Real time Analytics

with Apache Kafka and Spark

October 2014 Meetup

Organized by **Big Data Hyderabad**.

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

Rahul Jain

@rahuldausa

About Me

- Big data/Search Consultant
- 8+ years of learning experience.
- Worked (got a chance) on High volume distributed applications.
- Still a learner (and beginner)

Quick Questionnaire

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

How many people **know/work on *Apache Kafka***?

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

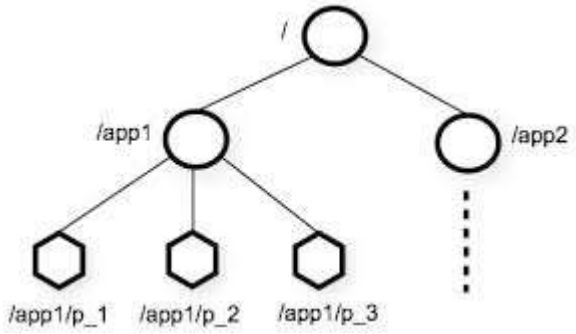
What we are going to learn/see today ?

- Apache Zookeeper (Overview)
- Apache Kafka – Hands-on
- Apache Spark – Hands-on
- Spark Streaming (Explore)

Apache Zookeeper ™

What is Zookeeper

- An Open source, High Performance coordination service for distributed applications
- Centralized service for
  - Configuration Management
  - Locks and Synchronization for providing coordination between distributed systems
  - Naming service (Registry)
  - Group Membership
- Features
  - hierarchical namespace
  - provides watcher on a znode
  - allows to form a cluster of nodes
- Supports a large volume of request for data retrieval and update
- http://zookeeper.apache.org/



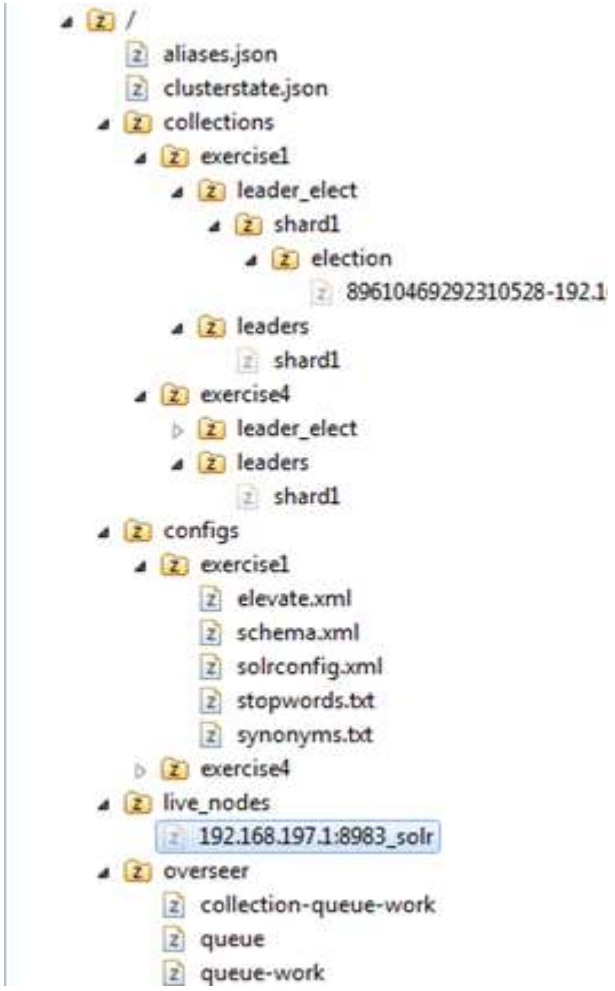
Source : http://zookeeper.apache.org

Zookeeper Use cases

- Configuration Management
  - Cluster member nodes Bootstrapping configuration from a central source
- Distributed Cluster Management
  - Node Join/Leave
  - Node Status in real time
- Naming Service – e.g. DNS
- Distributed Synchronization – locks, barriers
- Leader election
- Centralized and Highly reliable Registry

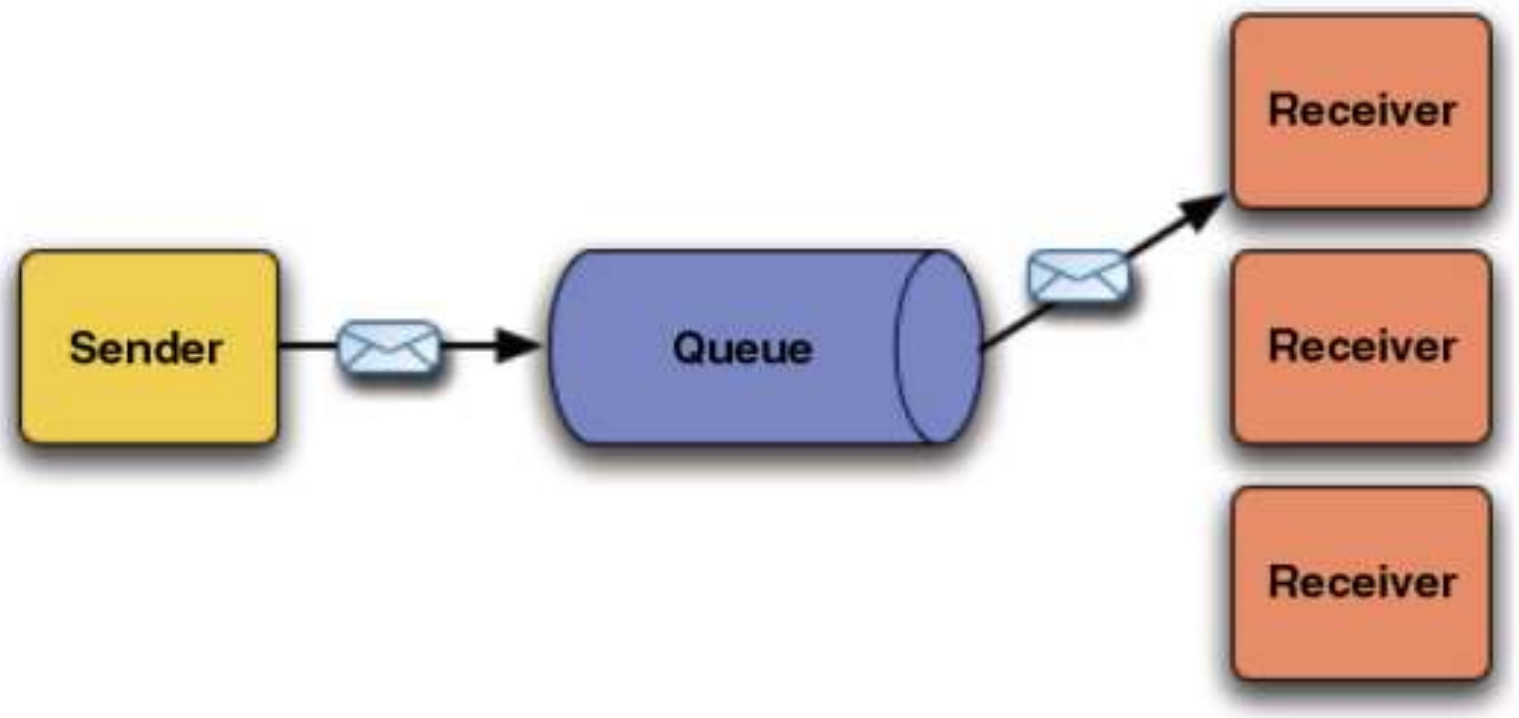
Zookeeper Data Model

- Hierarchical Namespace
- Each node is called “znode”
- Each znode has data(stores data in byte[] array) and can have children
- znode
  - Maintains “Stat” structure with version of data changes , ACL changes and timestamp
  - Version number increases with each changes



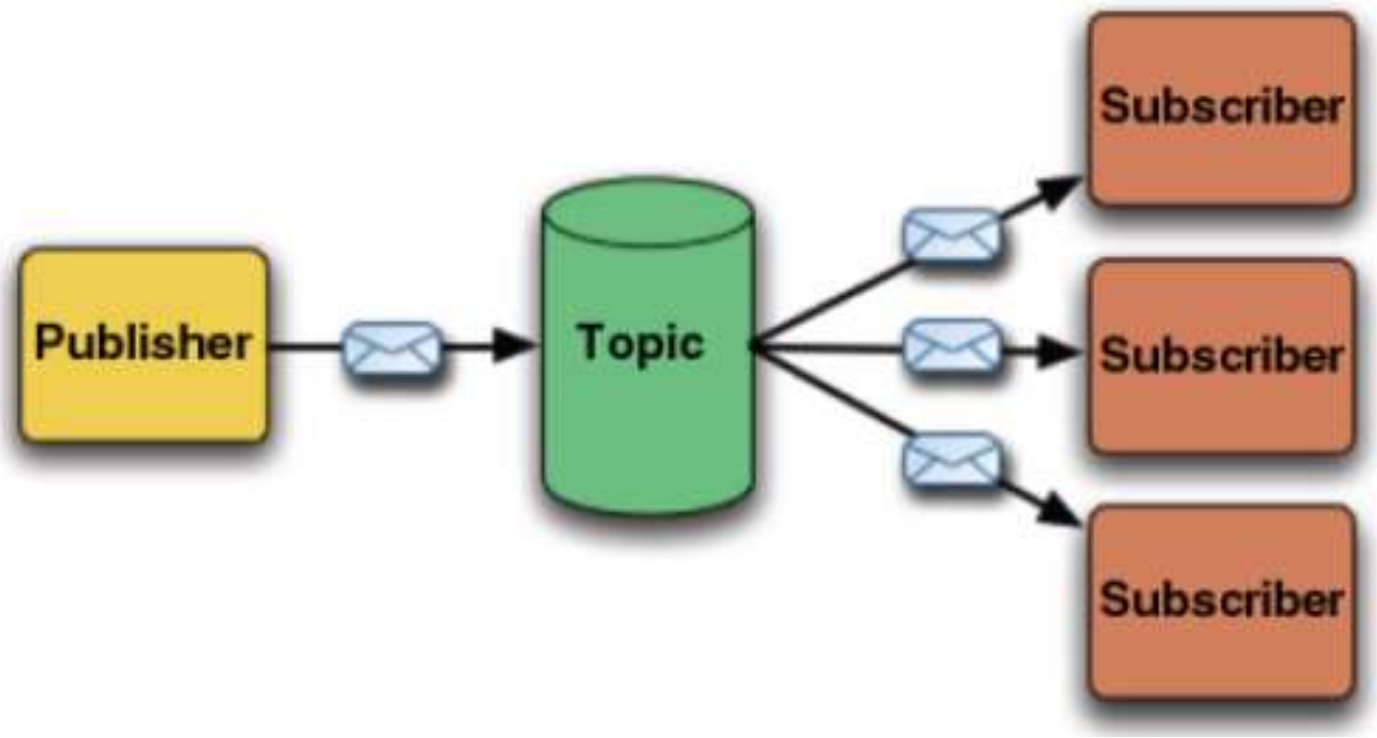
Let’s recall basic concepts of
Messaging System

Point to Point Messaging (Queue)



Credit: http://fusesource.com/docs/broker/5.3/getting_started/FuseMBStartedKeyJMS.html

Publish-Subscribe Messaging (Topic)



Credit: http://fusesource.com/docs/broker/5.3/getting_started/FuseMBStartedKeyJMS.html

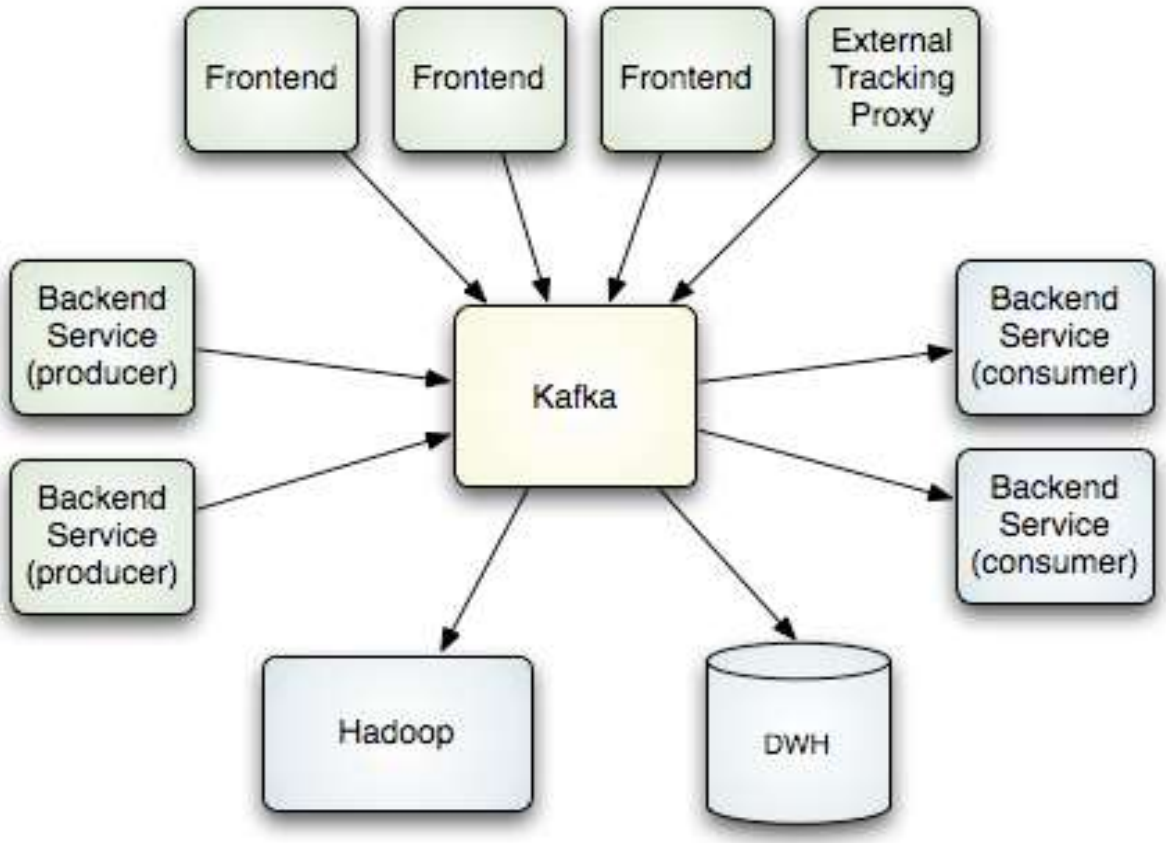
Apache Kafka

Overview

- An apache project initially developed at LinkedIn
- Distributed publish-subscribe messaging system
- Designed for processing of real time activity stream data e.g. logs, metrics collections
- Written in Scala
- Does not follow JMS Standards, neither uses JMS APIs
- Features
  - Persistent messaging
  - High-throughput
  - Supports both queue and topic semantics
  - Uses Zookeeper for forming a cluster of nodes (producer/consumer/broker)

  and many more…
- http://kafka.apache.org/

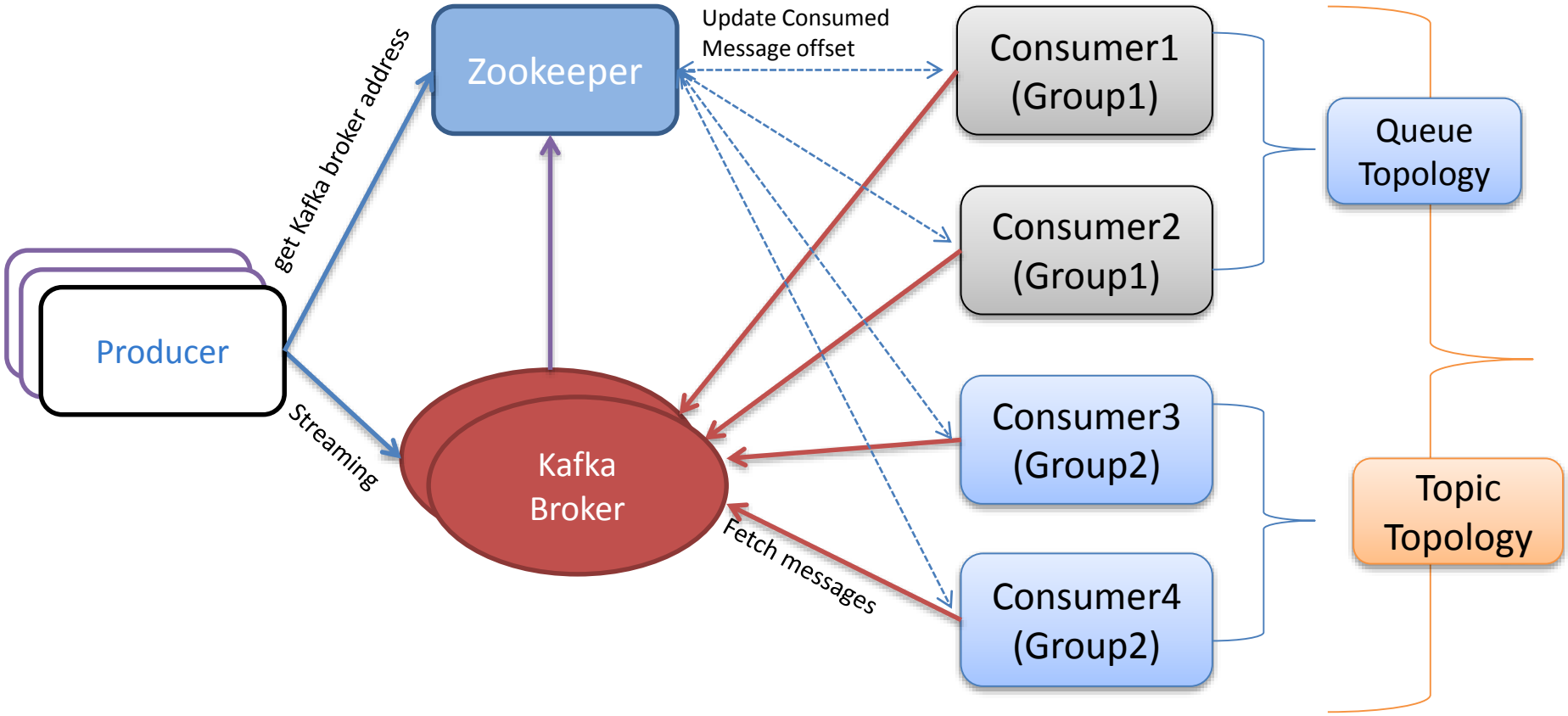
How it works



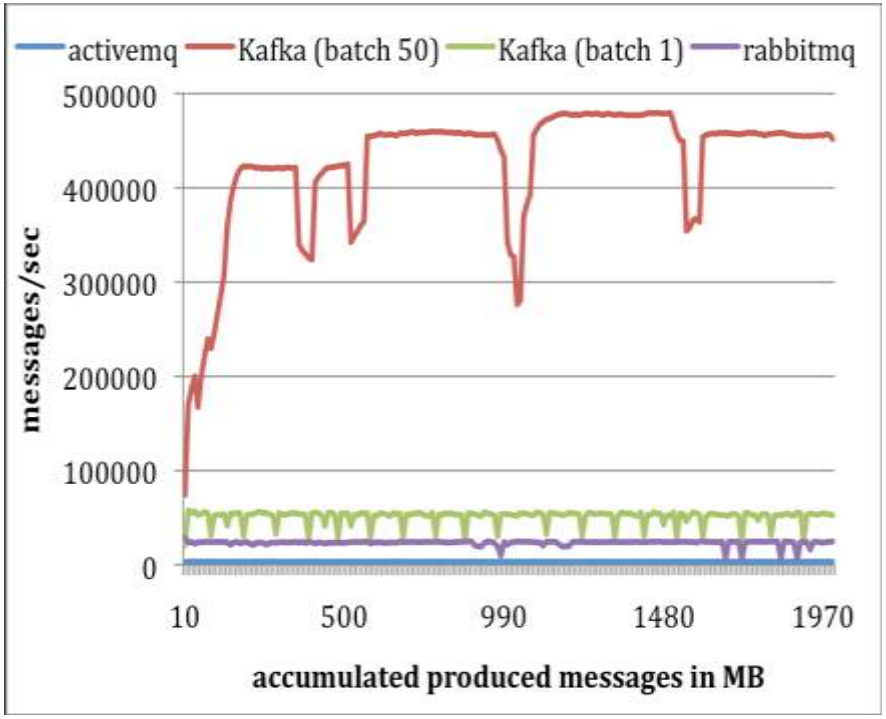
Credit : http://kafka.apache.org/design.html

Real time transfer

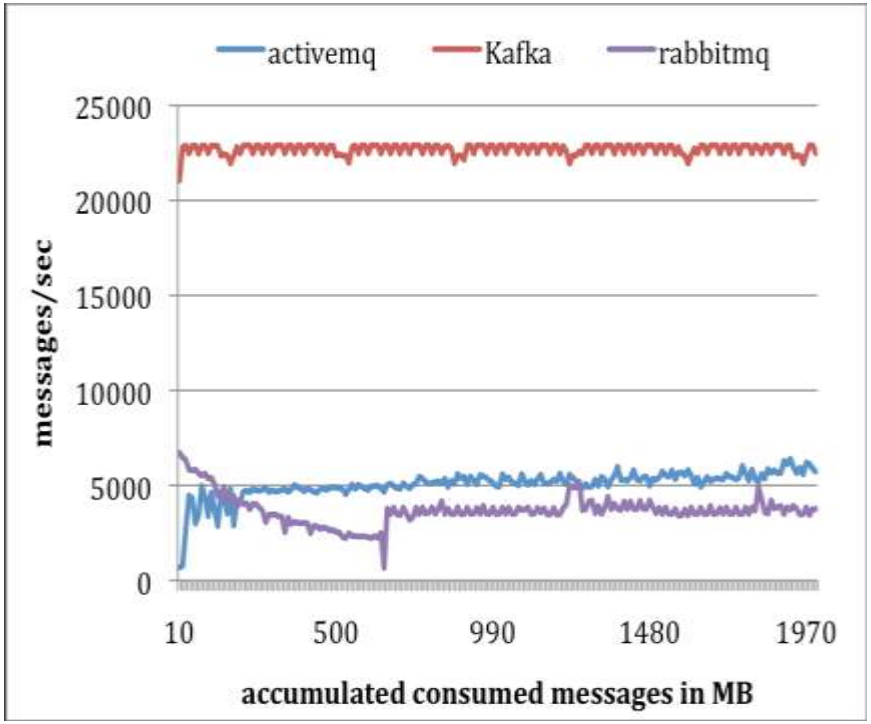
Broker does not ***Push*** messages to Consumer, Consumer ***Polls*** messages from Broker.



Performance Numbers



Producer Performance



Consumer Performance

Credit : http://research.microsoft.com/en-us/UM/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf

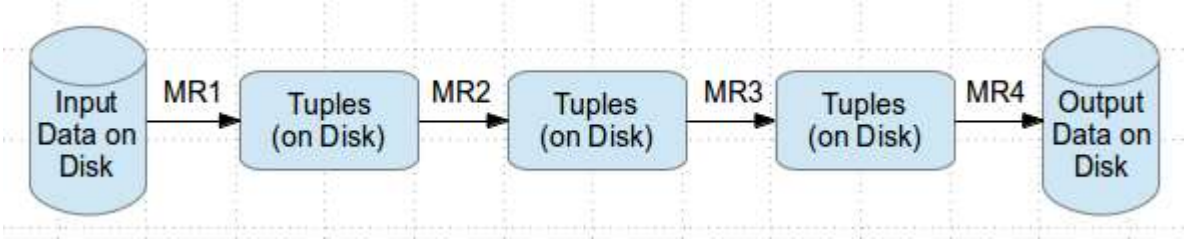
About Apache Spark

- 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/

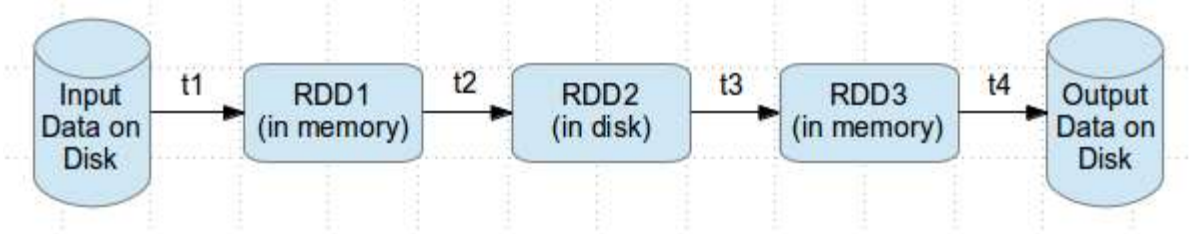
So Why Spark ?

- 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



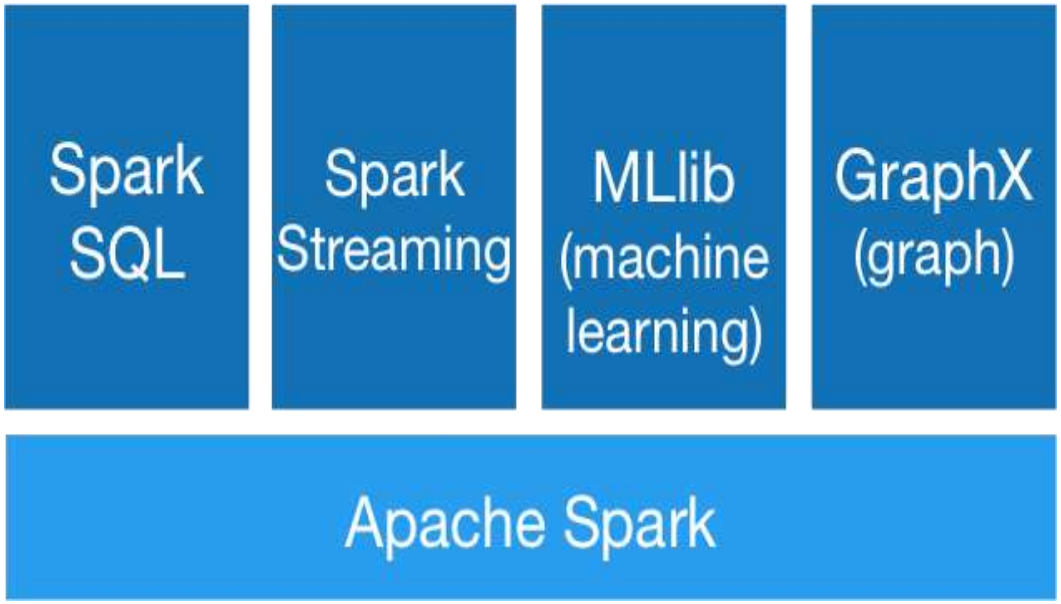
Spark execution flow



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

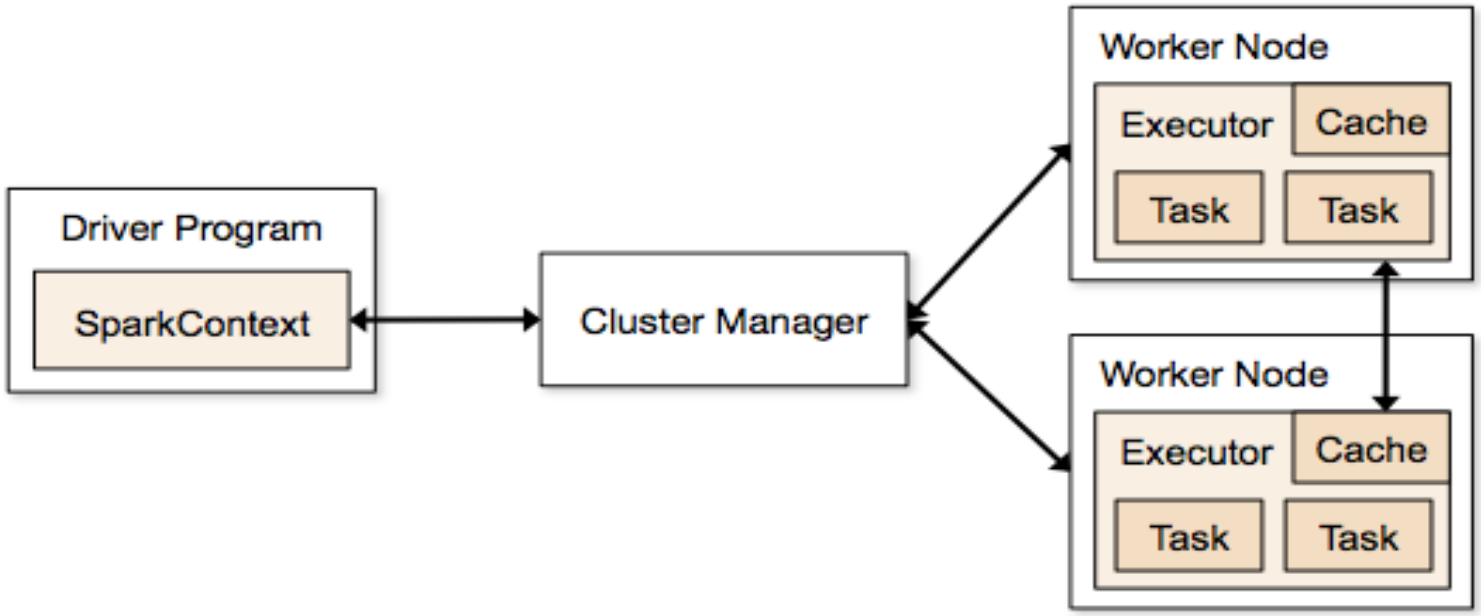
Spark Stack

- 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.

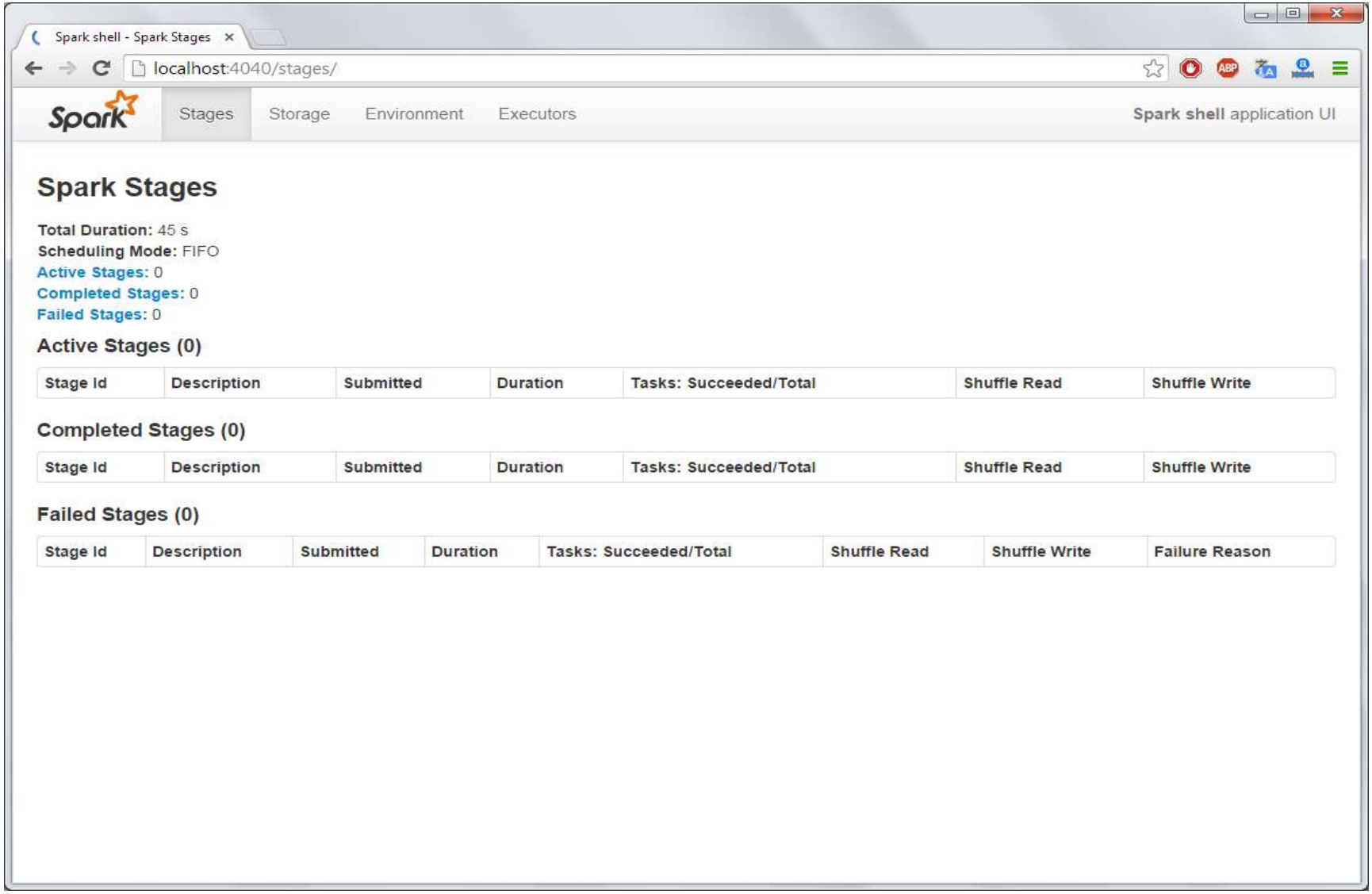
Resilient Distributed Dataset (RDD)

- 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)

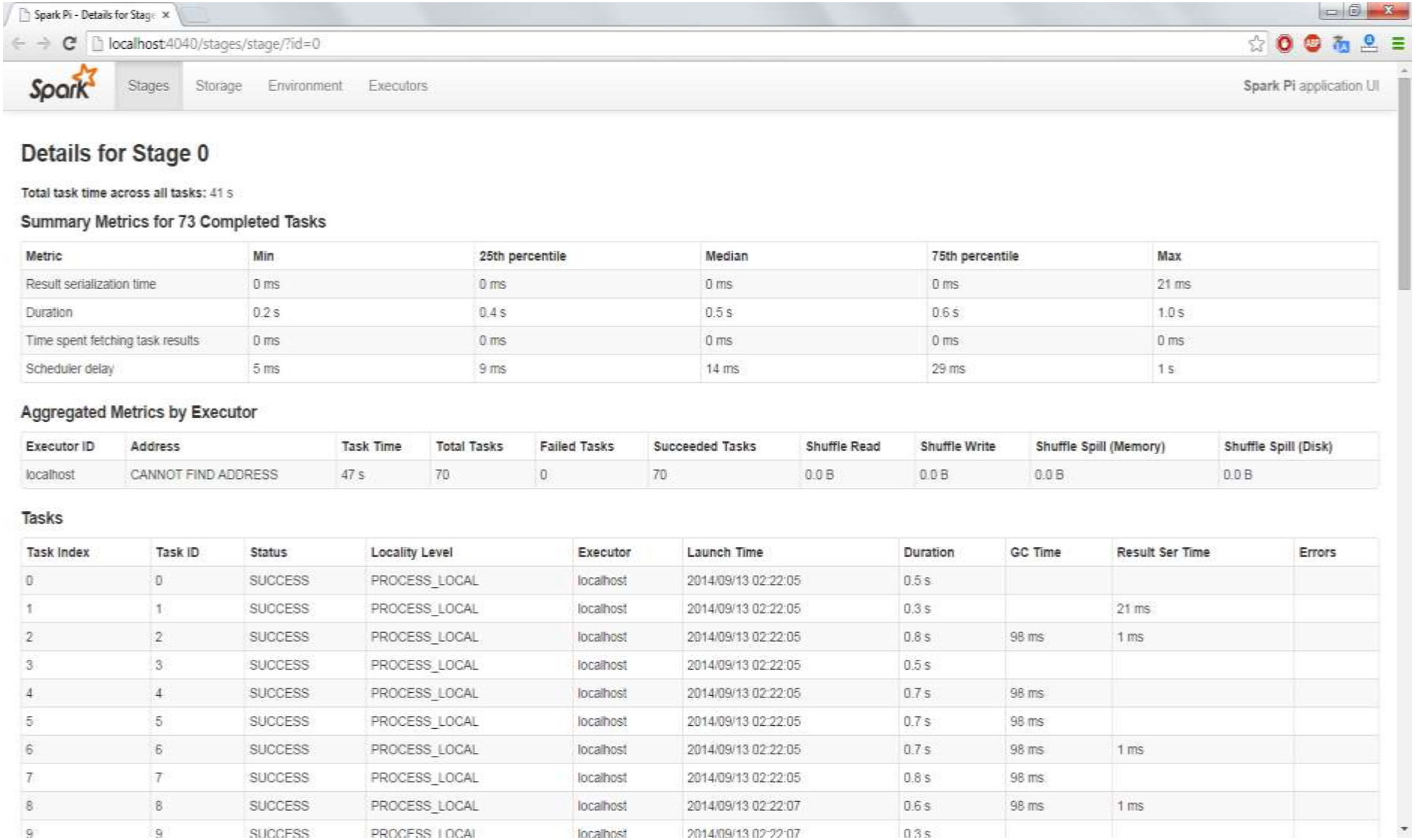
Cluster Deployment

- 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 shell - Spark Stages
localhost:4040/stages/
Spark
Stages
Storage
Environment
Executors
Spark shell application UI
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



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 start getting hands dirty

Kafka Installation

- Download
  - http://kafka.apache.org/downloads.html

- Untar it

```
> tar -xzf kafka_<version>.tgz
> cd kafka_<version>
```

Start Servers

- Start the Zookeeper server

  > bin/zookeeper-server-start.sh config/zookeeper.properties

Pre-requisite: Zookeeper should be up and running.

- Now Start the Kafka Server

  > bin/kafka-server-start.sh config/server.properties

Create/List Topics

- Create a topic

  > bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test

- List down all topics

  > bin/kafka-topics.sh --list --zookeeper localhost:2181

  Output: test

Producer

- Send some Messages

  > bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

  ```
  Now type on console:
  This is a message
  This is another message
  ```

Consumer

- Receive some Messages

  > bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test --from-beginning

  This is a message

  This is another message

Multi-Broker Cluster

- Copy configs

```
> cp config/server.properties config/server-1.properties
> cp config/server.properties config/server-2.properties
```

- Changes in the config files.

config/server-1.properties:

broker.id=1
port=9093
log.dir=/tmp/kafka-logs-1

config/server-2.properties:

broker.id=2
port=9094
log.dir=/tmp/kafka-logs-2

Start with New Nodes

- Start other Nodes with new configs

```
> bin/kafka-server-start.sh config/server-1.properties &
> bin/kafka-server-start.sh config/server-2.properties &
```

- Create a new topic with replication factor as 3

```
> bin/kafka-topics.sh --create --zookeeper
localhost:2181 --replication-factor 3 --partitions
1 --topic my-replicated-topic
```

- List down the all topics

```
> bin/kafka-topics.sh --describe --zookeeper
localhost:2181 --topic my-replicated-topic
```

Topic:my-replicated-topic PartitionCount:1 ReplicationFactor:3 Configs:

Topic: my-replicated-topic Partition: 0 Leader: 1 Replicas: 1,2,0 Isr: 1,2,0

Let’s move to Apache Spark

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);
  }
}
```

Job Submission

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

Configuration

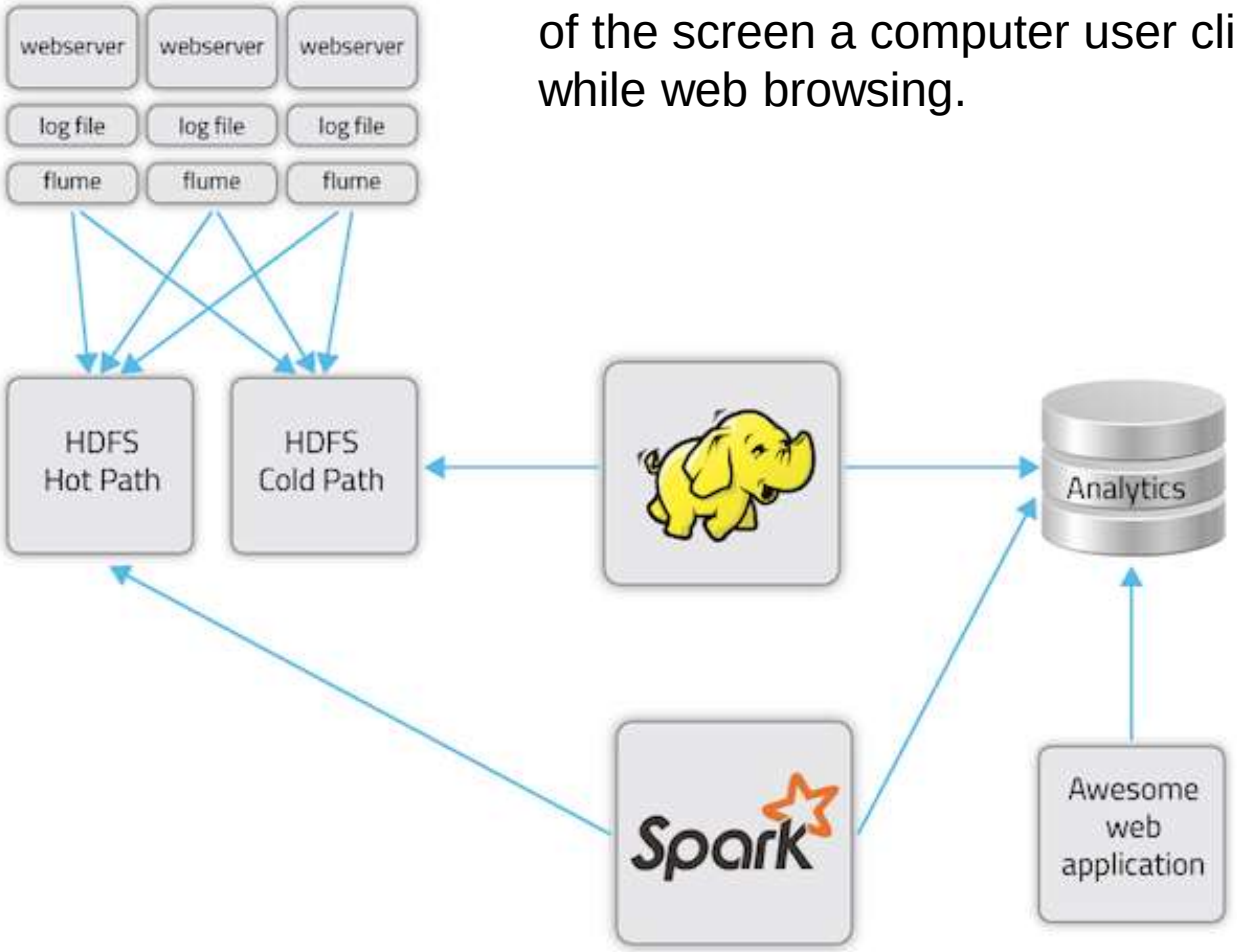
```
val conf = new SparkConf()
      .setMaster("local")
      .setAppName("CountingSheep")
      .set("spark.executor.memory", "1g")

val sc = new SparkContext(conf)
```

Let’s discuss

Real-time Clickstream Analytics

Analytics High Level Flow

A **clickstream** is the recording of the parts of the screen a computer user clicks on while web browsing.



http://blog.cloudera.com/blog/2014/03/letting-it-flow-with-spark-streaming/

Spark Streaming

Makes it easy to build scalable fault-tolerant streaming applications.

- Ease of Use
- Fault Tolerance
- Combine streaming with batch and interactive queries.

Support Multiple Input/Outputs

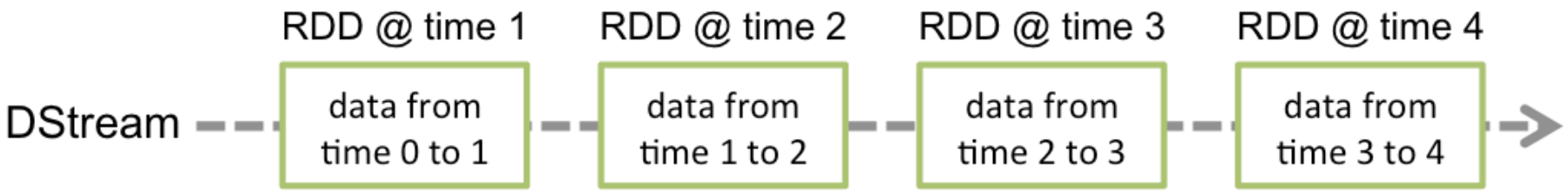
Credit : http://spark.apache.org

Streaming Flow

Credit : http://spark.apache.org

Spark Streaming Terminology

- Spark Context
  - An object with configuration parameters
- Discretized Steams (DStreams)
  - Stream from input sources or generated after transformation
  - Continuous series of RDDs
- Input DSteams
  - Stream of raw data from input sources



Spark Context

```
SparkConf sparkConf = new SparkConf().setAppName(”PageViewsCount");
sparkConf.setMaster("local[2]");//running locally with 2 threads

// Create the context with a 5 second batch size
JavaStreamingContext jssc = new JavaStreamingContext(sparkConf, new Duration(5000));
```

Reading from Kafka Stream

```
 int numThreads = Integer.parseInt(args[3]);
Map<String, Integer> topicMap = new HashMap<String, Integer>();
String[] topics = args[2].split(",");
for (String topic: topics) {
  topicMap.put(topic, numThreads);
}
JavaPairReceiverInputDStream<String, String> messages =
        KafkaUtils.createStream(jssc, <zkQuorum>, <group> , topicMap);
```

Processing of Stream

```
/**
 *  Incoming:
   // (192.168.2.82,1412977327392,www.example.com,192.168.2.82)
   // 192.168.2.82: key : tuple2._1()
   // 1412977327392,www.example.com,192.168.2.82: value: tuple2._2()
*/
// Method signature: messages.map(InputArgs, OutputArgs)
JavaDStream<Tuple2<String, String>> events = messages.map(new Function<Tuple2<String, String>,
Tuple2<String, String>>() {
    @Override
    public Tuple2<String, String> call(Tuple2<String, String> tuple2) {
     String[] parts = tuple2._2().split(",");
     return new Tuple2<>(parts[2], parts[1]);
    }
 });
```

```
JavaPairDStream<String, Long> pageCountsByUrl = events.map(new Function<Tuple2<String,String>, String>(){

 @Override
  public String call(Tuple2<String, String> pageView) throws Exception {
              return pageView._2();
     }
  }).countByValue();

 System.out.println("Page counts by url:");

 pageCountsByUrl.print();
```

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/