SPARK

If you don’t drive your business,
you will be driven out of business

- Introduction to Spark
- History of spark
- Why Spark ?
- Spark vs Hadoop
- Example
- Hardware dependences
- Prerequisites

Creative Brain

Big Data

Based on context,

- Data that exceeds the processing capacity of traditional DBs
- ‘Big Data’ is similar to ‘small data’, but bigger in size.
- Big data Measurement terms:

1000 Gigabytes (GB)	=	1 Terabyte (TB)
1000 Terabytes	=	1 Petabyte (PB)
1000 Petabytes	=	1 Exabyte (EB)
1000 Exabyte	=	1 Zettabyte (ZB)
1000 Zettabytes	=	1 Yottabyte (YB)

First distributed systems

- Proprietary
- Custom Hardware and software
- Centralized data
- Hardware based fault recovery
- Ex: Teradata, Netezza etc

Second generation

- Open source
- Commodity hardware
- Distributed data
- Software based fault recovery
- Ex : Hadoop, NoSQL

Third generation distributed systems

- Handle both batch processing and real time
- Exploit RAM as much as disk
- Multiple core aware
- They use
  - ✓ HDFS for storage
  - ✓ Apache Mesos / YARN for distribution
  - ✓ Plays well with Hadoop

SPARK

- Apache spark is an open source, parallel data processing framework, which uses in memory cluster computing which can access faster than disk access.

- Spark is fast , map-Reduce like engine.

- Spark improves efficiency when compared to other processing algorithms like MapReduce
  - In-memory computing primitives
  - General computation graphs

- Created by AMPLab now Databricks

History

- Started as a research project at the UC Berkeley AMPLab in 2009, and was open sourced in early 2010 under a BSD license.

- After being released, Spark grew a developer community on GitHub and entered Apache in 2013 as its permanent home.

- Now Apache Spark has become a top level Apache project from Feb-2014.

- At present major contribution to spark is from DataBricks.

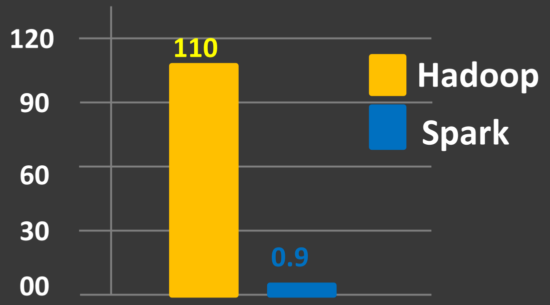
- Codebase size
  - Spark : 20,000 LOC
  - Hadoop 1.0 : 90,000 LOC

Why spark?

When you are choosing a framework, you should consider many factors

1. Speed

- Run programs up to 100x faster than
- Hadoop MapReduce in memory

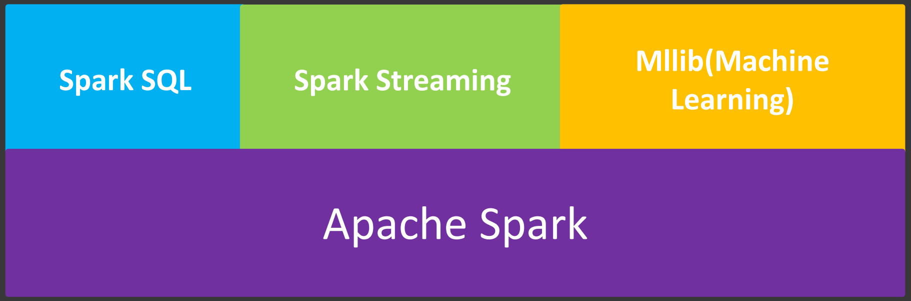


2. Ease of Use

- It provides support for many languages like
  - Java,
  - Scala,
  - Python, R

3. Generality

- All things in place - Spark powers a stack of libraries including SQL and Data frames , Mlib for machine learning, and Spark Streaming.
- You can combine these libraries seamlessly in the same application.



Spark SQL

Spark Streaming

Mllib(Machine Learning)

Apache Spark

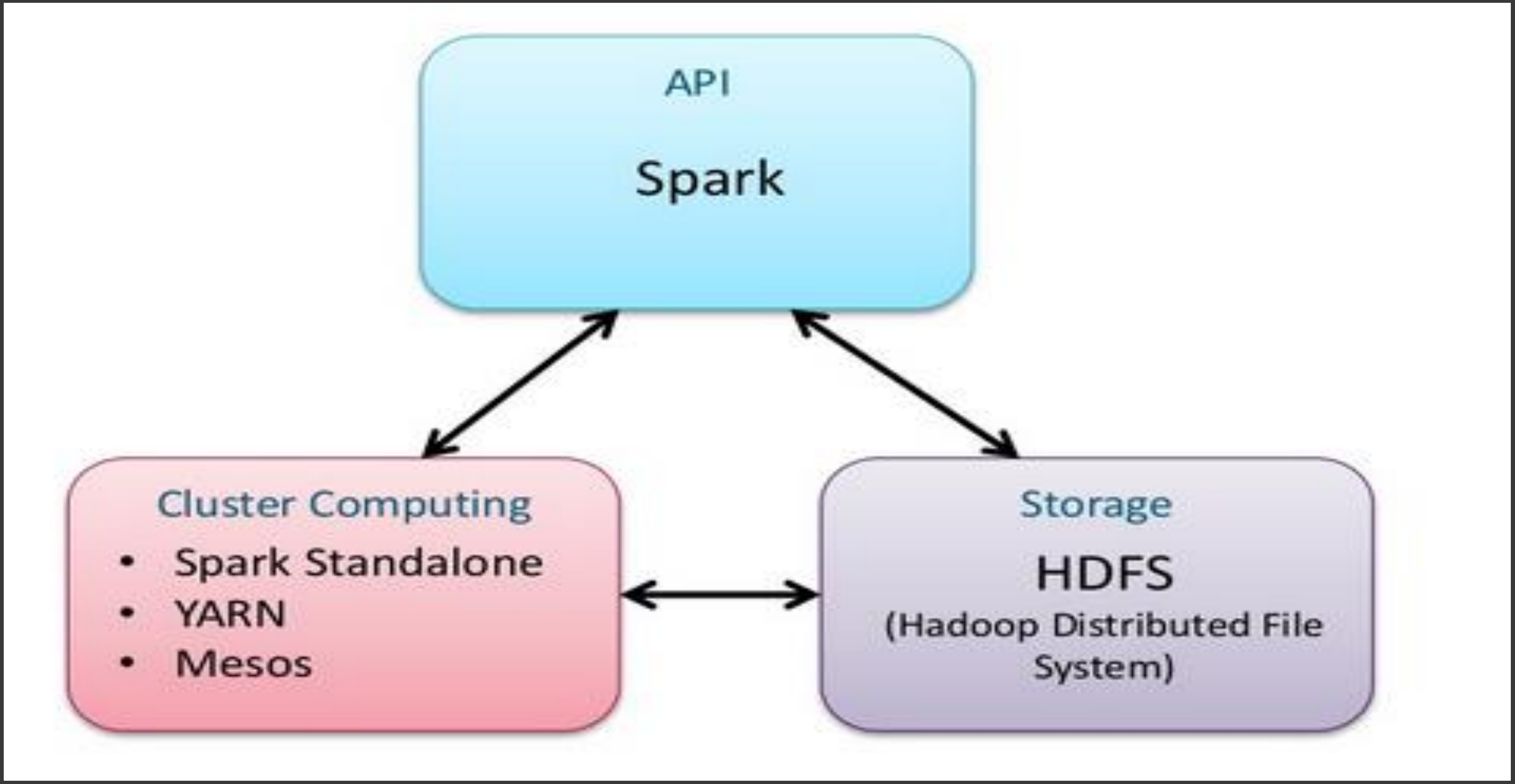
4. Runs Everywhere –

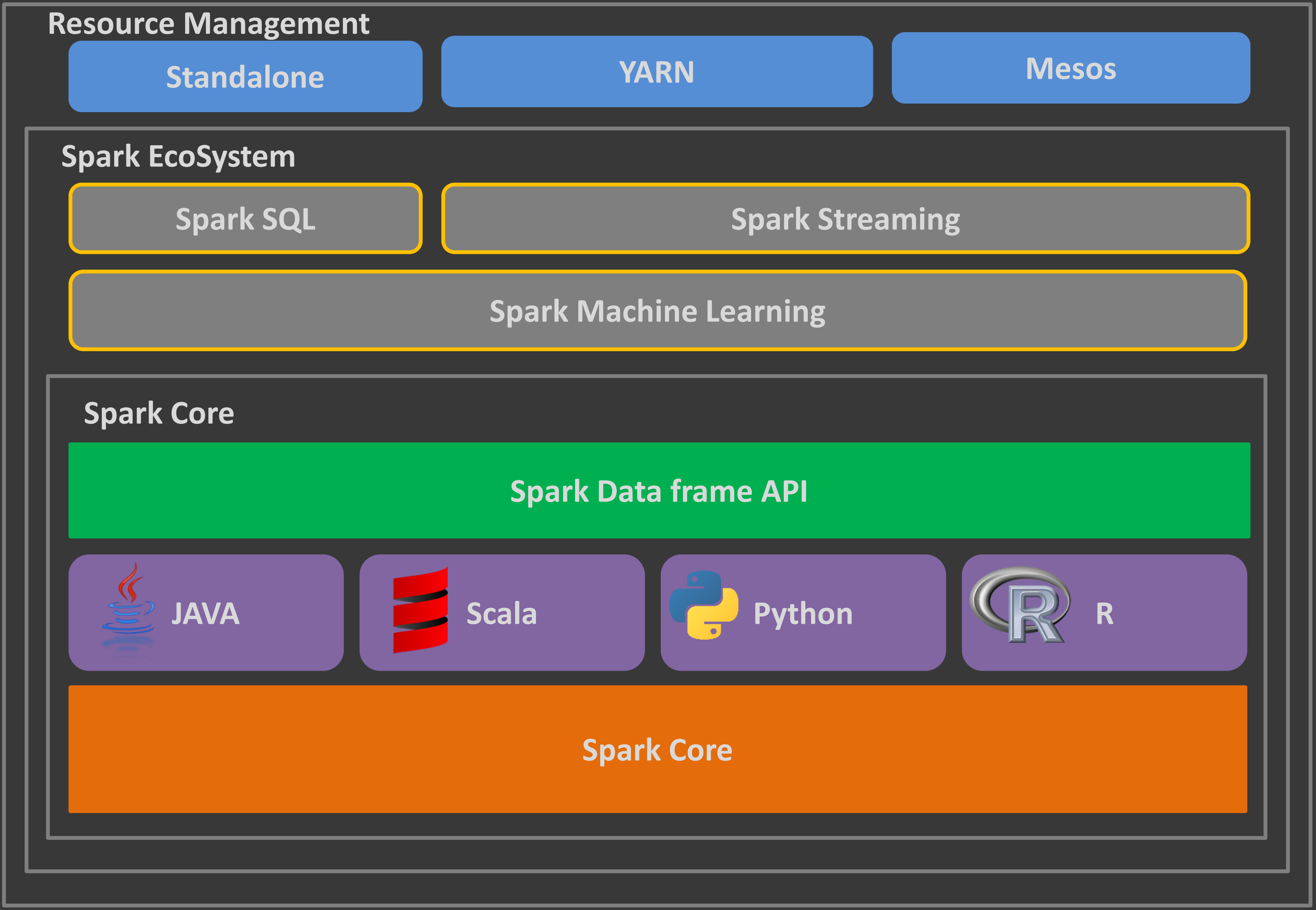
- Spark runs on Hadoop (YARN),
- Mesos,
- Standalone or in the cloud.
- It can access diverse data sources including HDFS, Cassandra, HBase, and S3.

5. Data serialization

- It will Tune data serialization method, Currently It provides two serialization libraries:
  - Java serialization
  - Kryo serialization

Spark Framework





Resource Management

Standalone

YARN

Mesos

Spark EcoSystem

Spark SQL

Spark Streaming

Spark Machine Learning

Spark Core

Spark Data frame API

JAVA

Scala

Python

R

Spark Core

Using SPARK with Hadoop

- Use Hadoop as storage (for input and/or output) for SPARK
- Spark program as a YARN program (by using the resource manager of YARN) (Yarn childs will be spark processes)

Spark Core Engine:

The core engine for entire spark framework; provides utilities and architecture for other components

Spark SQL:

- used for structured data
- can expose many datasets as tables
- can be integrated with Hive

Spark Streaming

- Enables live streaming data processing
- A good alternative to Storm

MLLib

- Machine learning library being built on top of Spark
- Provision for support to many machine learning algorithms with speeds upto 100 times faster than MR
- Mahout is also being migrated to MLLib

SparkR

- Package for R language to enable R-users to leverage Spark power from R shell

- Industries are using Hadoop extensively to analyse their data sets.

- The reason is that Hadoop framework is based on a simple programming model (MapReduce) and it enables a computing solution that is scalable, flexible, fault-tolerant and cost effective.

- Here, the main concern is to maintain speed in processing large datasets in terms of waiting time between queries and waiting time to run the program.

- This is the main cause for evolution of spark.

- Spark is for near real-time processing or faster batch processing which eliminates the drawbacks of current MapReduce engine.

- Spark is not a replacement for MapReduce, spark will not best choice when you want to run a batch process for a very large dataset.

Spark VS Hadoop

- **Hadoop** is parallel data processing framework that has traditionally been used to run map/reduce jobs. These are long running jobs that take minutes or **hours to complete**.

- **Spark** has designed to run on top of Hadoop and it is an alternative to the traditional batch map/reduce model that can be used for real-time stream data processing and fast interactive queries that finish **within seconds**.

- We should look at Hadoop as a general purpose Framework that supports multiple models and **We should look at Spark as an alternative to Hadoop MapReduce rather than a replacement to Hadoop**.

- Spark stores data in-memory whereas Hadoop stores data on disk. **Hadoop uses replication to achieve fault tolerance whereas Spark uses different data storage model, resilient distributed datasets (RDD)**, uses a clever way of guaranteeing fault tolerance that minimizes network I/O

In-memory aka Speed

- In Spark, you can cache hdfs data in main memory of worker nodes
- Spark analysis can be executed directly on in memory data
- Shuffling also can be done from in memory
- Fault tolerant

Integration with Hadoop

- No separate storage layer
- Integrates well with HDFS
- Can run on Hadoop 1.0 and Hadoop 2.0 YARN
- Excellent integration with ecosystem projects like Apache Hive, HBase etc

Eco System

Hadoop	Spark
Hive	Spark SQL
Apache Mahout	MLLib
Impala	Spark SQL
Apache Giraph	GraphX
Apache Storm	Spark Streaming

Compare Spark with MR

MR Code,

- WordCountMapper.java
- WordCountReducer.java
- WordCount.java

Compile and Build the Jar and then copy the jar onto edge node and test it on HDFS file.

Spark Code,

```
sc.textFile(“/user/cloudera/README.md") .flatMap(line => line.split("
")) .map(word => (word, 1)).reduceByKey(_ + _).collect()
```

Compare Spark with MR

MR Output,

```
16/07/28 07:57:23 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
……..
16/07/28 07:57:44 INFO mapreduce.Job:  map 0% reduce 0%
16/07/28 07:57:52 INFO mapreduce.Job:  map 100% reduce 0%
16/07/28 07:58:15 INFO mapreduce.Job:  map 100% reduce 100%
16/07/28 07:58:15 INFO mapreduce.Job: Job job_1467173113214_0001 completed successfully
```

52 seconds Execution Time

Spark Output,

```
16/07/28 08:05:33 INFO scheduler.DAGScheduler: Job 0 finished: collect at <console>:28, took 2.163420 s
res0: Array[(String, Int)] = Array((package,1), (For,2), (Programs,1), (processing.,1), (Because,1), (The,1), (cluster.,1), (its,1), ([run,1), (APIs,1), (have,1), (Try,1), (computation,1), (through,1), (several,1), (This,2), ("yarn-cluster",1), (graph,1), (Hive,2), (storage,1), (["Specifying,1), (To,2), (page](http://spark.apache.org/documentation.html),1), (Once,1), (application,1), (prefer,1), (SparkPi,2), (engine,1), (version,1), (file,1), (documentation,,1), (processing,,1), (the,21), (are,1), (systems.,1), (params,1), (not,1), (different,1), (refer,2), (Interactive,2), (R,,1), (given.,1), (if,4), (build,3), (when,1), (be,2), (Tests,1), (Apache,1), (./bin/run-example,2), (programs,,1), (including,3), (Spark.,1), (package.,1), (1000).count(),1), (Versions,1), (HDFS,1), (Data.,1),
```

Compare Spark with MR

Spark	MR
Spark Code is Concise	MR Code is high in code base size
Spark has supports shells execution	No Shell support is present
Easy to test in Spark Applications in Shell itself	Need to develop Jar every time and submit on the cluster
Low Latency results	Very High Latency in results
Development Efforts will be less because of rich set of APIs available	Development Efforts will be more than double the efforts of Equivalent Spark Code Development
Needs More RAM memory	Even Less RAM memory will be sufficient

Compare SparkSql with Hive

Lets compare job execution time between spark and MR

Input data

```
-9999,49775,1,82365242,1,null,null,null,0,0,690
48537129,49775,1,72196636,1,null,null,null,0,0,690
.
.
.
-9999,49775,1,82365241,1,null,null,null,0,0,690
```

Now Create a hive table and load data into that as shown below,

```
CREATE EXTERNAL TABLE survey_detials(trnd_prdc_id int ,store_id int,csl_id int,product_id
int,sales_units tinyint,dsp_k boolean,feature_indic boolean, display_indic
boolean,price_mult tinyint, tpr_indic tinyint,period_id int) row format delimited fields
terminated by ',' stored as textfile location '/poc/survey_detials';
```

```
LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/input_data.txt' OVERWRITE INTO
TABLE survey_detials;
```

Execute Below Query in Hive

```
select product_id, sum(sales_units) from survey_detials where period_id=690 and
store_id=49775 group by product_id ;
```

Executing same query in spark SQL

```
sqlContext.sql("select product_id, sum(sales_units) from survey_detials where
period_id=690 and store_id=49775 group by product_id").collect()
```

Executing same query in native spark

```
case class  survey_detials_pojo(trnd_prdc_id: String, store_id: String, csl_id: String,product_id: String,sales_units:
String,dsp_k: String,feature_indic: String,display_indic:String,price_mult: String,tpr_indic: String)
val period_id = 690
val retailBeanList = sc.textFile("hdfs://quickstart.cloudera/poc/survey_detials/data.txt").map(_.split(",")).map(p
=> survey_detials_pojo(p(0).trim, p(1).trim, p(2).trim, p(3).trim,  p(4).trim, p(5).trim, p(6).trim, p(7).trim, p(8).trim,
p(9).trim))
val retailFinalRDD = retailBeanList.filter(survey_detials_pojo => survey_detials_pojo.store_id=="49775")
val groupedResult = retailFinalRDD.groupBy(_.product_id)
for ((k,v) <- groupedResult.collect()) {
var sales_units_count = 0.0
for ( survey_detials_pojo <- v.toArray) {
sales_units_count += survey_detials_pojo.sales_units.toDouble
}
println(k+"---------"+sales_units_count)
}
```

Results From Above Query

Framework	Time Taken
Hive	74.598 secs
Spark SQL	7.6 secs
Native Spark	1.0 secs

Hardware dependencies

- In general, Spark can run well with anywhere from 8 GB to hundreds of gigabytes of memory per machine.
- In all cases, it is recommend allocating only at most 75% of the memory for Spark; leave the rest for the operating system and buffer cache.
- Any Linux OS distribution.

Prerequisites

- Knowledge on any programing language.
- Familiarity with Big Data issues, tools and concepts.

Any Questions ?

THANK YOU