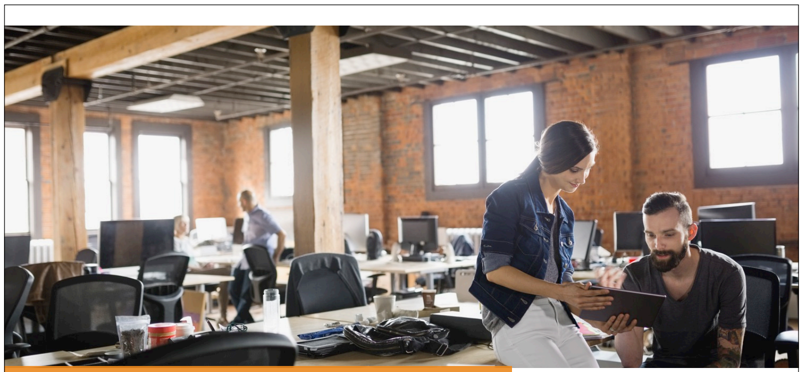
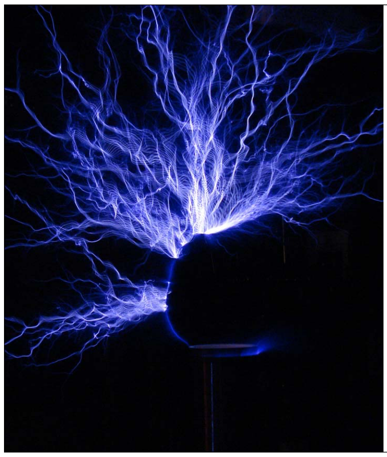
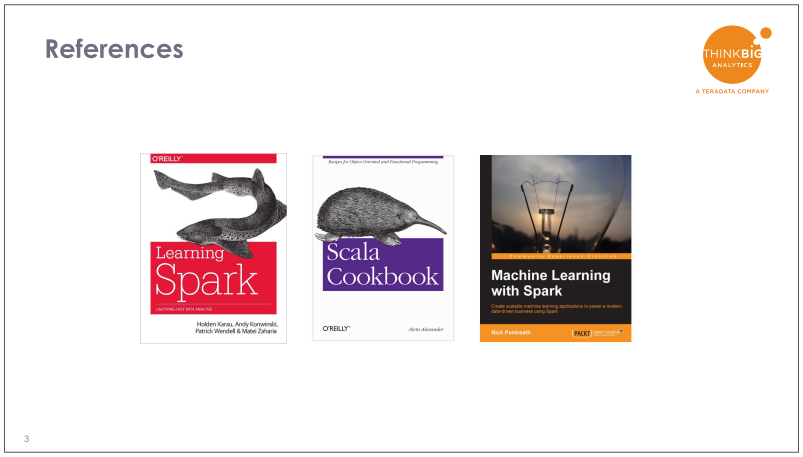


Spark Architecture and Concepts

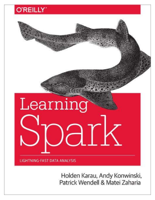


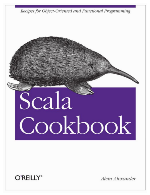
What we'll cover

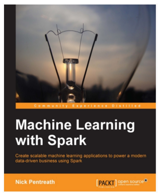
- Architecture
- Comparing Spark with MapReduce
- How to start the spark-shell
- Running some hands-on Spark code



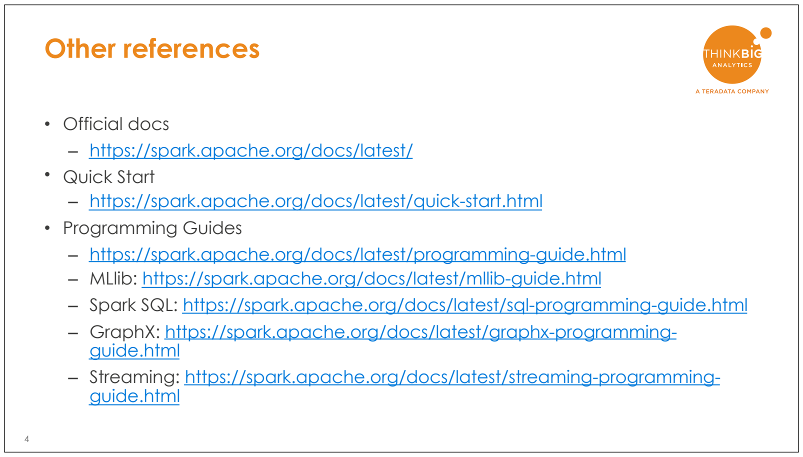
References


O'REILLY
Learning Spark
LIGHTNING-FAST DATA ANALYSIS
Holden Karau, Andy Konwinski, Patrick Wendell & Matei Zaharia




Machine Learning with Spark
Create scalable machine learning applications to power a modern data-driven business using Spark
Nick Pentreath
PACKT

Indispensable. Note that some of the details have changed since the 2nd Edition.

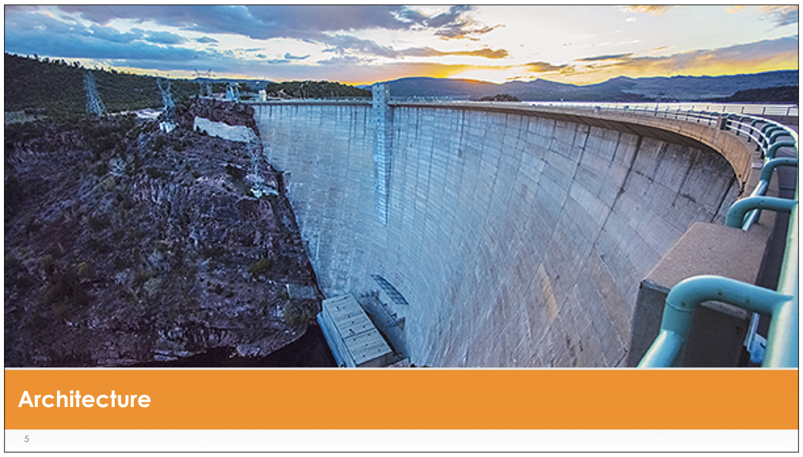


Other references

- Official docs
  - https://spark.apache.org/docs/latest/
- Quick Start
  - https://spark.apache.org/docs/latest/quick-start.html
- Programming Guides
  - https://spark.apache.org/docs/latest/programming-guide.html
  - MLlib: https://spark.apache.org/docs/latest/mllib-guide.html
  - Spark SQL: https://spark.apache.org/docs/latest/sql-programming-guide.html
  - GraphX: https://spark.apache.org/docs/latest/graphx-programming-guide.html
  - Streaming: https://spark.apache.org/docs/latest/streaming-programming-guide.html

4

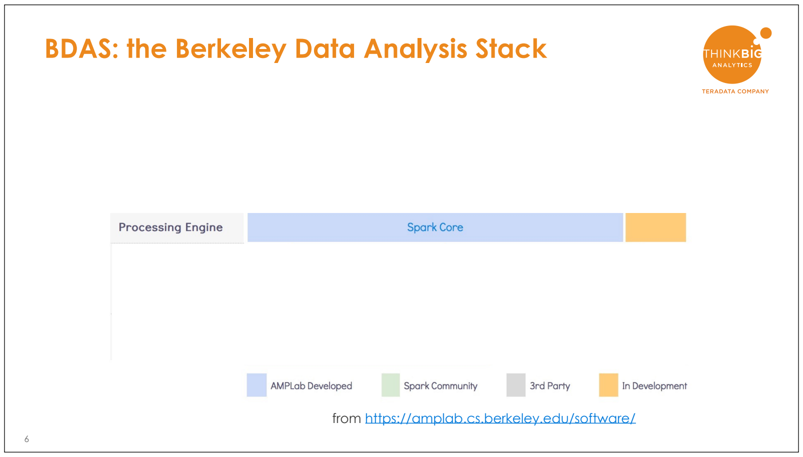
Because Spark changes so quickly—a new release comes along about every 3 months—the best references are the official Spark documentation. Just go to http://spark.apache.org/docs/latest to stay up to date.



Architecture

5

Spark's power comes from its Architecture. The Hadoop ecosystem evolved from the three main Google papers describing their equivalents of HDFS, MapReduce, and NoSQL. Other tools such as Hive, Pig, Kafka, and Storm simply evolved on top of that.



BDAS: the Berkeley Data Analysis Stack

Processing Engine	Spark Core	

🟦 AMPLab Developed 🟩 Spark Community ⬜ 3rd Party 🟧 In Development

from https://amplab.cs.berkeley.edu/software/

6

In contrast, Spark was designed by the Berkeley AMPLab, not as an add-on, but a complete software stack.

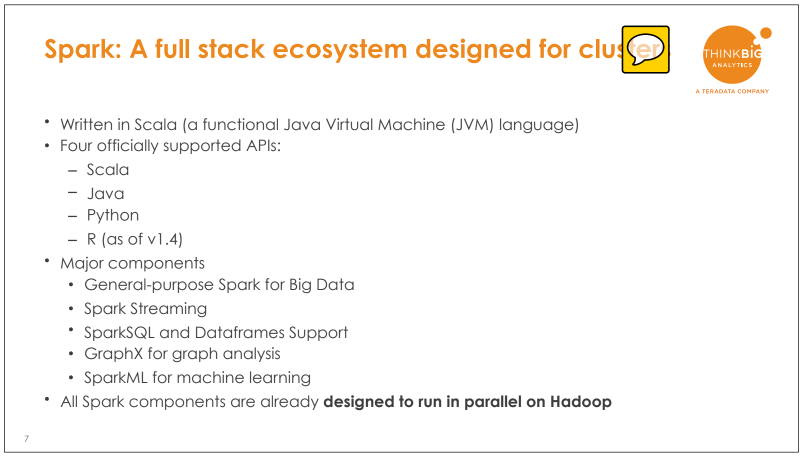
At the heart of Spark is the Spark Core. It contains the basic processing engine of Spark. However, that core was designed to run on top of other components. <<click to next build>>.

Now Spark can happily run on top of the traditional Hadoop components of YARN and HDFS. However, Spark was originally designed to use its own cluster OS, Mesos. And while Spark was designed to use HDFS as its primary storage, the AMPLab also added an in-memory accelerator to that called Tachyon.

And Berkeley didn't just work on the infrastructure of clusters. They also thought about the stack above the Spark Core and developed tools for applications. <<click for next build>>

On top of the core, Berkeley added a host of applications, which we'll discuss on the next slide.

In short, Spark is unlike Hadoop because all its components were designed to work together instead of simply evolving.



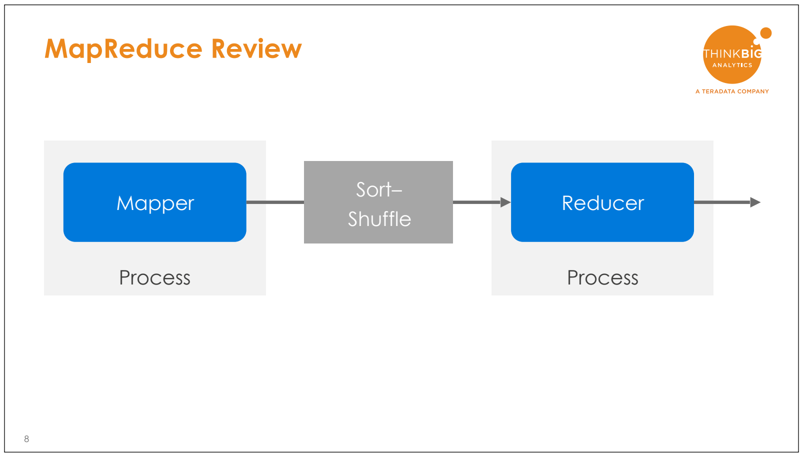
Spark: A full stack ecosystem designed for cluster

- Written in Scala (a functional Java Virtual Machine (JVM) language)
- Four officially supported APIs:
  - Scala
  - Java
  - Python
  - R (as of v1.4)
- Major components
  - General-purpose Spark for Big Data
  - Spark Streaming
  - SparkSQL and Dataframes Support
  - GraphX for graph analysis
  - SparkML for machine learning
- All Spark components are already **designed to run in parallel on Hadoop**

7

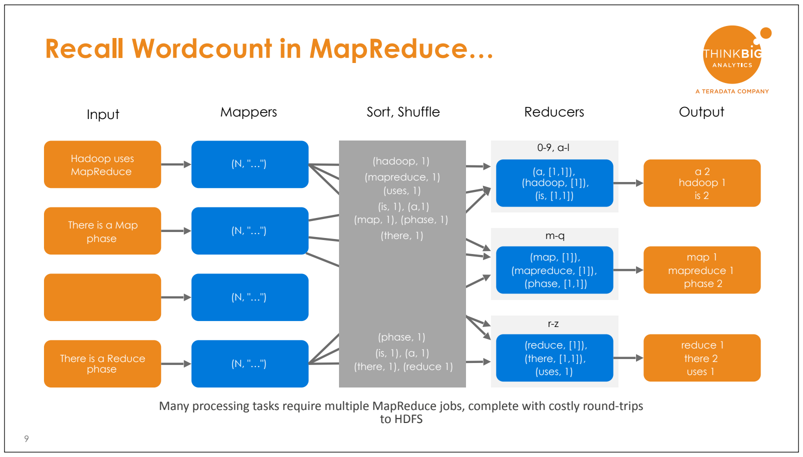
- Has R 1.4
- Has a parallel processes run

Spark is written in Scala, which is a Java Virtual Machine language. It natively supports four languages instead of Hadoop's one (Java). It hosts a wealth of applications and most importantly, every Spark application is **designed to run in parallel on a cluster.**



MapReduce Review

If you recall our discussion of WordCount when we talked about Big Data, It consisted of three major steps: Map, Sort-Shuffle, and Reduce. However, as you'll recall, more than that was going on.

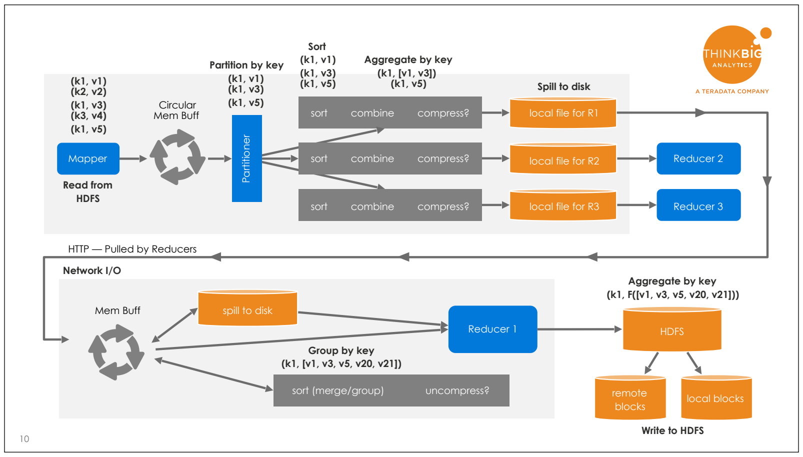


Recall Wordcount in MapReduce…

Many processing tasks require multiple MapReduce jobs, complete with costly round-trips to HDFS

9

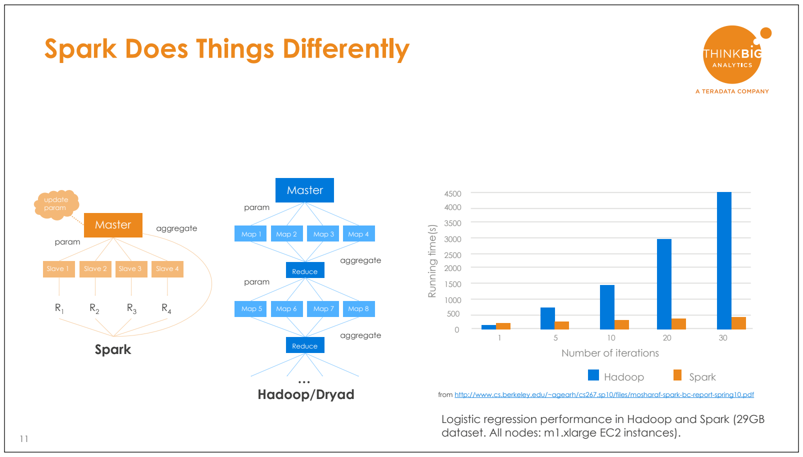
You remember that MapReduce was designed all around key-value pairs that contained bits and pieces of our Big Data. What we glossed over, though, is how we use these key-value pairs when they may contain terabytes or petabytes of information, and when cluster memory may only be measured in gigabytes.



In fact, what actually goes on is that MapReduce is designed so that it doesn't have to store those key value pairs in RAM. Instead, it has mechanisms to spill data to disk at every step of its processing.

The mapper writes to a circular memory buffer that spills to disk when it gets close to filling or at the end of the map process. The partitioner splits the K-V pairs into the same number of streams as there are reducers and each stream is sorted, the Combiner is applied (if there is one) and compressed (if turned on). The mapper's task tracker informs the job tracker when results are available, which informs the reducer's task tracker. The reducer then starts pulling data over using HTTP (even before the mapper is finished). The pulled data is stored in memory, spilling to disk as needed. The streams from the different mappers are merged (it's called a sort, but it's really a merge) after uncompressing, if necessary. The final merge step, reading from memory and disk, is fed into the reducer, which outputs to HDFS (or other…), with one replica of each block stored locally and the other replicas sent to other cluster nodes. Note that the reducer receives key-value pairs that are key-[val1,val2,…]. Deciding what vals belong with each key is called "grouping".

So while it might appear that MapReduce only uses HDFS for input and output, in fact it may be using disks all through the computation. Making this situation even worse is the fact that many Big Data jobs require multiple MapReduce steps.



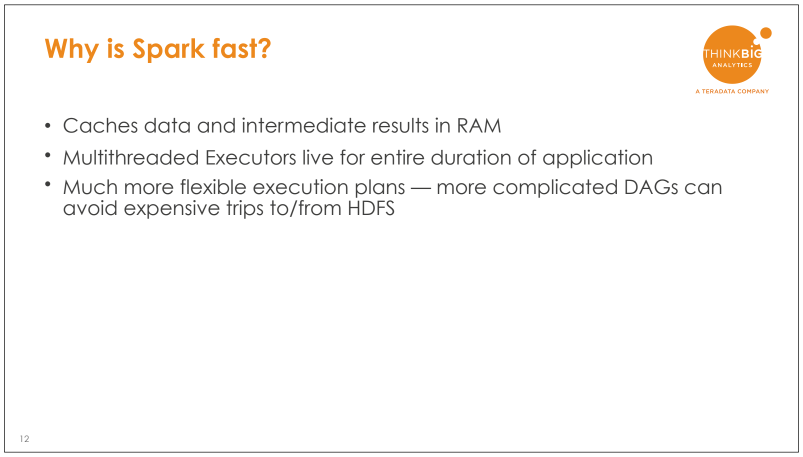
Spark Does Things Differently

from http://www.cs.berkeley.edu/~agearh/cs267.sp10/files/mosharaf-spark-bc-report-spring10.pdf

Logistic regression performance in Hadoop and Spark (29GB dataset. All nodes: m1.xlarge EC2 instances).

- Run anything that you want

Spark on the other hand does things differently. Instead of locking the user into a fixed Map-Shuffle-Reduce computational model, it allows developers to create arbitrary computations, represented as Directed Acyclical Graphs or DAGs. That combined with its in-memory storage allows Spark to outperform MapReduce in many Big Data workloads.



Why is Spark fast?

- Caches data and intermediate results in RAM
- Multithreaded Executors live for entire duration of application
- Much more flexible execution plans — more complicated DAGs can avoid expensive trips to/from HDFS

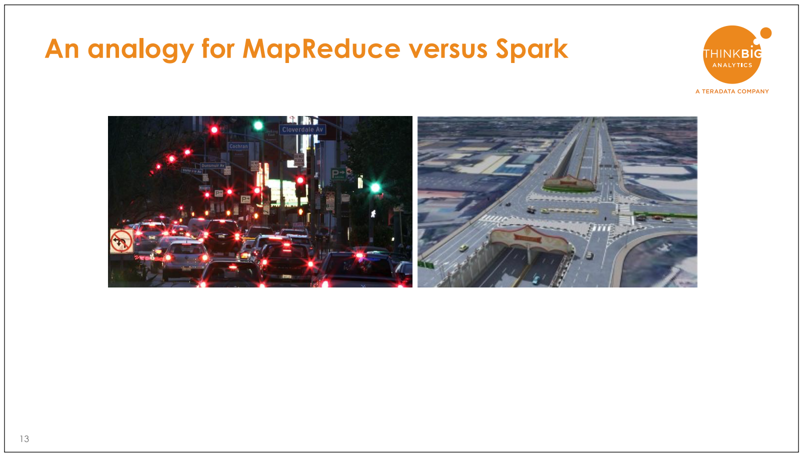
12

There are three reasons that Spark is fast.

The one that everyone knows is that Spark caches data in RAM. That's true and it does speed things up.

What a lot of people aren't aware of though is that some of Spark's speed comes from not always spinning Java Virtual Machines up and down. In MapReduce, a new JVM is started for each Map, Shuffle, and Reduce phase. In Spark, though, the JVMs are started once at the beginning of a job and only destroyed when the job is over. This makes various Spark job phases run much faster.

Finally, Spark has much more flexible execution plans. Instead of being locked into a Map-Shuffle-Reduce sequence, it can run arbitrary DAGs and optimize across the entire DAG. That means that Spark can often eliminate entire branches of computations if they aren't needed by the results requested.



An analogy for MapReduce versus Spark

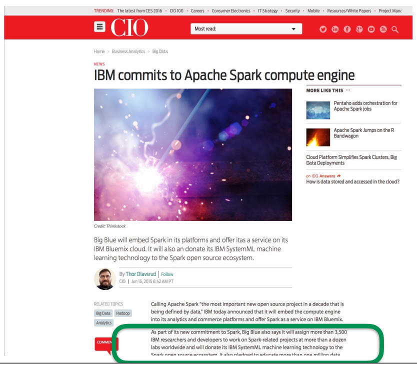
13

- DAGs -- Directed acyclical graphs

As an analogy, think of a MapReduce job as akin to a car traveling across a busy city with stoplights. The stoplights assure that the vehicle proceeds safely, but they do require it to stop periodically. In contrast, Spark is more like traveling on a superhighway. It has fixed entrances and exits, but in between, the vehicle can just speed along.

Industry has committed to Spark

THINKBIG ANALYTICS
A TERADATA COMPANY



TRENDING: The latest from CES 2016 · CIO 100 · Careers · Consumer Electronics · IT Strategy · Security · Mobile · Resources/White Papers · Project Man

CIO

Most read:

Home > Business Analytics > Big Data

NEWS

IBM commits to Apache Spark compute engine

Credit: Thinkstock

Big Blue will embed Spark in its platforms and offer itas a service on its IBM Bluemix cloud. It will also an donate its IBM SystemML machine learning technology to the Spark open source ecosystem.

By Thor Olavsrud | Follow
CIO | Jun 15, 2015 6:42 AM PT

RELATED TOPICS
Big Data Hadoop Analytics

Calling Apache Spark "the most important new open source project in a decade that is being defined by data," IBM today announced that it will embed the compute engine into its analytics and commerce platforms and offer Spark as a service on IBM Bluemix.

As part of its new commitment to Spark, Big Blue also says it will assign more than 3,500 IBM researchers and developers to work on Spark-related projects at more than a dozen labs worldwide and will donate its IBM SystemML machine learning technology to the Spark open source ecosystem. It also pledged to educate more than one million data

MORE LIKE THIS

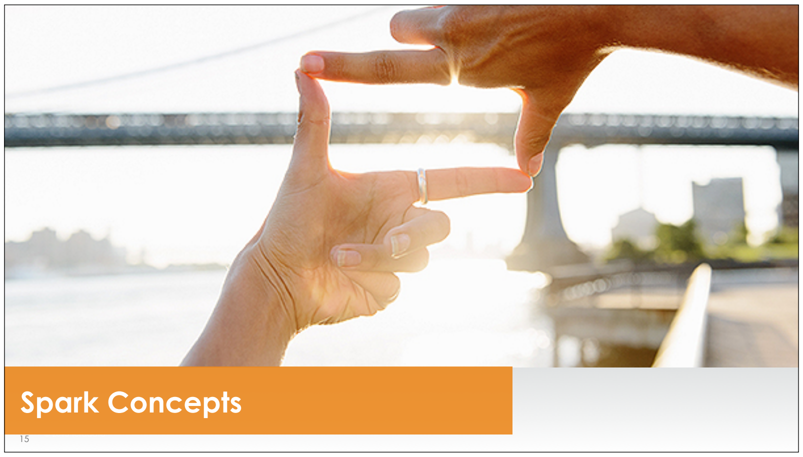
Pentaho adds orchestration for Apache Spark jobs

Apache Spark Jumps on the R Bandwagon

Cloud Platform Simplifies Spark Clusters, Big Data Deployments

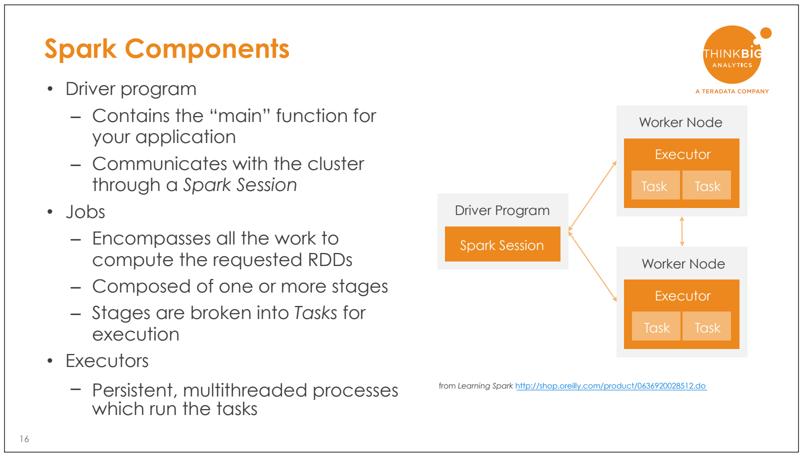
on IDG Answers
How is data stored and accessed in the cloud?

While there may be competing in-memory technologies—Apache Flink, for example, has many of the same characteristics as the Spark execution engine—the industry has put its money on Spark to date. IBM alone has roughly 3,500 Spark developers on staff already. And it is just one of many IT vendors supporting Spark.



Spark Concepts

How does Spark actually work then?



Spark Components

- Driver program
  - Contains the "main" function for your application
  - Communicates with the cluster through a *Spark Session*
- Jobs
  - Encompasses all the work to compute the requested RDDs
  - Composed of one or more stages
  - Stages are broken into *Tasks* for execution
- Executors
  - Persistent, multithreaded processes which run the tasks

THINKBIG ANALYTICS
A TERADATA COMPANY
Driver Program
Spark Session
Worker Node
Executor
Task
Task
Worker Node
Executor
Task
Task

from *Learning Spark* http://shop.oreilly.com/product/0636920028512.do

16

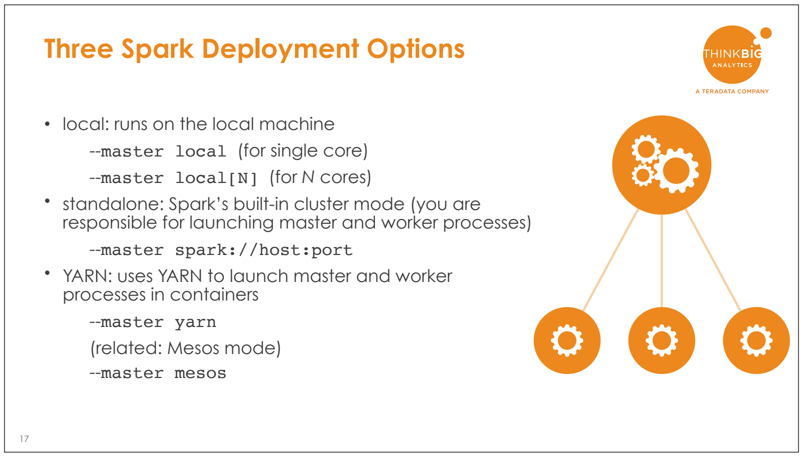
- In spark you can specify exactly how many resources are being used
- Spark session -- allows you to dictate the number of resources
- Jobs --
- User Interface for Spark

First, we'll define some terms.

The Spark Driver program contains the "main" function for your application and communicates with the cluster through a Spark Session.

Jobs encompass all the work to compute the requested distributed datasets, which are called RDDs. A job consists of one or more stages. Stages, in turn, are broken into Tasks for execution.

Persistent, multithreaded processes called Executors actually run the tasks. Each worker node making up a Spark cluster will have one or more Spark executors.



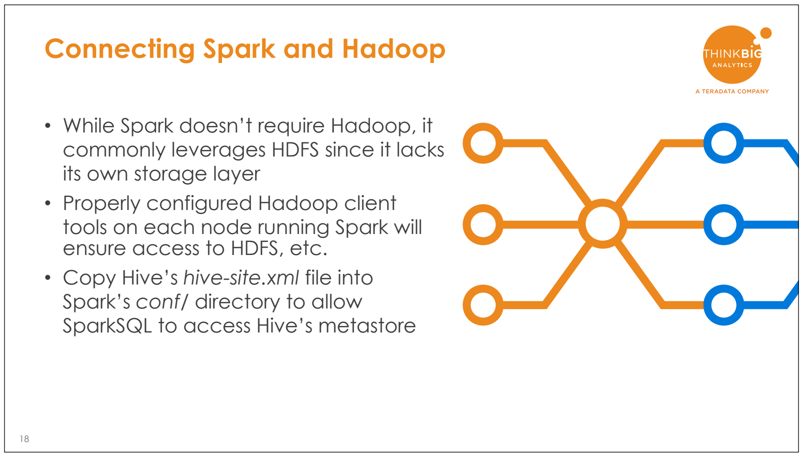
Three Spark Deployment Options

- local: runs on the local machine
  `--master local` (for single core)
  `--master local[N]` (for *N* cores)
- standalone: Spark's built-in cluster mode (you are responsible for launching master and worker processes)
  `--master spark://host:port`
- YARN: uses YARN to launch master and worker processes in containers
  `--master yarn`
  (related: Mesos mode)
  `--master mesos`

Spark can run in three ways:

1. local mode which only runs on one processor, albeit using multiple cores.
2. standalone mode, which is where Spark owns the cluster. No other jobs other than Spark ones run in standalone mode. YARN isn't run either.
3. YARN mode. This is the most common mode, and it's one where the Spark driver and executors run in containers managed by YARN.

Mesos mode is also available in case you are running a Mesos cluster, but that isn't common.



Connecting Spark and Hadoop

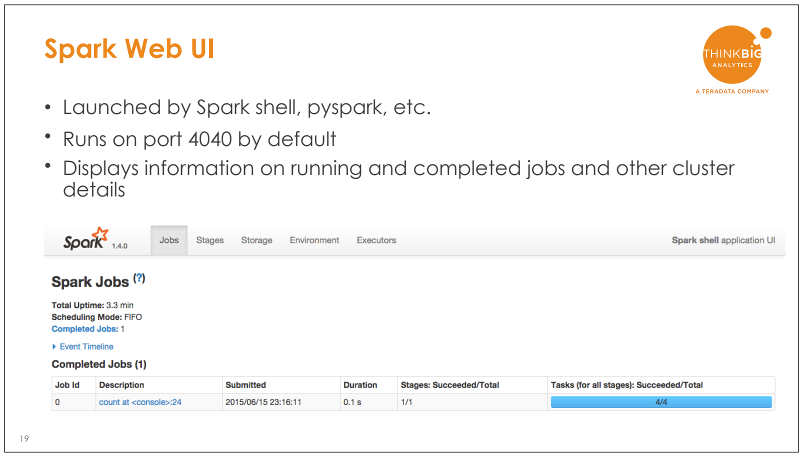
- While Spark doesn't require Hadoop, it commonly leverages HDFS since it lacks its own storage layer
- Properly configured Hadoop client tools on each node running Spark will ensure access to HDFS, etc.
- Copy Hive's *hive-site.xml* file into Spark's *conf/* directory to allow SparkSQL to access Hive's metastore

18

- Needs to be directed to the Metadata

Spark is independent of Hadoop, but needs to connect to it because it needs a storage layer.

While Spark doesn't require Hadoop, it commonly leverages HDFS since it lacks its own storage layer
Properly configured Hadoop client tools on each node running Spark will ensure access to HDFS, etc.
Copy Hive's hive-site.xml file into Spark's conf/ directory to allow SparkSQL to access Hive's metastore



Spark Web UI

- Launched by Spark shell, pyspark, etc.
- Runs on port 4040 by default
- Displays information on running and completed jobs and other cluster details

Spark 1.4.0
Jobs
Stages
Storage
Environment
Executors
Spark shell application UI

Spark Jobs (?)

Total Uptime: 3.3 min
Scheduling Mode: FIFO
Completed Jobs: 1

Event Timeline

Completed Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	count at <console>:24	2015/06/15 23:16:11	0.1 s	1/1	4/4

A Web user interface for Spark is started up whenever a Spark job runs. Typically this will be available on port 4040. However, if another job is still running, the Spark UI may be started on a sequential port afterward, such as 4041 or 4042. Spark will usually tell you what port its GUI is running on.

This user interface has a huge amount of information about what Spark is doing and how the applications are running. You should spend time exploring it sometime when you have run a few Spark jobs and see all the information it provides.

Getting started with Spark in Scala

To run bin/spark-shell on exactly four cores, use:

```
$ ./bin/spark-shell --master local[4]
```

Or, to also add code.jar to its classpath, use:

```
$ ./bin/spark-shell --master local[4] --jars code.jar
```

For a complete list of options, run spark-shell --help. Behind the scenes, spark-shell invokes the more general `spark-submit` script.

Getting started with Spark in Python

For example, to run `bin/pyspark` on exactly four cores, use:

```
$ ./bin/pyspark --master local[4]
```

Or, to also add `code.py` to the search path (in order to later be able to `import code`), use:

```
$ ./bin/pyspark --master local[4] --py-files code.py
```

For a complete list of options, run `pyspark --help`. Behind the scenes, `pyspark` invokes the more general `spark-submit` script.

Spark and MapReduce Comparison Exercise

22

We now going to perform three labs which will get you used to connecting to your clusters. You should have a directory with all the course materials called sparkclass. Underneath that directory are several other directories including

data
exercises
handouts
images
slides

You'll find these labs in your exercises/Hadoop-Architecture directory.

You can read these labs in either pdf or Markdown form. The PDF form is prettier, but the text in the Markdown version will be easier to copy and paste when you get to complicated commands.

Go ahead and run those three labs. We'll allot about 15 minutes for you to do this.

Comparing Spark and MapReduce Speed

Vagrant version

- `Vagrant ssh edge` to your home directory (`cd ~`)
- Time and run MapReduce Pi Estimator for 100000 samples

```
time hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples.jar pi 4 100000
```

- Your run time is probably around 25 seconds

- Now run the same example under Spark. This example will actually run 400000 samples

```
sudo su
sed 's/INFO/WARN/g' /vagrant/latest-spark/conf/log4j.properties.template \
     >/vagrant/latest-spark/conf/log4j.properties
exit
time $SPARK_HOME/bin/run-example --master local[*] SparkPi 4
```

- Your run time is probably around 8 seconds

Comparing Spark and MapReduce Speed

EMR version

- ssh to your home directory on the cluster (cd ~)
- Time and run MapReduce Pi Estimator for 100000 samples

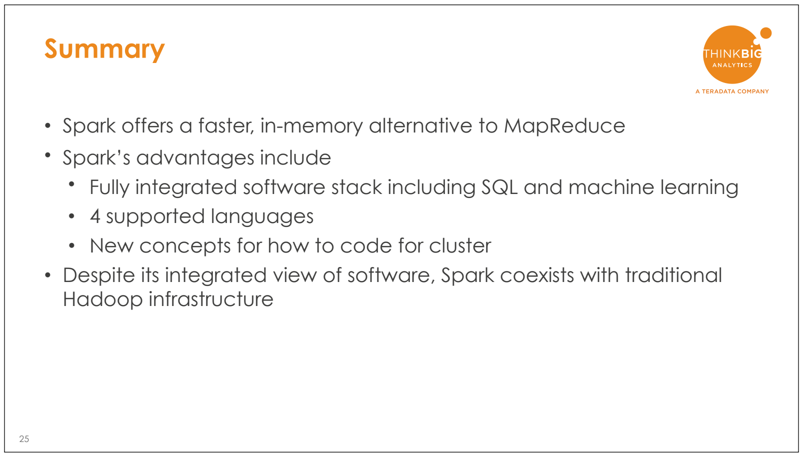
```
time hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-2.7.3-amzn-3.jar pi 4 100000
```

- Your run time is probably around 28 seconds

- Now run the same example under Spark.
- Turn down Spark logging by typing the following

```
sudo su
sed 's/INFO/WARN/g' /etc/spark/conf/log4j.properties.template >/etc/spark/conf/log4j.properties
exit
time /usr/lib/spark/bin/run-example --master local[*] SparkPi 4
```

- Your run time is probably around 7 seconds
- Note that user time may be greater than real time, showing parallel execution



Summary

- Spark offers a faster, in-memory alternative to MapReduce
- Spark's advantages include
  - Fully integrated software stack including SQL and machine learning
  - 4 supported languages
  - New concepts for how to code for cluster
- Despite its integrated view of software, Spark coexists with traditional Hadoop infrastructure

25

- These are the 5 principles
  - Using resources to save resources
  - etc.