Apache Spark

In-Memory Data Processing

September 2014 Meetup

Organized by **Big Data Hyderabad Meetup Group**.

http://www.meetup.com/Big-Data-Hyderabad/

Rahul Jain

@rahuldausa

Agenda

- Why Spark
- Introduction
- Basics
- Hands-on
  - Installation
  - Examples

Quick Questionnaire

How many people know/work on ***Scala* ?**

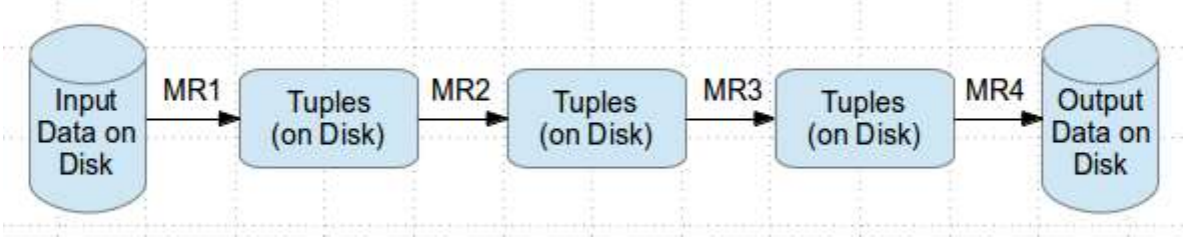
How many people **know/work on *Python*** ?

How many people know/heard/are using ***Spark*** ?

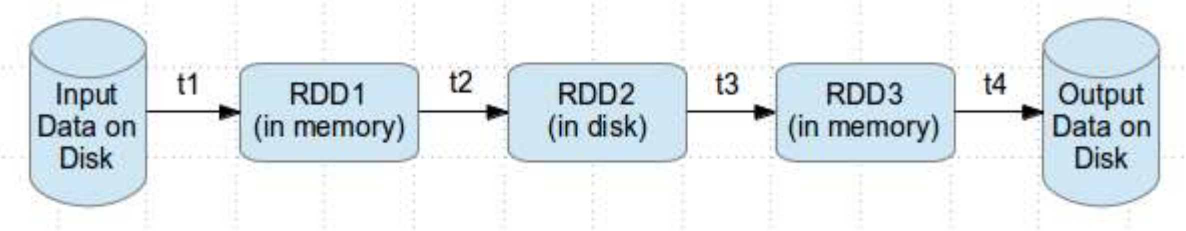
Why Spark ?

- Most of Machine Learning Algorithms are iterative because each iteration can improve the results
- With Disk based approach each iteration's output is written to disk making it slow

Hadoop execution flow



Spark execution flow



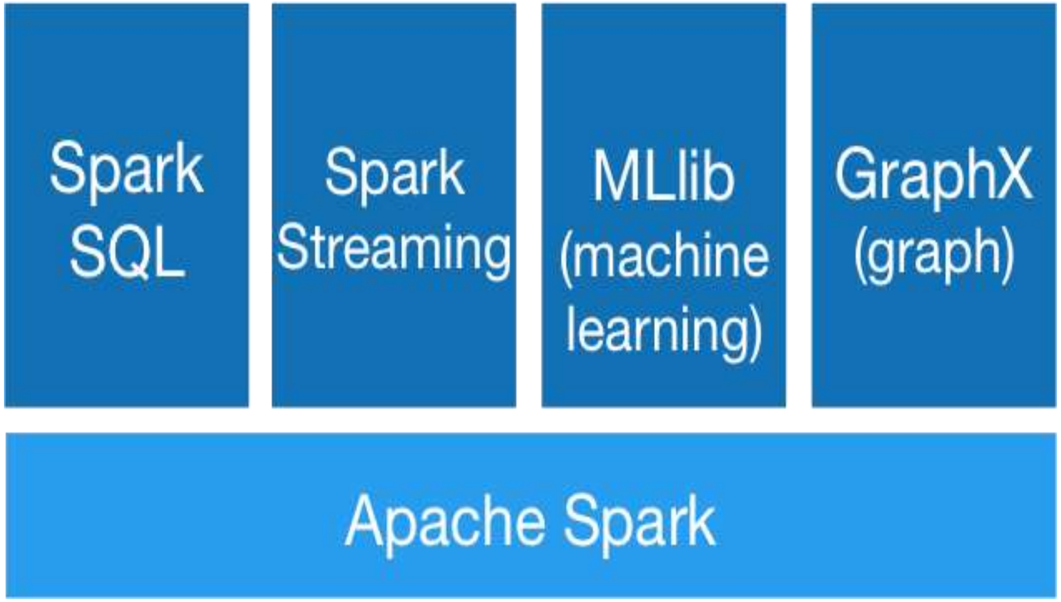
http://www.wiziq.com/blog/hype-around-apache-spark/

About Apache Spark

- Initially started at UC Berkeley in 2009
- Fast and general purpose cluster computing system
- 10x (on disk) - 100x (In-Memory) faster
- Most popular for running *Iterative Machine Learning Algorithms.*
- Provides high level APIs in
  - Java
  - Scala
  - Python
- Integration with Hadoop and its eco-system and can **read existing data.**
- http://spark.apache.org/

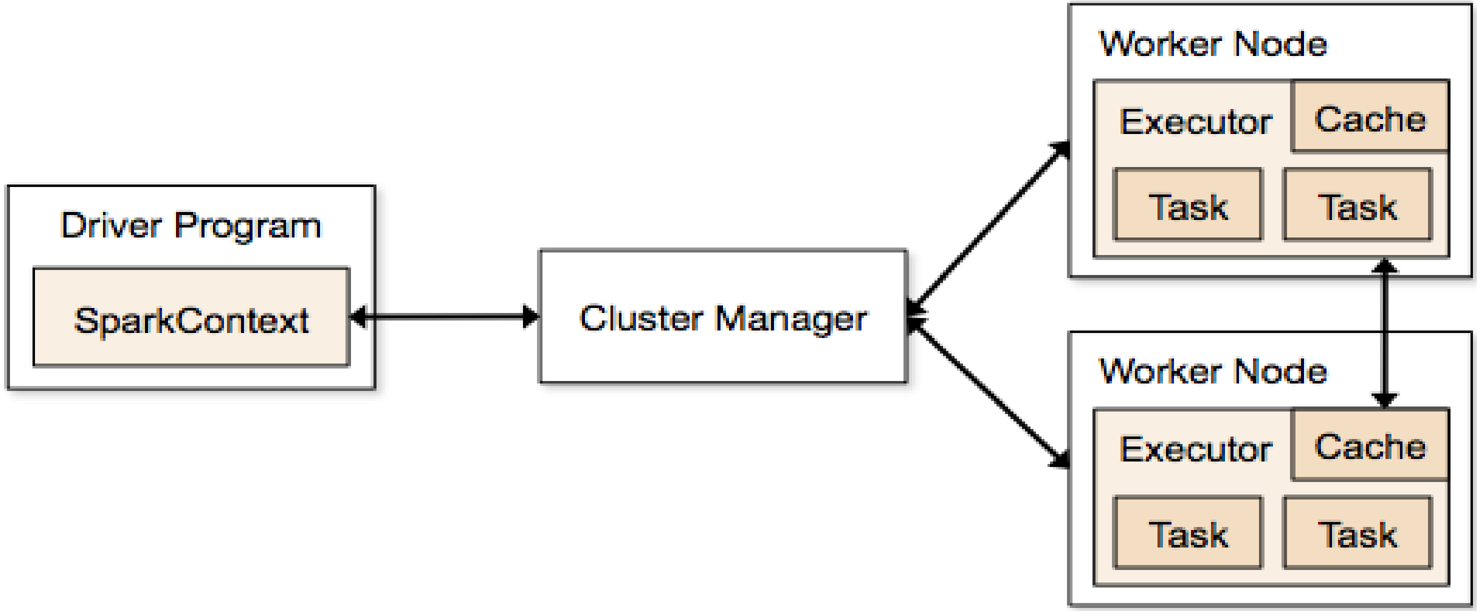
Spark Stack

- Spark SQL
  - For SQL and unstructured data processing
- MLib
  - Machine Learning Algorithms
- GraphX
  - Graph Processing
- Spark Streaming
  - stream processing of live data streams



http://spark.apache.org

Execution Flow



http://spark.apache.org/docs/latest/cluster-overview.html

Terminology

- **Application Jar**
  - User Program and its dependencies except Hadoop & Spark Jars bundled into a Jar file
- **Driver Program**
  - The process to start the execution (main() function)
- **Cluster Manager**
  - An external service to manage resources on the cluster (standalone manager, YARN, Apache Mesos)
- **Deploy Mode**
  - **cluster** : Driver inside the cluster
  - **client** : Driver outside of Cluster

Terminology (contd.)

- **Worker Node :** Node that run the application program in cluster
- **Executor**
  - Process launched on a worker node, that runs the Tasks
  - Keep data in memory or disk storage
- **Task :** A unit of work that will be sent to executor
- **Job**
  - Consists multiple tasks
  - Created based on a Action
- **Stage :** Each Job is divided into smaller set of tasks called Stages that is sequential and depend on each other
- **SparkContext :**
  - represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.

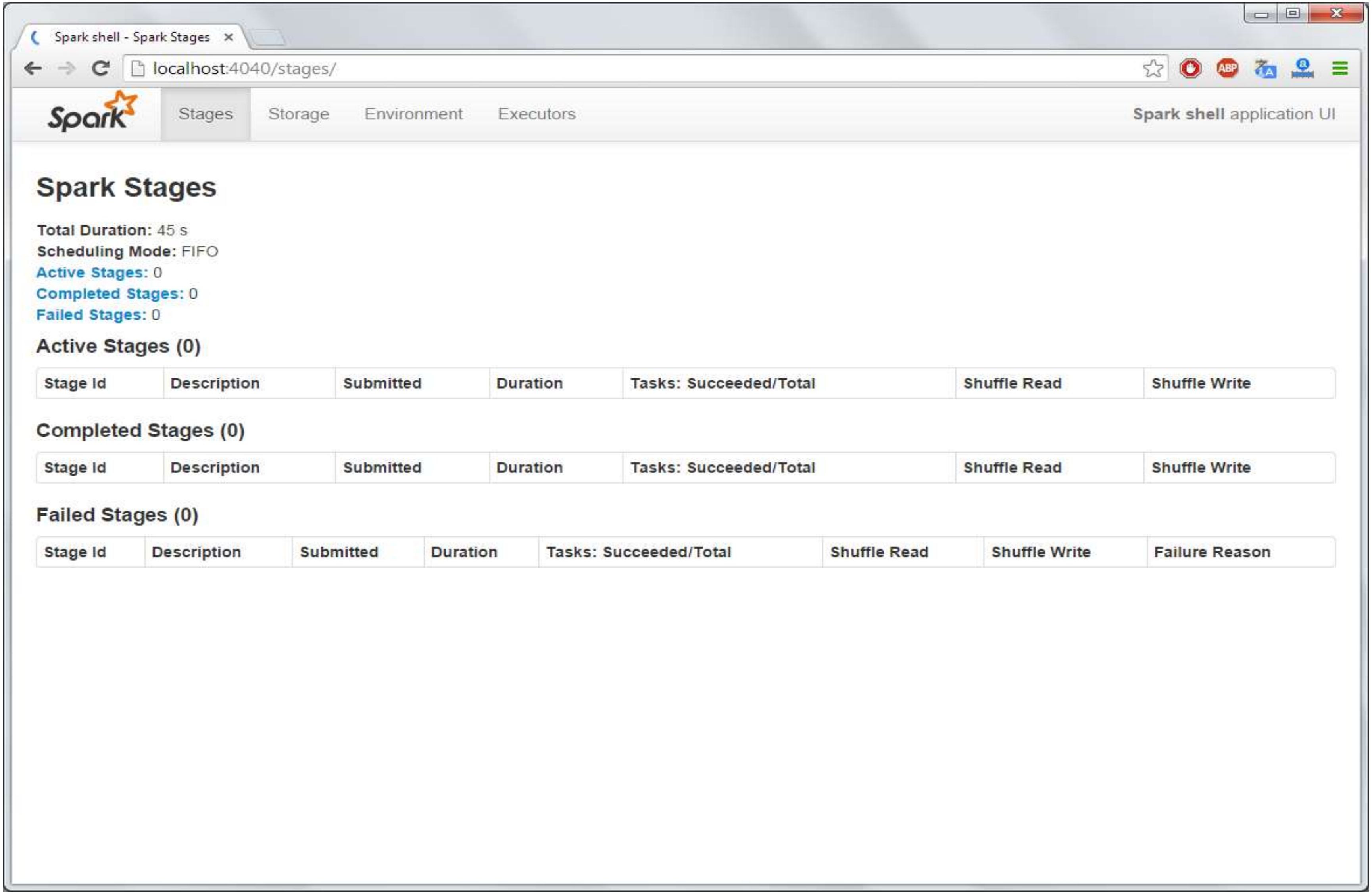
Resilient Distributed Dataset (RDD)

- Resilient Distributed Dataset (RDD) is a basic Abstraction in Spark
- Immutable, Partitioned collection of elements that can be operated in parallel
- Basic Operations
  - map
  - filter
  - persist
- Multiple Implementation
  - PairRDDFunctions : RDD of Key-Value Pairs, groupByKey, Join
  - DoubleRDDFunctions : Operation related to double values
  - SequenceFileRDDFunctions : Operation related to SequenceFiles
- RDD main characteristics:
  - A list of partitions
  - A function for computing each split
  - A list of dependencies on other RDDs
  - Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)
  - Optionally, a list of preferred locations to compute each split on (e.g. block locations for an HDFS file)
- Custom RDD can be also implemented (by overriding functions)

Cluster Deployment

- Standalone Deploy Mode
  - simplest way to deploy Spark on a private cluster
- Amazon EC2
  - EC2 scripts are available
  - Very quick launching a new cluster
- Apache Mesos
- Hadoop YARN

Monitoring



Spark Stages

Total Duration: 45 s
Scheduling Mode: FIFO
Active Stages: 0
Completed Stages: 0
Failed Stages: 0

Active Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write

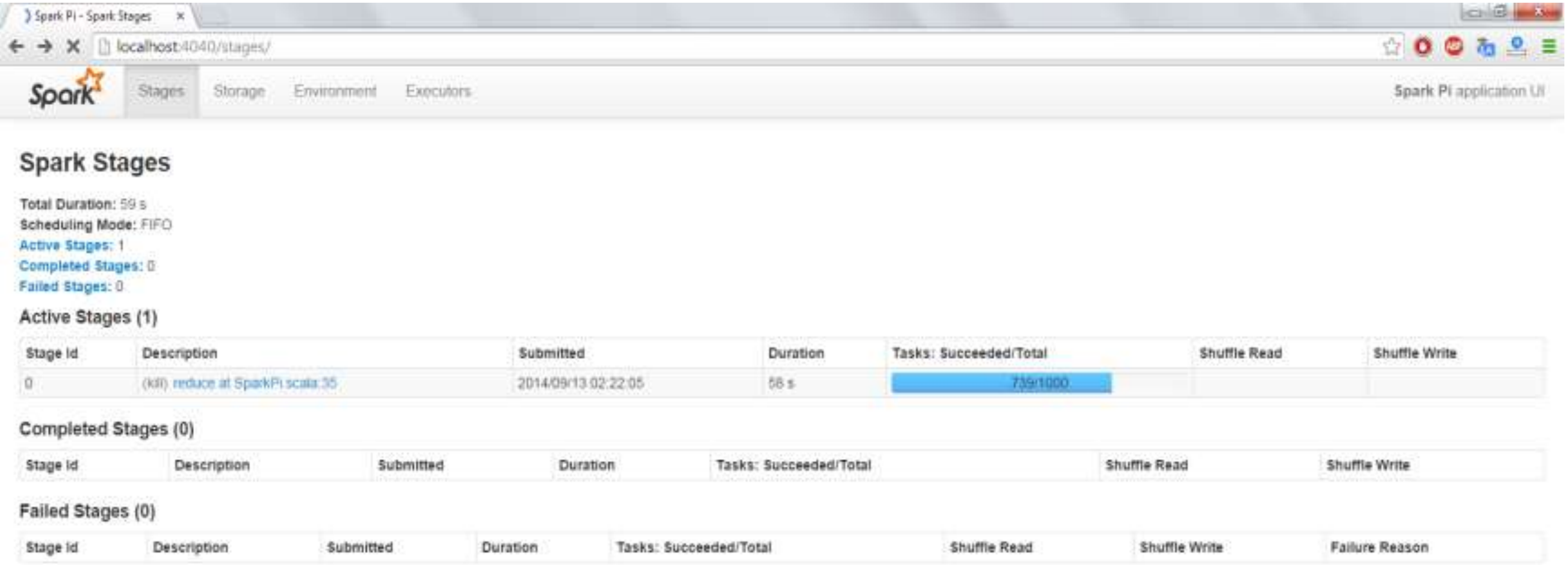
Completed Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write

Failed Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write	Failure Reason

Monitoring – Stages



Spark Stages

Total Duration: 59 s
Scheduling Mode: FIFO
Active Stages: 1
Completed Stages: 0
Failed Stages: 0

Active Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write
0	(kill) reduce at SparkPi.scala:35	2014/09/13 02:22:05	58 s	739/1000		

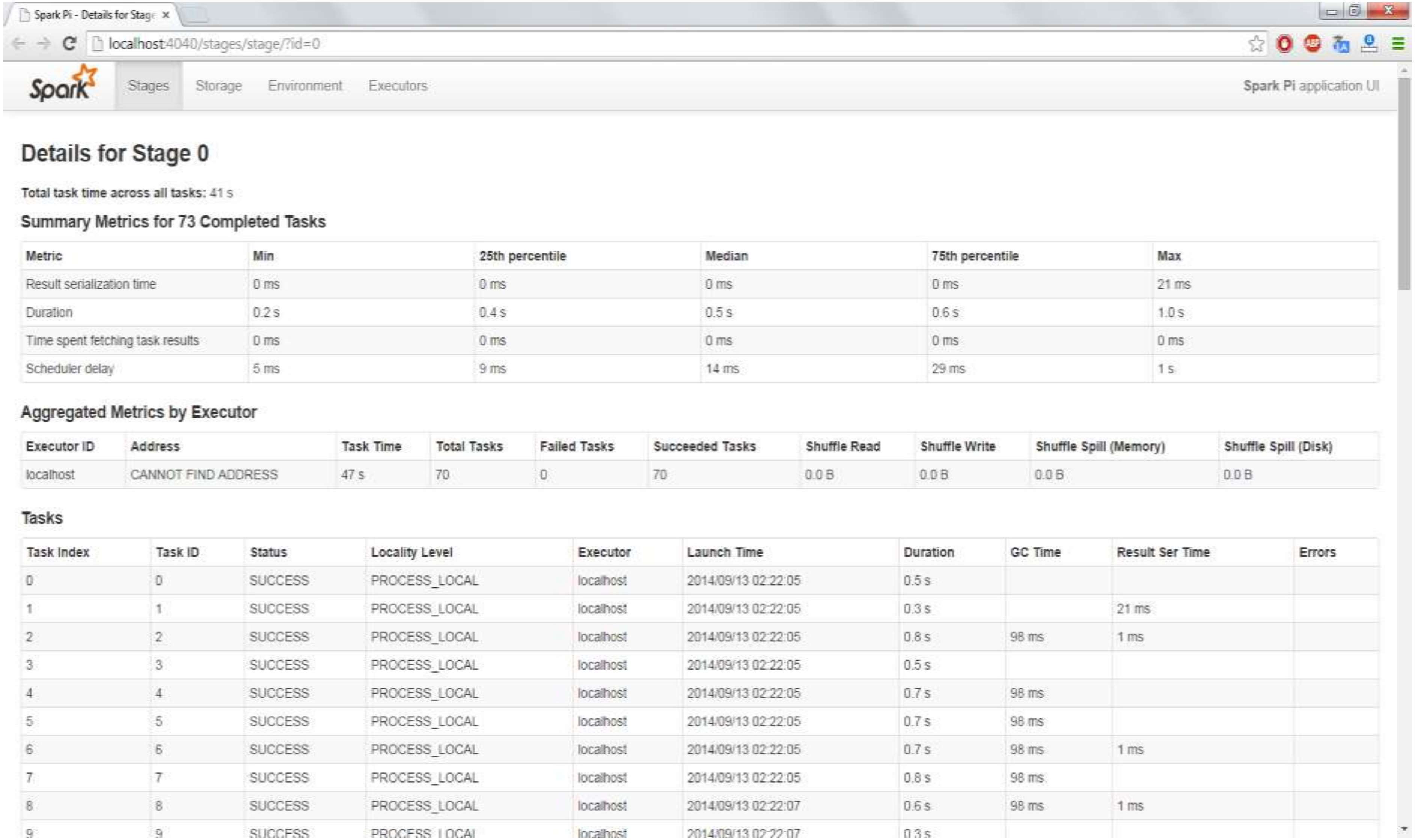
Completed Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write

Failed Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write	Failure Reason

Monitoring – Stages


Spark Pi - Details for Stage
localhost:4040/stages/stage/?id=0
Spark
Stages
Storage
Environment
Executors
Spark Pi application UI

Details for Stage 0

Total task time across all tasks: 41 s

Summary Metrics for 73 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Result serialization time	0 ms	0 ms	0 ms	0 ms	21 ms
Duration	0.2 s	0.4 s	0.5 s	0.6 s	1.0 s
Time spent fetching task results	0 ms	0 ms	0 ms	0 ms	0 ms
Scheduler delay	5 ms	9 ms	14 ms	29 ms	1 s

Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Shuffle Read	Shuffle Write	Shuffle Spill (Memory)	Shuffle Spill (Disk)
localhost	CANNOT FIND ADDRESS	47 s	70	0	70	0.0 B	0.0 B	0.0 B	0.0 B

Tasks

Task Index	Task ID	Status	Locality Level	Executor	Launch Time	Duration	GC Time	Result Ser Time	Errors
0	0	SUCCESS	PROCESS_LOCAL	localhost	2014/09/13 02:22:05	0.5 s			
1	1	SUCCESS	PROCESS_LOCAL	localhost	2014/09/13 02:22:05	0.3 s		21 ms	
2	2	SUCCESS	PROCESS_LOCAL	localhost	2014/09/13 02:22:05	0.8 s	98 ms	1 ms	
3	3	SUCCESS	PROCESS_LOCAL	localhost	2014/09/13 02:22:05	0.5 s			
4	4	SUCCESS	PROCESS_LOCAL	localhost	2014/09/13 02:22:05	0.7 s	98 ms		
5	5	SUCCESS	PROCESS_LOCAL	localhost	2014/09/13 02:22:05	0.7 s	98 ms		
6	6	SUCCESS	PROCESS_LOCAL	localhost	2014/09/13 02:22:05	0.7 s	98 ms	1 ms	
7	7	SUCCESS	PROCESS_LOCAL	localhost	2014/09/13 02:22:05	0.8 s	98 ms		
8	8	SUCCESS	PROCESS_LOCAL	localhost	2014/09/13 02:22:07	0.6 s	98 ms	1 ms	
9	9	SUCCESS	PROCESS_LOCAL	localhost	2014/09/13 02:22:07	0.3 s			

Let’s try some examples…

Spark Shell

```
./bin/spark-shell --master local[2]
```

The --master option specifies the master URL for a distributed cluster, or local to run locally with one thread, or local[N] to run locally with N threads. You should start by using local for testing.

```
./bin/run-example SparkPi 10
```

This will run 10 iterations to calculate the value of Pi

Basic operations…

```
scala> val textFile = sc.textFile("README.md")
textFile: spark.RDD[String] = spark.MappedRDD@2ee9b6e3
```

```
scala> textFile.count() // Number of items in this RDD
ees0: Long = 126

scala> textFile.first() // First item in this RDD
res1: String = # Apache Spark
```

```
scala> val linesWithSpark = textFile.filter(line =>
line.contains("Spark"))
linesWithSpark: spark.RDD[String] = spark.FilteredRDD@7dd4af09
```

```
Simplier - Single liner:
scala> textFile.filter(line => line.contains("Spark")).count()
// How many lines contain "Spark"?
res3: Long = 15
```

Map - Reduce

```
scala> textFile.map(line => line.split(" ").size).reduce((a, b)
=> if (a > b) a else b)
res4: Long = 15
```

```
scala> import java.lang.Math
scala> textFile.map(line => line.split(" ").size).reduce((a, b)
=> Math.max(a, b))
res5: Int = 15
```

```
scala> val wordCounts = textFile.flatMap(line => line.split("
")).map(word => (word, 1)).reduceByKey((a, b) => a + b)
wordCounts: spark.RDD[(String, Int)] =
spark.ShuffledAggregatedRDD@71f027b8
wordCounts.collect()
```

With Caching…

```
scala> linesWithSpark.cache()
res7: spark.RDD[String] = spark.FilteredRDD@17e51082

scala> linesWithSpark.count()
res8: Long = 15

scala> linesWithSpark.count()
res9: Long = 15
```

With HDFS…

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(line => line.startsWith(“ERROR”))
println(Total errors: + errors.count())
```

Standalone (Scala)

```
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {

  def main(args: Array[String]) {
    val logFile = "YOUR_SPARK_HOME/README.md" // Should be some file on your 
system
    val conf = new SparkConf().setAppName("Simple Application")
.setMaster(“local")
    val sc = new SparkContext(conf)
    val logData = sc.textFile(logFile, 2).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }
}
```

Standalone (Java)

```java
/* SimpleApp.java */
import org.apache.spark.api.java.*;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.function.Function;

public class SimpleApp {

  public static void main(String[] args) {

    String logFile = "YOUR_SPARK_HOME/README.md"; // Should be some file on your system
    SparkConf conf = new SparkConf().setAppName("Simple Application").setMaster("local");
    JavaSparkContext sc = new JavaSparkContext(conf);
    JavaRDD<String> logData = sc.textFile(logFile).cache();

    long numAs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("a"); }
    }).count();

    long numBs = logData.filter(new Function<String, Boolean>() {
      public Boolean call(String s) { return s.contains("b"); }
    }).count();

    System.out.println("Lines with a: " + numAs + ", lines with b: " + numBs);
  }
}
```

Standalone (Python)

```python
"""SimpleApp.py"""
from pyspark import SparkContext

logFile = "YOUR_SPARK_HOME/README.md" # Should be some file on your system
sc = SparkContext("local", "Simple App")
logData = sc.textFile(logFile).cache()

numAs = logData.filter(lambda s: 'a' in s).count()
numBs = logData.filter(lambda s: 'b' in s).count()

print "Lines with a: %i, lines with b: %i" % (numAs, numBs)
```

Job Submission

```
$SPARK_HOME/bin/spark-submit \
  --class "SimpleApp" \
  --master local[4] \
  target/scala-2.10/simple-project_2.10-1.0.jar
```

Configuration

```
val conf = new SparkConf()
       .setMaster("local")
       .setAppName("CountingSheep")
       .set("spark.executor.memory", "1g")

val sc = new SparkContext(conf)
```

Questions ?

Thanks!

@rahuldausa on twitter and slideshare
http://www.linkedin.com/in/rahuldausa

Join us @ For Solr, Lucene, Elasticsearch, Machine Learning, IR
http://www.meetup.com/Hyderabad-Apache-Solr-Lucene-Group/
http://www.meetup.com/DataAnalyticsGroup/

Join us @ For Hadoop, Spark, Cascading, Scala, NoSQL, Crawlers and all cutting edge technologies.
http://www.meetup.com/Big-Data-Hyderabad/