Inspire…Educate…Transform.

Applying ML to Big Data using Hadoop and Spark Ecosystem

Spark and towards Spark ML

Dr. Manoj Duse

Aug 5, 2018

Agenda

- Spark
- RDDs, Partitions
- Actions, Transformers
- Managing partitions
- Spark ML Overview

Another Parallelization platform: SPARK

- SPARK is not replacement for HADOOP
- SPARK on YARN
- SPARK and MR can coexist
- SPARK for real-time, MR for batch

What is Spark?

Fast and **Expressive** Cluster Computing System
Compatible with Apache Hadoop

Up to **10×** faster on disk,
100× in memory

2-5× less code

Efficient

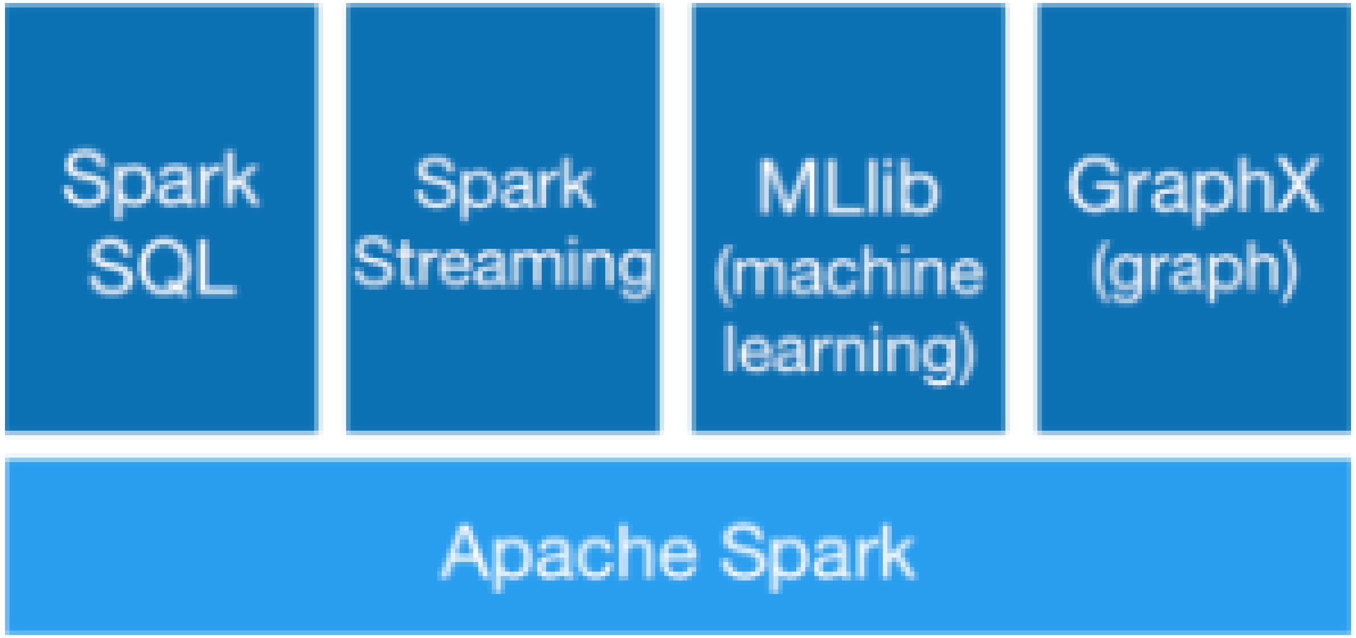
- General execution graphs
- In-memory storage

Usable

- Rich APIs in Java, Scala, Python
- Interactive shell

From McDonough Spark tutorial from Spark Summit 2013

SPARK ecosystem



Spark Core:

Responsible for:

- ✓ Memory Management and fault recovery

- ✓ Supports/implements key concepts of RDDs and Actions

- ✓ Scheduling, Monitoring, Distributing jobs on cluster [via YARN]

Lightning-Fast Cluster Computing

Key Concepts

Write programs in terms of **transformations** on **distributed datasets**

Resilient Distributed Datasets

- Collections of objects spread across a cluster, stored in RAM or on Disk
- Built through parallel transformations
- Automatically rebuilt on failure

Operations

- Transformations (e.g. map, filter, groupBy)
- Actions (e.g. count, collect, save)

Spark Terminology

Driver program	The process running the main() function of the application and creating the SparkContext

Executor	A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.

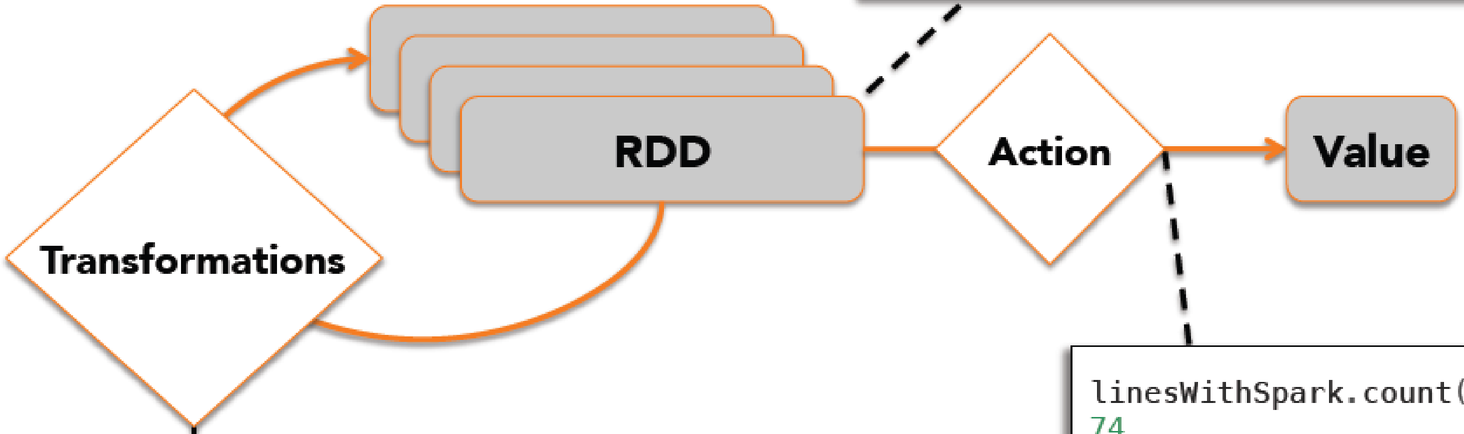
SparkContext works with RM

Programming languages supported by Spark include:

- Java
- Python [PySpark]
- Scala
- SQL
- R [SparkR]

Working With RDDs

```
textFile = sc.textFile("SomeFile.txt")
```




```
linesWithSpark.count()
74

linesWithSpark.first()
# Apache Spark
```

```
linesWithSpark = textFile.filter(lambda line: "Spark" in line)
```

RDDs are immutable

Transformation on one RDD results into a new RDD

Spark Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

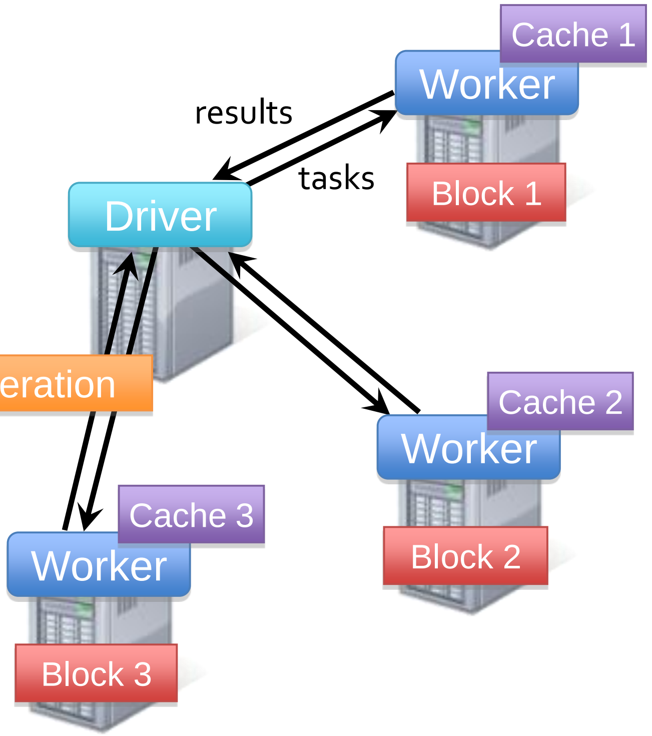
```
lines = sc.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("database")).count
cachedMsgs.filter(_.contains("memory")).count
. . .
```

Base RDD

Transformed RDD

Cached RDD

Parallel operation



You should cache RDDs you work with when you want to execute two or more actions on it for a better performance

Lazy Initialization

- Populating of blocks into memory deferred until action is invoked.
- RDD created but with no data

- Simply put, an action triggers actual evaluation of the RDD .
- Only actions can materialize the entire processing pipeline with real data.

DAG, Lineage Graph [resilient]

Examples of Transformations

- map
- filter
- flatMap
- groupByKey
- sortByKey

Examples of Actions

- Count
- Top(k)

Broadcast variable and Accumulator

- Broadcast variable is a read-only variable
- made available from the driver program that runs the SparkContext object to the nodes that will execute the computation.
- useful in applications that need to make the same [typically reference data] available to the worker nodes in an efficient manner, such as machine learning algorithms.
- one time thing : distributed to the workers only once

- An accumulator is also a variable that is broadcasted to the worker nodes.
- The key difference between a broadcast variable and an accumulator is that while the broadcast variable is read-only, the accumulator can be added to.

Printing contents of a RDD

- myRDD.collect().foreach(println)

- Very useful for debugging

Parallelizing Data

How many partitions my RDD is split into?

```
myRDD.partitions.size
```

How to enforce "degree of parallelism"

```
myRDD = sc.parallelize(1 to 500, 5)
```

```
myRDD.partitions.size
res27: Int = 5
```

repartition vs coalesce

repartition : will shuffle the original partitions and repartition them

coalesce : will just combine original partitions to the new number of partitions.

Shuffling could be very costly,
If all you want is to reduce the no of partitions: use coalesce

RDD → DataFrames → DataSets

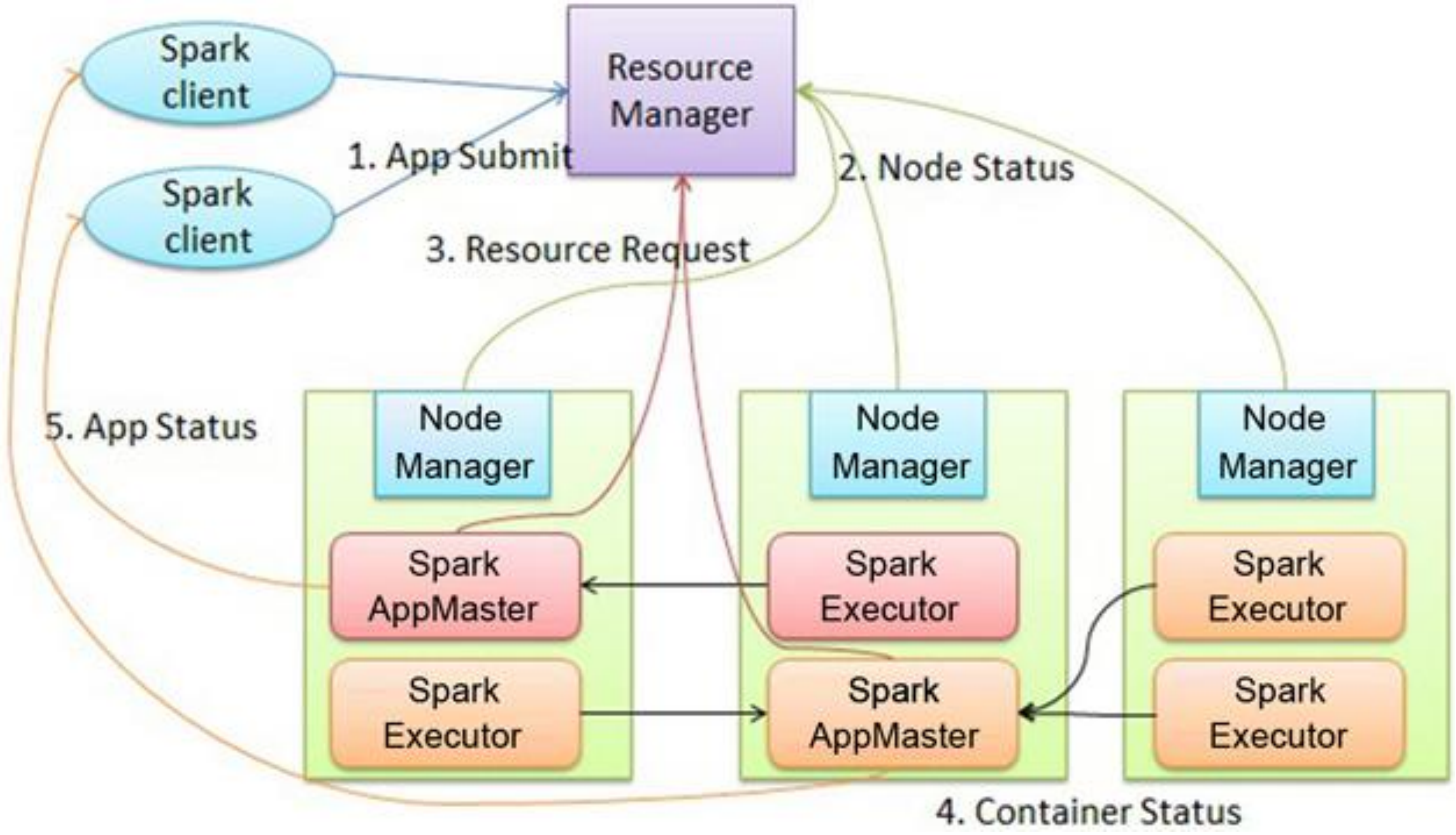
- **Spark Dataframe APIs –**
- Unlike an RDD, data organized into named columns.
- For example a table in a relational database.
- Allows developers to impose a structure onto a RDD

- **Spark Dataset APIs –**
- Datasets in Apache Spark are an extension of DataFrame API which provides type-safe [compile time], object-oriented programming interface.

- One can seamlessly move between DataFrame or Dataset and RDDs by simple API method calls like .rdd or .toDF
- DataFrames and Datasets are built on top of RDDs.

- **RDD –** The RDD APIs have been on Spark since the 1.0 release.
- **DataFrames –** Spark introduced DataFrames in Spark 1.3 release.
- **DataSet –** Spark introduced Dataset in Spark 1.6 release.

Relating back to YARN



Ref to be added

SPARK vs MR

- Ease of use
- Developer productivity
- Speed
- Ecosystem: Spark R, Spark MLLib, PySpark, SparkSQL
- Lot of apache projects also moving to support/leverage Spark

Spark ML

At a high level, it provides tools such as:

•ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering

•Featurization: feature extraction, transformation, dimensionality reduction, and selection

•Pipelines: tools for constructing, evaluating, and tuning ML Pipelines

•Persistence: saving and load algorithms, models, and Pipelines

•Utilities: linear algebra, statistics, data handling, etc.

ML Lib vs ML

ML spark.ml

- New
- DataFrames
- Pipelines

ML Lib spark.mllib

- Old
- RDDS
- But more features but ML catching up
- In maintenance mode; no new functionality will be added

What is the future direction…

- After reaching feature parity (roughly estimated for Spark 2.3), the RDD-based API will be deprecated.

- The RDD-based API is expected to be removed in Spark 3.0.

- **Estimator**: An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer.

E.g., a learning algorithm is an Estimator which trains on a DataFrame and produces a model.

- **Transformer**: A Transformer is an algorithm which can transform one DataFrame into another DataFrame.

E.g., an ML model is a Transformer
It transforms a ”DataFrame with features” ➔ into a ”DataFrame with predictions”.

- **Pipeline**: A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow.

Pipeline

- A Estimator implements a method fit(), which accepts a DataFrame and produces a Model, which is a Transformer.

For example, a learning algorithm such as LogisticRegression is an Estimator

Calling fit() trains a LogisticRegressionModel, which is a Model and hence a Transformer.

- Pipeline, which consists of a sequence of **PipelineStages**

Estimators and Transformers to be run in a specific order

To get a rough idea :

```
val lr = new LogisticRegression() .setMaxIter(10) .setRegParam(0.001)

val pipeline = new Pipeline() .setStages(Array(tokenizer, hashingTF, lr))

// Fit the pipeline to training documents.
val model = pipeline.fit(training)

 // Now we can optionally save the fitted pipeline to disk
model.write.overwrite().save("/tmp/spark-logistic-regression-model")

// Make predictions on test documents.
model.transform(test)
```

So what all we have learnt ?

Web:	http://www.insofe.edu.in
Facebook:	https://www.facebook.com/insofe
Twitter:	https://twitter.com/Insofeedu
YouTube:	http://www.youtube.com/InsofeVideos
SlideShare:	http://www.slideshare.net/INSOFE
LinkedIn:	http://www.linkedin.com/company/international-school-of-engineering