

Testing Spark: Best Practices

Anupama Shetty

Senior SDET, Analytics, Ooyala Inc

Neil Marshall

SDET, Analytics, Ooyala Inc

Spark Summit 2014



Agenda - Anu

1. Application Overview

- Batch mode
- Streaming mode with kafka

2. Test Overview

- Test environment setup
- Unit testing spark applications
- Integration testing spark applications

3. Best Practices

- Code coverage support with scoverage and scct
- Auto build trigger using jenkins hook via github

Agenda - Neil

4. Performance testing of Spark

- Architecture & technology overview
- Performance testing setup & run
- Result analysis
- Best practices

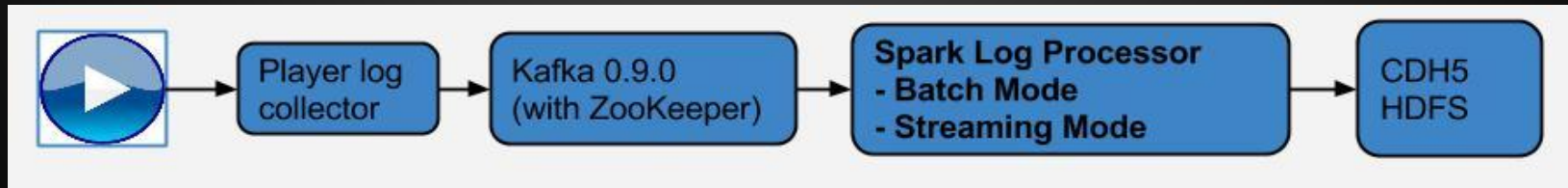
Company Overview



- Founded in 2007
- 300+ employees worldwide
- Global footprint of 200M unique users in 130 countries
- Ooyala works with the most successful broadcasts and media companies in the world
- Reach, measure, monetize video business
- Cross-device video analytics and monetization products and services

Application Overview

- Analytics ETL pipeline service
- Receives 5B+ player generated events such as plays, displays on a daily basis.
- Computed metrics include player conversion rate, video conversion rate and engagement metrics.
- Third party services used are
 - Spark 1.0 used to process player generated big data.
 - Kafka 0.9.0 with Zookeeper as our message queue
 - CDH5 HDFS as our intermediate storage file system

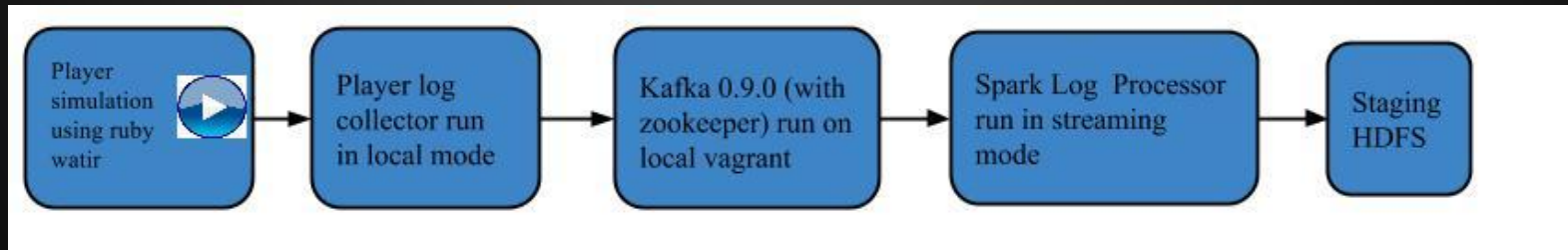


Spark based Log Processor details

- Supports two input data formats
 - Json
 - Thrift
- Batch Mode Support
 - Uses Spark Context
 - Consumes input data via a text file
- Streaming Mode Support
 - Uses Spark streaming context
 - Consumes data via kafka stream

Test pipeline setup

- Player simulation done using [Watir](#) (ruby gem based on Selenium).
- Kafka(with zookeeper) setup as local virtual machine using [vagrant](#). VMs can be monitored using [VirtualBox](#).
- Spark cluster run in local mode.



Unit test setup - Spark in Batch mode

- Spark cluster setup for testing
 - Build your spark application jar using `sbt "assembly"`
 - Create config with spark.jar set to application jar and spark.master to "local"
 - `var config = ConfigFactory parseString """"spark.jar = "target/scala-2.10/SparkLogProcessor.jar",spark.master = "local" """"`
 - Store local spark directory path for spark context creation
 - `val sparkDir = <path to local spark directory> + "spark-0.9.0-incubating-bin-hadoop2/assembly/target/scala-2.10/spark-assembly_2.10-0.9.0-incubating-hadoop2.2.0.jar").mkString`
- Creating spark context
 - `var sc: SparkContext = new SparkContext("local", getClass.getSimpleName, sparkDir, List(config.getString("spark.jar")))`

Test Setup for batch mode using Spark Context

Before block

```
before{  
  // To avoid Akka rebinding to the same port, since it doesn't unbind immediately on shutdown  
  System.clearProperty("spark.driver.port")  
  System.clearProperty("spark.hostPort")  
  
  // Create a Spark context  
  sc = new SparkContext("local", getClass.getSimpleName, sparkDir, List(config.getString("spark.jar")))  
}
```

After block

```
after {  
  // Cleanup spark context data  
  sc.stop()  
  sc = null  
  System.clearProperty("spark.driver.port")  
  System.clearProperty("spark.hostPort")  
}
```

Scala test framework “FunSpec” is used with “ShouldMatchers” (for assertions) and “BeforeAndAfter” (for setup/teardown).

Kafka setup for spark streaming

- Bring up Kafka virtual machine using [Vagrantfile](#) with following command
`vagrant up kafkavm`

```
Vagrant::Config.run do |conf|
  conf.vm.define :kafkavm do |box|
    box.vm.box = "kafka"
    box.vm.host_name = "kafkavm"
    box.vm.network :hostonly, "1.2.3.15"
    box.vm.forward_port 22, 2335 #ssh
    box.vm.forward_port 2181, 2181 #zk
    box.vm.forward_port 9092, 9092 #kafka
    box.vm.customize [ "modifyvm", :id, "--name", "kafkavm", "--memory", "4096", ]
  end
end
```

- Configure [Kafka](#)
 - Create topic
 - `bin/kafka-create-topic.sh --zookeeper "localhost:2181" --topic "thrift_pings"``
 - Consume messages using
 - `bin/kafka-console-consumer.sh --zookeeper "localhost:2181" --topic "thrift_pings" --group "testThrift" &>/tmp/thrift-consumer-msgs.log &`

```

val sparkKafkaDir = <path to directory> + "spark-0.9.0-incubating-bin-hadoop2/external/" +
  |"kafka/target/spark-streaming-kafka_2.10-0.9.0-incubating.jar").mkString

val sparkMaster = config.getString("spark.master")
var sc: SparkContext = _
var ssc: StreamingContext = _
var bytes: RDD[Array[Byte]] = _
val jars = List(config.getString("spark.jar"))

before {
  // To avoid Akka rebinding to the same port, since it doesn't unbind immediately on shutdown
  System.clearProperty("spark.driver.port")
  System.clearProperty("spark.hostPort")
  System.setProperty("spark.cleaner.ttl", "300")

  sc = new SparkContext(sparkMaster, "thrift-spark-streaming", sparkDir, jars ++ List(sparkKafkaDir))

  // Create a streaming context
  ssc = new StreamingContext(sc, Seconds(60))

  val metrics = LogProcessorMetrics.getAccumulators(sc)(config)

  val kafkaInputs = (1 to 2).map { _ =>
    KafkaUtils.createStream(ssc, "localhost", "testingThrift", Map("thrift_pings" -> 1), StorageLevel.
      MEMORY_ONLY).map( x => x._2.getBytes)
  }

  val bytessc = ssc.union(kafkaInputs)

  bytessc.foreach{ bytes =>
    // Application specific
    LogProcessor.process(bytes,sc,config,metrics)
  }

  ssc.start()
  ssc.awaitTermination()
}

```

Testing streaming mode with Spark Streaming Context

Test 'After' block and assertion block for spark streaming mode

After Block

```
after {  
    // Cleanup spark context data  
    sc.stop()  
    sc = null  
  
    ssc.stop()  
    ssc = null  
  
    System.clearProperty("spark.driver.port")  
    System.clearProperty("spark.hostPort")  
}
```

Test Assertion

```
describe("Verify SparkStreaming"){  
    it("should verify session output post process"){  
        // Verify creation of a spark context  
        sc instanceof [SparkContext] should be(true)  
        ssc instanceof [StreamingContext] should be(true)  
    }  
}
```

Testing best practices - Code Coverage

- Tracking code coverage with [Scoverage](#) and/or [Scct](#)
- Enable fork = true to avoid spark exceptions caused by spark context conflicts.
- SCCT configurations
 - ScctPlugin.instrumentSettings
 - parallelExecution in ScctTest := false
 - fork in ScctTest := true
 - Command to run it - `sbt "scct:test"`
- Scoverage configurations
 - ScoverageSbtPlugin.instrumentSettings
 - ScoverageSbtPlugin.ScoverageKeys.excludedPackages in
ScoverageSbtPlugin.scoverage := ".*benchmark.*;.*util.*"
 - parallelExecution in ScoverageSbtPlugin.scoverageTest := false
 - fork in ScoverageSbtPlugin.scoverageTest := true
 - Command to run it - `sbt "scoverage:test"`

Testing best practices - Jenkins auto test build trigger

- Requires enabling 'github-webhook' on github repo settings page. Requires admin access for the repo.
- Jenkins job should be configured with corresponding github repo via “GitHub Project” field.
- Test jenkins hook by triggering a test run from github repo.
- "Github pull request builder" can be used while configuring jenkins job to auto publish test results on github pull requests after every test run. This also lets you rerun failed tests via github pull request.

What is a performance testing?

- A practice striving to build performance into the implementation, design and architecture of a system.
- Determine how a system performs in terms of responsiveness and stability under a particular workload.
- Can serve to investigate, measure, validate or verify other quality attributes of a system, such as scalability, reliability and resource usage.

What is a Gatling?

- Stress test tool



Why is Gatling selected over other Perf Test tools as JMeter?

- Powerful scripting using Scala
- Akka + Netty
- Run multiple scenarios in one simulation
- Scenarios = code + DSL
- Graphical reports with clear & concise graphs

How does Gatling work with Spark

- Access Web applications / services



Develop & setup a simple perf test example

A perf test will run against spark-jobserver for word counts.



What is a spark jobserver?

- Provides a RESTful interface for submitting and managing Apache Spark jobs, jars and job contexts
- Scala 2.10 + CDH5/Hadoop 2.2 + Spark 0.9.0
- For more depths on jobserver, see Evan Chan & Kelvin Chu's [Spark Query Service](#) presentation.

Steps to set up & run Spark-jobserver

- Clone spark-jobserver from git-hub

```
$ git clone https://github.com/ooyala/spark-jobserver
```

- Install SBT and type “sbt” in the spark-jobserver repo

```
$ sbt
```

- From SBT shell, simply type “re-start”

```
> re-start
```

Steps to package & upload a jar to the jobserver

- Package the test jar of the word count example

```
$ sbt job-server-tests/package
```

- Upload the jar to the jobserver

```
$ curl --data-binary @job-server-tests/target/job-server-tests-0.3.1.jar  
localhost:8090/jars/test
```

Run a request against the jobserver

```
$ curl -d "input.string = a b c a b see" 'http://localhost:8090/jobs?
appName=test&classPath=spark.jobserver.WordCountExample&sync=true'
{
  "status": "OK",
  "result": {
    "a": 2,
    "b": 2,
    "c": 1,
    "see": 1
  }
}
```

Source code of Word Count Example

```
object WordCountExample extends SparkJob {  
  def main(args: Array[String]) {  
    val sc = new SparkContext("local[4]", "WordCountExample")  
    val config = ConfigFactory.parseString("")  
    val results = runJob(sc, config)  
    println("Result is " + results)  
  }  
  
  override def validate(sc: SparkContext, config: Config): SparkJobValidation = {  
    Try(config.getString("input.string"))  
      .map(x => SparkJobValid)  
      .getOrElse(SparkJobInvalid("No input.string config param"))  
  }  
  
  override def runJob(sc: SparkContext, config: Config): Any = {  
    val dd = sc.parallelize(config.getString("input.string").split(" ").toSeq)  
    dd.map(_._1).reduceByKey(_ + _).collect().toMap  
  }  
}
```

Script Gatling for the Word Count Example

Scenario defines steps that Gatling does during a runtime:

```
val scn = scenario("WordCount Example Test")
  .repeat(2) {
    exec(http("Post a word string")
      .post("/jobs?appName=test&classPath=spark.jobserver.WordCountExample&sync=true")
      .body(StringBody("input.string = a b c a b see"))
      .header("Content-Type", "application/text")
      .check(status.is(200)))
  }
```

Script Gatling for the Word Count Example

Setup puts users and scenarios as workflows plus assertions together in a performance test simulation

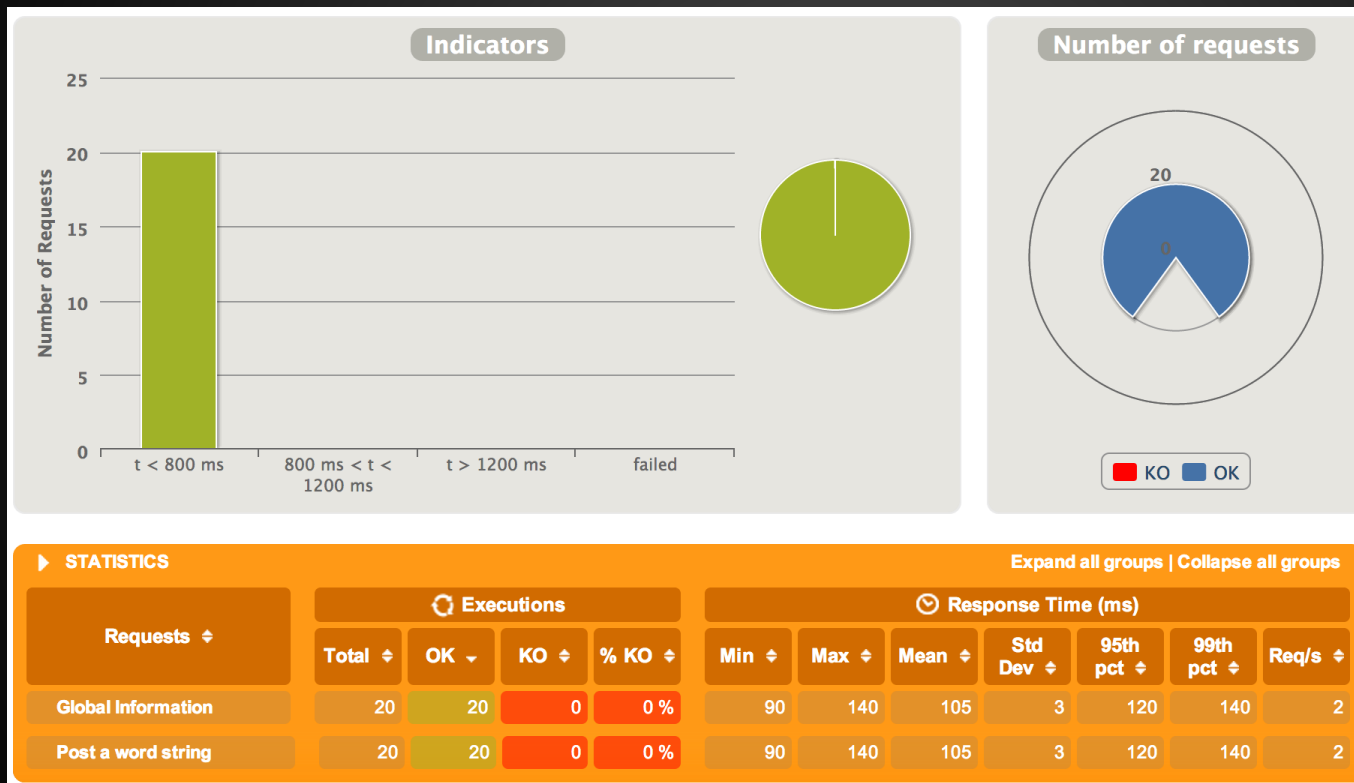
- Inject 10 users in 10 seconds into scenarios in 2 cycles
- Ensure successful requests greater than 80%

```
setUp(scen.inject(ramp(10 users) over (10 seconds)))  
  .protocols(httpProtocol)  
  .assertions(  
    global.successfulRequests.percent.greaterThan(80)  
  )
```

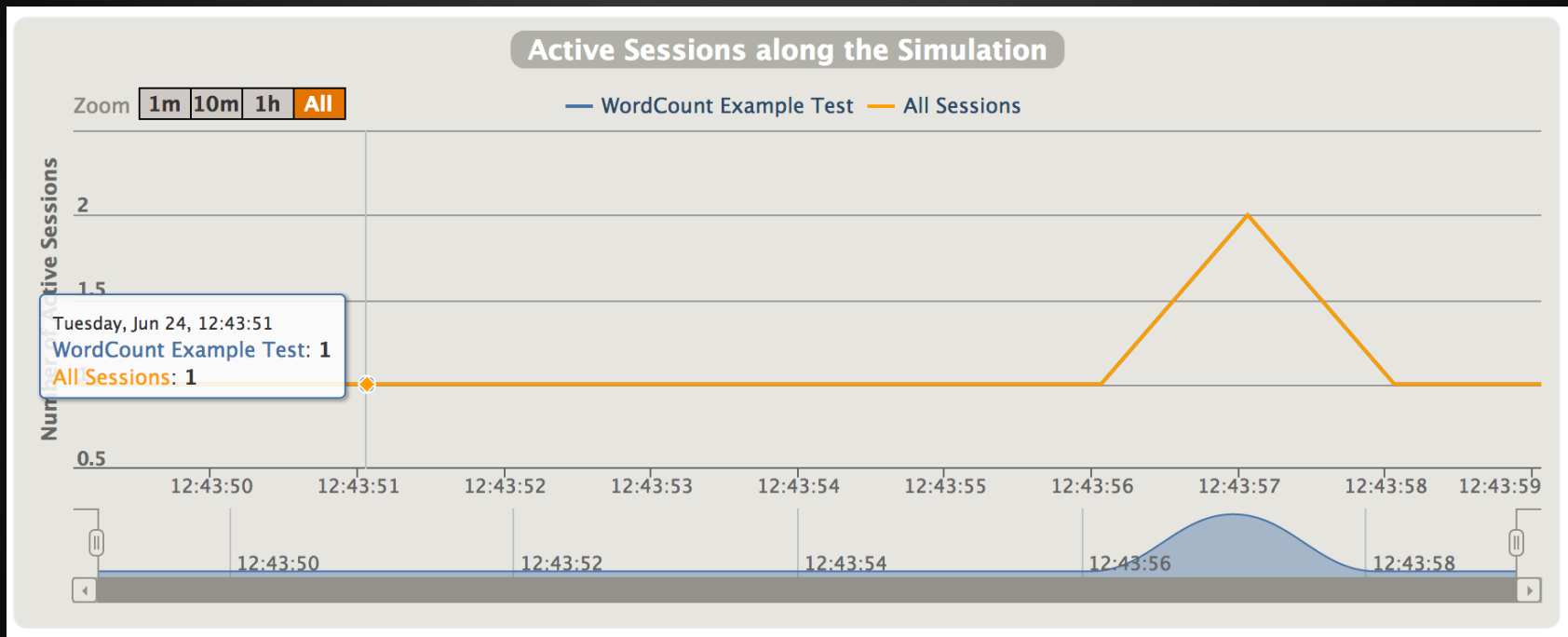
Test Results in Terminal Window

```
=====
---- Global Information -----
> numberOfRequests                20 (OK=20      K0=0      )
> minResponseTime                 90 (OK=90      K0=-      )
> maxResponseTime                 140 (OK=140     K0=-      )
> meanResponseTime                105 (OK=105     K0=-      )
> stdDeviation                    3  (OK=3       K0=-      )
> percentiles1                   120 (OK=120     K0=-      )
> percentiles2                   140 (OK=140     K0=-      )
> meanNumberOfRequestsPerSecond   2  (OK=2      K0=-      )
---- Response Time Distribution -----
> t < 800 ms                      20 (100%)
> 800 ms < t < 1200 ms            0 ( 0%)
> t > 1200 ms                     0 ( 0%)
> failed                          0 ( 0%)
=====
```

Gatling Graph - Indicator



Gatling Graph - Active Sessions



Best Practices on Performance Tests

- Run performance tests on Jenkins
- Set up baselines for any of performance tests with different scenarios & users

Any Questions?

References

Contact Info:

Anupama Shetty: anupama@ooyala.com

Neil Marshall: nmarshall@ooyala.com

References:

<http://www.slideshare.net/AnuShetty/spark-summit2014-techtalk-testing-spark>