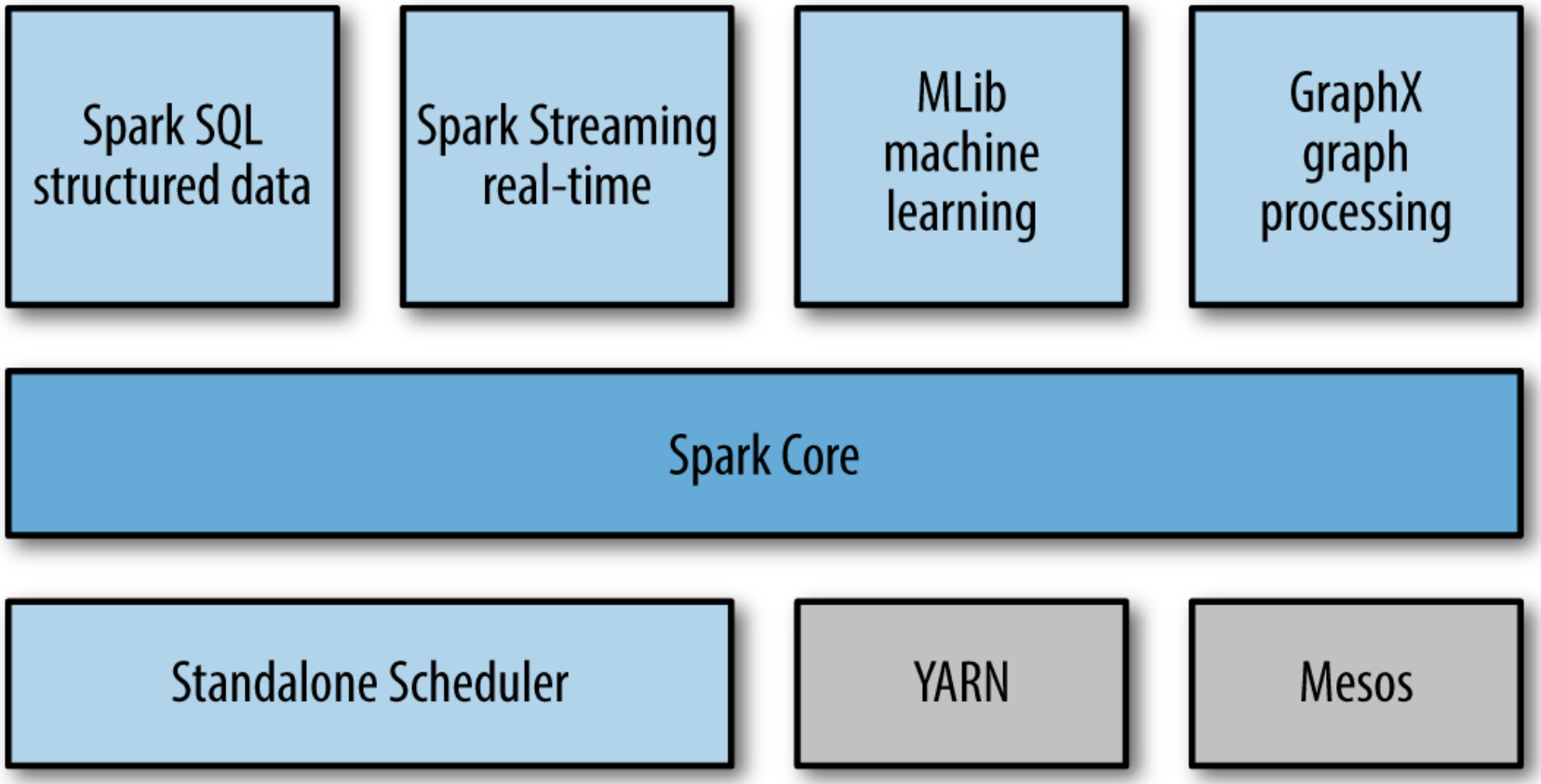
Minimum cost for sorting a very large amount of data on a public cloud - currently 100 TB




(**fonte**: `http://sortbenchmark.org/`)

# Topic

# Schema dell'architettura



# Driver

Lancia operazioni in parallelo verso un cluster

- REPL (Scala/Python)
- spark-submit (CLI)

# SparkContext

```
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;

SparkConf conf = new SparkConf()
    .setMaster("local") // il cluster (local == no cluster)
    .setAppName("sparktest"); // id per cluster manager ui
JavaSparkContext sc = new JavaSparkContext(conf);
```

# RDD

Resilient Distributed Dataset

Collezione di oggetti:

- immutabile
- distribuita

Si crea:

- caricando un dataset esterno
- parallelizzando collezione esistente

# RDD Esempi

## Dataset esterno

```
JavaSparkContext sc = new JavaSparkContext(conf);
JavaRDD<String> rdd = sc.textFile("/usr/share/dict/words");
```

## Collezione esistente

```
JavaRDD<String> rdd2 = sc.parallelize(
    Arrays.asList("pippo", "pluto", "paperino"));
```

# Topic

# Trasformazioni

- Creano un nuovo RDD a partire da un RDD origine
- Sono eseguite in maniera lazy cioè quando sono usate in una azione

## Filtro

```
JavaRDD<String> filteredRdd = rdd.filter(
    s -> s.startsWith("progr") );
```

## Unione

```
JavaRDD<String> filteredRdd2 = rdd.filter(
    s -> s.startsWith("jug") );
JavaRDD<String> unionRdd = filteredRdd1.union(filteredRdd2);
```

# Map

- La funzione viene eseguita su ogni elemento dell'RDD, in output ho un nuovo RDD con un elemento per ogni elemento dell'RDD origine.

## Map

```
JavaRDD<String> strTest = sc.parallelize(
   Arrays.asList("Programming in Spark","is really","fun"));
JavaRDD<String[]> m = strTest.map(s -> s.split(" "));
//[ ["Programming", "in", "Spark"],
//  ["is", "really"],
//  ["fun"] ]
```

# FlatMap

La funzione viene eseguita su ogni elemento dell'RDD, ma invece di restituire un singolo valore resituisce un iterator con i valori (quindi 0 o più valori). Inoltre l'output è *appiattito* di un livello

## FlatMap

```
JavaRDD<String> strTest = sc.parallelize(
    Arrays.asList("Programming in Spark","is really","fun"));
JavaRDD<String> fm = strTest.flatMap(
    s -> Arrays.asList(s.split(" ")).iterator());
//[ "Programming", "in", "Spark",
//  "is", "really",
//  "fun" ]
```

# Azioni

Restituiscono il valore al driver o scrivono verso il sistema di storage

## Count

```
JavaRDD<String> rdd = sc.textFile("/usr/share/dict/words");
JavaRDD<String> filteredRdd1 = rdd.filter(
    s -> s.startsWith("progr") );
long cnt = filteredRdd1.count();
```

Altre azioni comuni:

- take(int)
- collect()
- ...

# Esempio java 7 vs java 8

L'introduzione delle funzioni in java ha snellito la sintassi.

## Java 7

```
JavaRDD<String> java7syntax = rdd.filter(
    new Function<String, Boolean>(){
public Boolean call(String x) {
    return x.startsWith("progr");
}
    }
);
```

## Java 8

```
JavaRDD<String> java8syntax = rdd.filter(
    s -> s.startsWith("progr"));
```

# History server

Permette di analizzare i job eseguiti sia in locale che nel cluster
`http://localhost:18080`

# Topic

# DataFrame API

- API pensata per costruire query plan relazionali
- schema che descrive i dati
- dati memorizzati in formato binario off-heap
- spark esegue le operazioni sui dati in formato binario

## Esempio

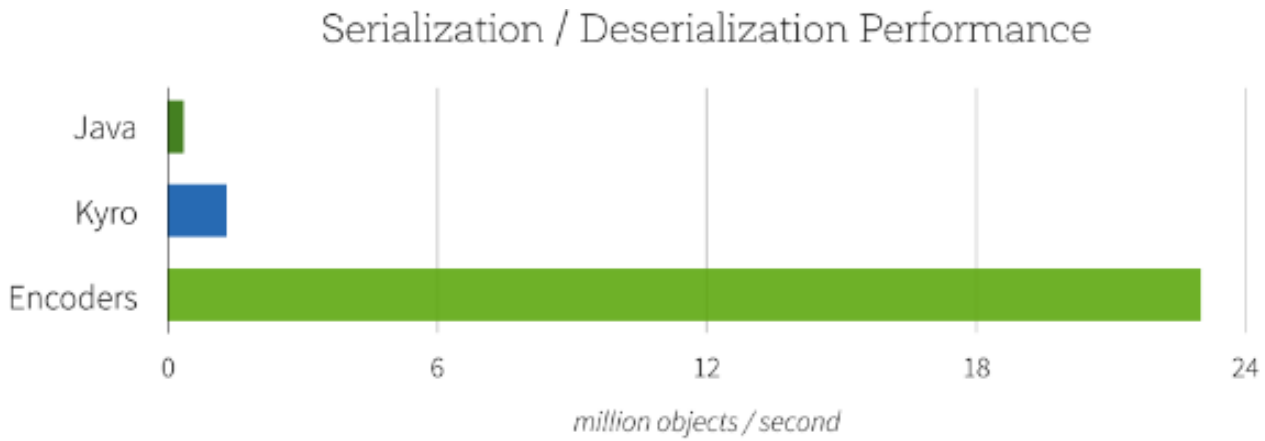
```
String path = "/path/to/a/json/file";
SparkConf conf = new SparkConf()
    .setAppName("DataFrameExample");
JavaSparkContext sc = new JavaSparkContext(conf);
SQLContext sqlCtx = new SQLContext(sc);

DataFrame df = sqlCtx.load(path);
df.select("age > 18");
```

# RDD vs DataFrame

- RDD
  - API object oriented
  - performance scarse (java serialization)
- DataFrame
  - API data oriented
  - performance molto migliori (spark serialization)

# Dataset

- Meccanismo di serializzazione di DataFrame
- API object oriented




- Serializzazione/deserializzazione più veloce
- I dati serializzati hanno dimensioni inferiori

(**fonte**: `https://databricks.com/blog/2016/01/04/introducing-apache-spark-datasets.html`)

# Sorgente dati

## LastFM dataset

*This dataset contains <user, artist, plays> tuples (for ~360,000 users) collected from Last.fm API, using the user.getTopArtists() method.*

## Statistiche

File usersha1-artmbid-artname-plays.tsv:

- Total Lines: 17,559,530
- Unique Users: 359,347

# Struttura dati

- File usersha1-artmbid-artname-plays.tsv:

```
user-mboxsha1 \t musicbrainz-artist-id \t artist-name
\t plays
```

*esempio*

```
000063d3fe1cf2ba248b9e3c3f0334845a27a6be \t
a3cb23fc-acd3-4ce0-8f36-1e5aa6a18432 \t u2 \t 31
```

- File usersha1-profile.tsv:

```
user-mboxsha1 \t gender (m|f|empty) \t age (int|empty)
\t country (str|empty) \t signup (date|empty)
```

*esempio*

```
000063d3fe1cf2ba248b9e3c3f0334845a27a6be \t m \t
19 \t Mexico \t Apr 28, 2008
```

# Caricamento dei file nel contesto Spark

```
String path =
"lastfm-dataset-360K/usersha1-artmbid-artname-plays.tsv";

Dataset<Row> dsLastFmPlays = spark //SparkSession
    .read()
    .format("com.databricks.spark.csv")
    .option("delimiter", "\t")
    .load(path);

dsLastFmPlays.printSchema();
```

# Output

La struttura del file viene determinata a runtime con dei default

```
...
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
...
```

# Impostazione dello schema del file

```
StructField[] fields = {
    DataTypes.createStructField("userMboxSHA1",
DataTypes.StringType, true)
    // same for musicbrainzArtistId
    // same for artistName
    // same for play
};

StructType lastFMType = DataTypes.createStructType(fields);

Dataset<Row> dsLastFmPlays = spark //SparkSession
    .read()
    .format("com.databricks.spark.csv")
    .option("delimiter", "\t")
    .schema(lastFMType)  //passo lo schema
    .load(path);
```

# Impostazione dell'encoder

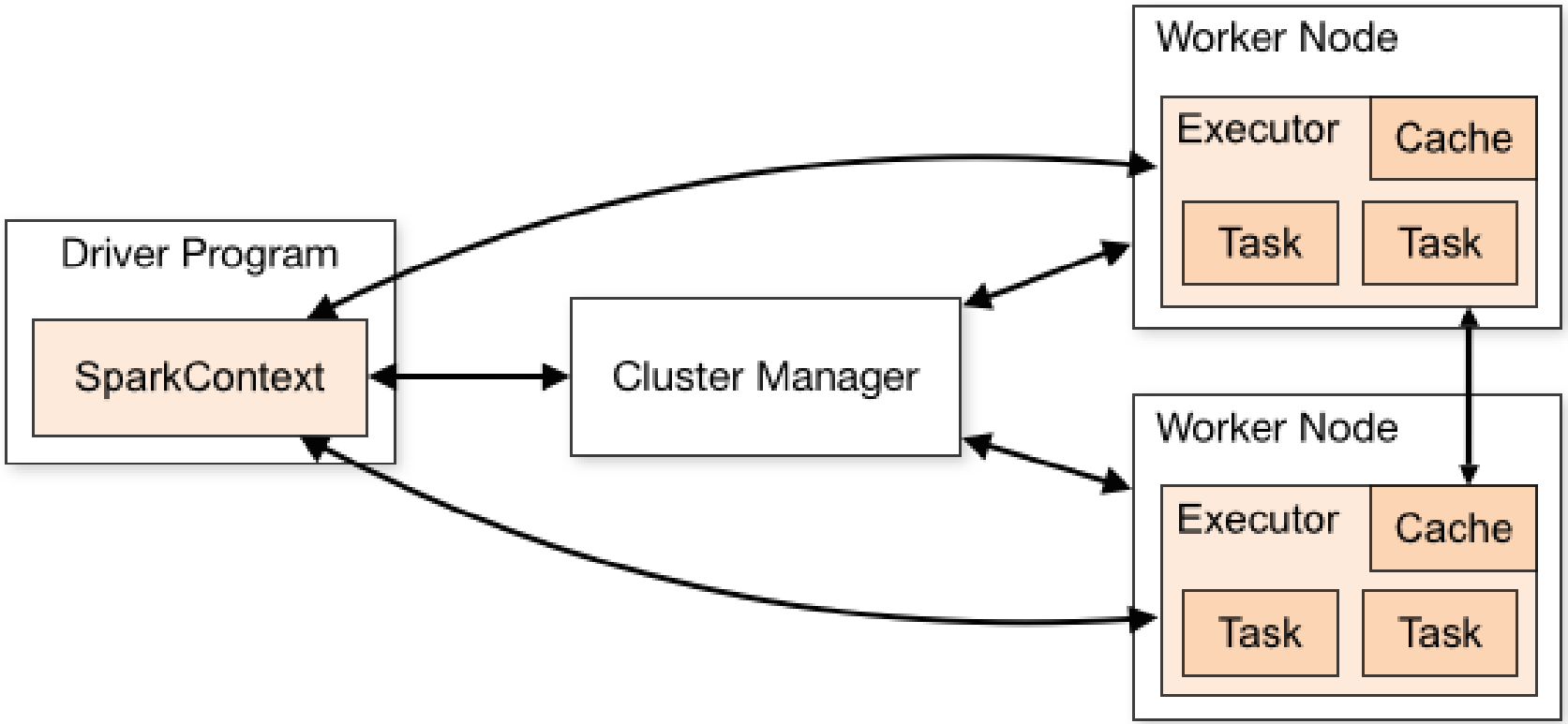
```
Encoder<LastFMPlays> lastFMEncoder =
    Encoders.bean(LastFMPlays.class);

Dataset<LastFMPlays> dsLastFmPlays = spark
    .read()
    .format("com.databricks.spark.csv")
    .option("delimiter", "\t")
    .schema(lastFMType)
    .load(path)
    .as(lastFMEncoder)
    ;
```

# Esempio SQL

```
// la join mi dn nuovo dataset
Dataset<Row> joinedDS = dsLastFmPlays
    .join(dsLastFMArtists, "userMboxSHA1");

// creazione di una vista sul dataset in join
joinedDS.createOrReplaceTempView("lastfm");

Dataset<Row> rowsDF = spark
    .sql("select * from lastfm where
  plays > 100 order by artistName");

rowsDF.show();
```

# Topic

# Schema




- ogni sparkcontext ha i suoi executors (isolamento)
- non è necessario specificare il cluster manager
- il driver dovrebbe stare nella stessa rete dei workers

# Cluster manager

- Standalone
- Apache Mesos
- Hadoop YARN

# Esempio di cluster standalone

Ogni nodo (sia il master che gli slaves) deve avere una installazione di Spark

## Avvio di master

```
$ cd $SPARK_HOME
$ sbin/start-master.sh
```

## Avvio di un slave (worker)

```
$ cd $SPARK_HOME
$ sbin/start-slave -c 2 -m 2G spark://${IP}:7077
```

## Opzioni di avvio degli slaves (workers)

- -c CORES : quanti *cpu cores* può usare il worker
- -m MEM : quanta *memoria* può usare il worker
- si fornisce l'url del master (nella forma *spark* : //)

# Topic

# Streaming, MLlib, GraphX

- https://spark.apache.org/streaming/
- https://spark.apache.org/mllib/
- https://spark.apache.org/graphx/

# Topic

# Contatti

| | |
|---|---|
| mail | andrea.nasato@gmail.com |
| web | `http://www.gtrev.it` |
| github | `https://github.com/gtrev` |
| linkedin | `https://www.linkedin.com/in/gtrev` |