CAR CRASH ANALYSIS USING SPARK

Introduction

My goal was to get the total count in states when it was a weekend (Saturday, Sunday), accident was fatal, driver was a teen, the driver was drunk from the data using Scala.

Data Source:

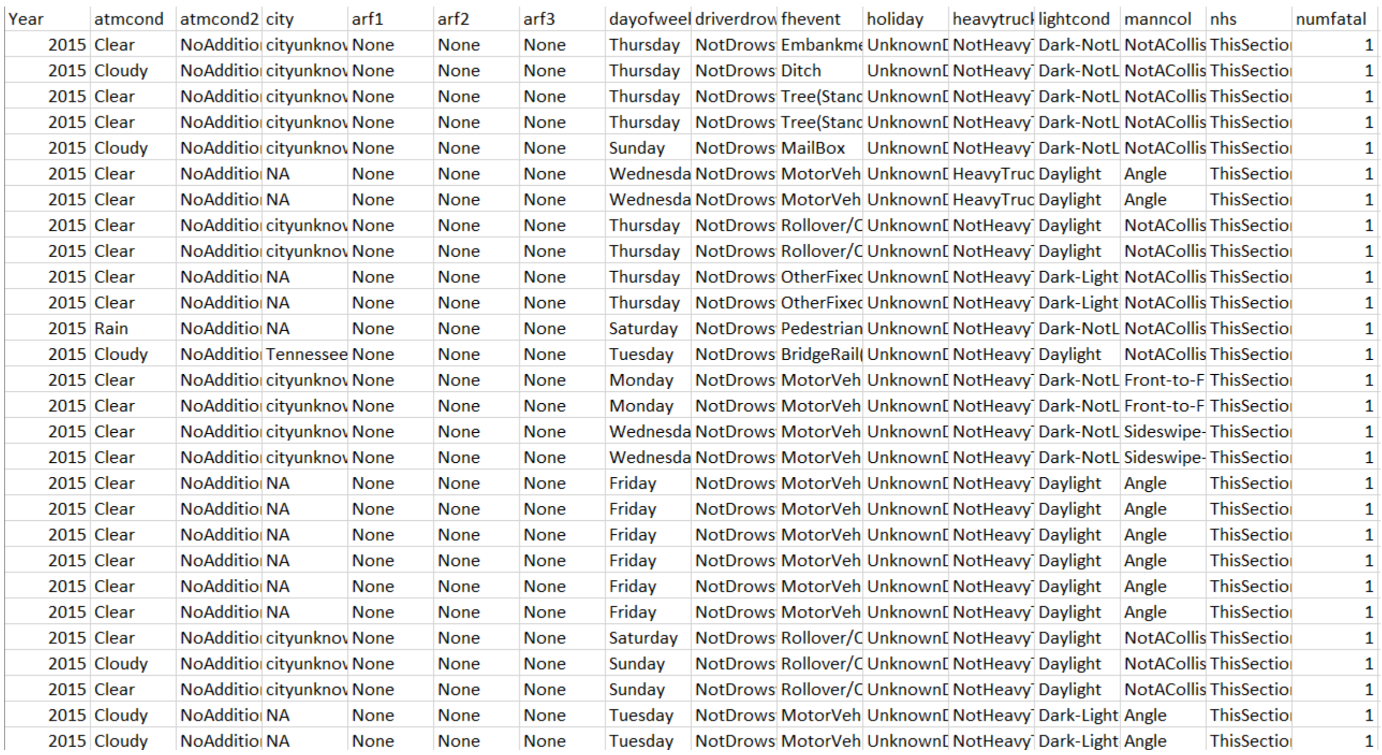
Car crash analysis data was found on https://www-fars.nhtsa.dot.gov//QueryTool/QuerySection/SelectYear.aspx includes data from year 2011 to 2015.

Data source description:

This data source is a repository of car crash analysis for each year from 1994 to 2016. I chose to work from year 2011-2015. The data is in csv format with 195 attributes.

Data description and schema:

The dataset includes close to 80,000 rows of records for each year with as many as 195 attributes, I removed the unused attributes and the count in the final is 116. Dataset looks like:



Year	atmcond	atmcond2	city	arf1	arf2	arf3	dayofweel	driverdrow	fhevent	holiday	heavytruck	lightcond	manncol	nhs	numfatal
2015	Clear	NoAdditio	cityunknov	None	None	None	Thursday	NotDrows	Embankme	UnknownD	NotHeavy	Dark-NotL	NotACollis	ThisSectio	1
2015	Cloudy	NoAdditio	cityunknov	None	None	None	Thursday	NotDrows	Ditch	UnknownD	NotHeavy	Dark-NotL	NotACollis	ThisSectio	1
2015	Clear	NoAdditio	cityunknov	None	None	None	Thursday	NotDrows	Tree(Stanc	UnknownD	NotHeavy	Dark-NotL	NotACollis	ThisSectio	1
2015	Clear	NoAdditio	cityunknov	None	None	None	Thursday	NotDrows	Tree(Stanc	UnknownD	NotHeavy	Dark-NotL	NotACollis	ThisSectio	1
2015	Cloudy	NoAdditio	cityunknov	None	None	None	Sunday	NotDrows	MailBox	UnknownD	NotHeavy	Dark-NotL	NotACollis	ThisSectio	1
2015	Clear	NoAdditio	NA	None	None	None	Wednesda	NotDrows	MotorVeh	UnknownD	HeavyTruc	Daylight	Angle	ThisSectio	1
2015	Clear	NoAdditio	NA	None	None	None	Wednesda	NotDrows	MotorVeh	UnknownD	HeavyTruc	Daylight	Angle	ThisSectio	1
2015	Clear	NoAdditio	cityunknov	None	None	None	Thursday	NotDrows	Rollover/C	UnknownD	NotHeavy	Daylight	NotACollis	ThisSectio	1
2015	Clear	NoAdditio	cityunknov	None	None	None	Thursday	NotDrows	Rollover/C	UnknownD	NotHeavy	Daylight	NotACollis	ThisSectio	1
2015	Clear	NoAdditio	NA	None	None	None	Thursday	NotDrows	OtherFixec	UnknownD	NotHeavy	Dark-Light	NotACollis	ThisSectio	1
2015	Clear	NoAdditio	NA	None	None	None	Thursday	NotDrows	OtherFixec	UnknownD	NotHeavy	Dark-Light	NotACollis	ThisSectio	1
2015	Rain	NoAdditio	NA	None	None	None	Saturday	NotDrows	Pedestrian	UnknownD	NotHeavy	Dark-NotL	NotACollis	ThisSectio	1
2015	Cloudy	NoAdditio	Tennessee	None	None	None	Tuesday	NotDrows	BridgeRail(	UnknownD	NotHeavy	Daylight	NotACollis	ThisSectio	1
2015	Clear	NoAdditio	cityunknov	None	None	None	Monday	NotDrows	MotorVeh	UnknownD	NotHeavy	Dark-NotL	Front-to-F	ThisSectio	1
2015	Clear	NoAdditio	cityunknov	None	None	None	Monday	NotDrows	MotorVeh	UnknownD	NotHeavy	Dark-NotL	Front-to-F	ThisSectio	1
2015	Clear	NoAdditio	cityunknov	None	None	None	Wednesda	NotDrows	MotorVeh	UnknownD	NotHeavy	Dark-NotL	Sideswipe-	ThisSectio	1
2015	Clear	NoAdditio	cityunknov	None	None	None	Wednesda	NotDrows	MotorVeh	UnknownD	NotHeavy	Dark-NotL	Sideswipe-	ThisSectio	1
2015	Clear	NoAdditio	NA	None	None	None	Friday	NotDrows	MotorVeh	UnknownD	NotHeavy	Daylight	Angle	ThisSectio	1
2015	Clear	NoAdditio	NA	None	None	None	Friday	NotDrows	MotorVeh	UnknownD	NotHeavy	Daylight	Angle	ThisSectio	1
2015	Clear	NoAdditio	NA	None	None	None	Friday	NotDrows	MotorVeh	UnknownD	NotHeavy	Daylight	Angle	ThisSectio	1
2015	Clear	NoAdditio	NA	None	None	None	Friday	NotDrows	MotorVeh	UnknownD	NotHeavy	Daylight	Angle	ThisSectio	1
2015	Clear	NoAdditio	NA	None	None	None	Friday	NotDrows	MotorVeh	UnknownD	NotHeavy	Daylight	Angle	ThisSectio	1
2015	Clear	NoAdditio	NA	None	None	None	Friday	NotDrows	MotorVeh	UnknownD	NotHeavy	Daylight	Angle	ThisSectio	1
2015	Clear	NoAdditio	cityunknov	None	None	None	Saturday	NotDrows	Rollover/C	UnknownD	NotHeavy	Daylight	NotACollis	ThisSectio	1
2015	Cloudy	NoAdditio	cityunknov	None	None	None	Sunday	NotDrows	Rollover/C	UnknownD	NotHeavy	Daylight	NotACollis	ThisSectio	1
2015	Clear	NoAdditio	cityunknov	None	None	None	Sunday	NotDrows	Rollover/C	UnknownD	NotHeavy	Daylight	NotACollis	ThisSectio	1
2015	Cloudy	NoAdditio	NA	None	None	None	Tuesday	NotDrows	MotorVeh	UnknownD	NotHeavy	Dark-Light	Angle	ThisSectio	1
2015	Cloudy	NoAdditio	NA	None	None	None	Tuesday	NotDrows	MotorVeh	UnknownD	NotHeavy	Dark-Light	Angle	ThisSectio	1

Schema:
This dataset includes some of the crashes related columns like
Year
Atmospheric condition
Atmospheric condition 2
City
School bus related
Day of the week
Speeding
Work zone
Light condition
Rail grade crossing identifier
Drowsy driver
State
Some of the columns include details of people involved in the accident. They are
Age
Sex or gender
License state
Drug test results
Drug test type
Injury severity
Police-reported alcohol involvement
Alcohol test type
Drug test results
Drug test type
Alcohol test results

Data Preprocessing:
Some of the fields included a comma (,) which was resulting in merging of two columns a pipe symbol replaced This comma to maintain consistency across data. Every column had zeros, ones and had metadata attached to it. I used Scala for replacing them with the original data so that it would be easy to analyze the results when compared to zeros and ones.

Data load:
The csv file was loaded into the local virtual machine.

Spark algorithm:

Step 1: Start spark in local mode.

Step 2: Read the csv file into a RDD called "input".

Step 3: split the file on comma (since it is a csv) and filter it on field 7(day of the week) looking for "Saturday, Sunday" and assign the results to weekend.

Step 4: now split and filter on field 24 (age) on weekend (previous results) filtering for values "less than 18years". Assign this value to age.

Step 5: split and filter in age on field 46(driver drunk) for values "yes" and assign results to drunk.

Step 6: split and filter in drunk on field 40(type of injury) for values "fatal" and assign the results to injury.

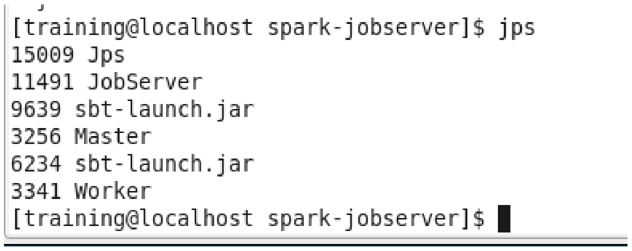
Step 7: split the injury on comma and we mapped row 89 which represents the state where accident happened and then performed simple map reduce operation to find the total count in each state.

Description of ecosystem or tools:

I have concentrated on performance testing and have found that it is feasible to use Gatling with Spark Job server to simulate a performance run for multiple requests at a given time or bursts of requests over a period. The inspiration for doing performance testing with Gatling on Spark Job server was taken from the presentation of "Anupama Shetty" from spark-summit 2014[1].

To perform these simulation runs, there is need to setup stand-alone spark instance on a virtual machine with a worker and master running.
Spark can be run in standalone mode by running the start-all.sh in sbin folder of the spark-1.6 and make sure the master and worker are up and running.
The algorithm is archived as a jar, it can be executed in standalone mode by running spark submit command to see if the standalone spark instance is installed right.

Example: spark-submit --class driverresult --master local --executor-memory 1G --num-executors 15 --executor-cores 11 /home/training/driveresult.jar file:///home/training/Desktop/Data.csv file:///home/training/Desktop/output.txt 1
The setup for jobserver involved in cloning of the open source Jobserver repo from git[2] and having a sbt package manager along with java 1.8 installed on the local machine.
I have used online tutorial[3] on setting up the job server, the basic commands used in setting up the job server includes.
Running sbt from the base folder of the job server and restarting the job server(re-start). Once these commands are executed, it can be validated to see if the job server is running fine by executing the jps command checking all the java processes running at that time.



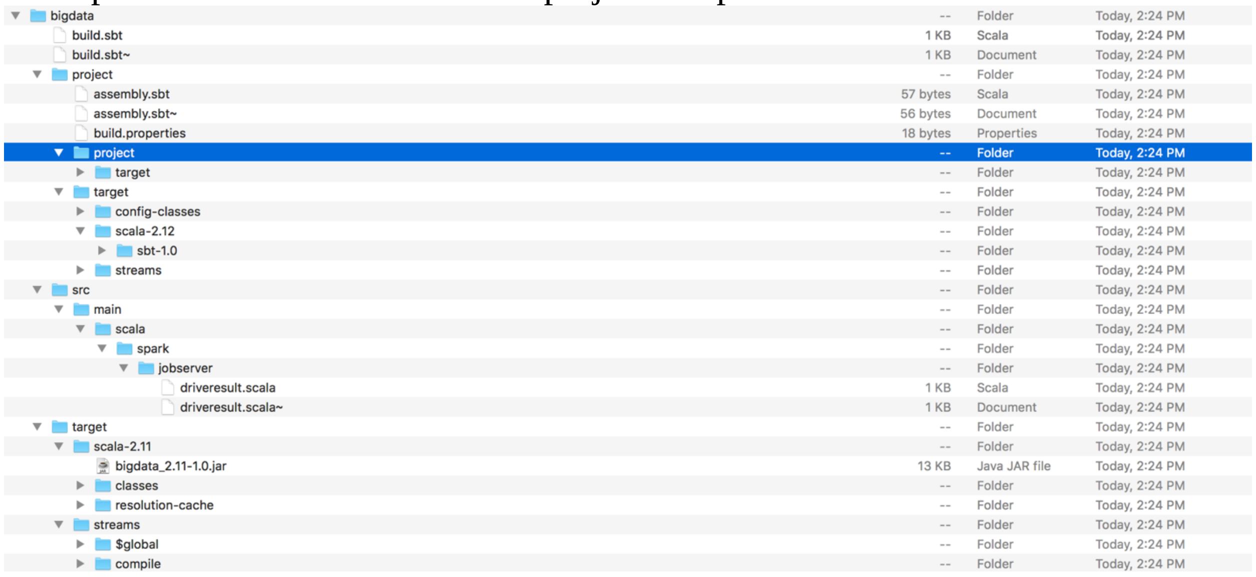
```
[training@localhost spark-jobserver]$ jps
15009 Jps
11491 JobServer
9639 sbt-launch.jar
3256 Master
6234 sbt-launch.jar
3341 Worker
[training@localhost spark-jobserver]$
```

Some sample programs can be run on the jobserver to test its functionality. These test jar are located under the spark-jobserver/job-server-tests/target/scala/scala-2.11/*.jar
Running jobserver involves in uploading the jar using the curl command and triggering the job from the rest call.
Example:
curl--data-binary @job-server-tests/target/scala-2.11/bigdata.jar localhost:8090/jars/test

curl -d "input.string=/home/training/Desktop/Data.csv" 'localhost:8090/jobs?appName=test&classPah=spark.jobserver.driveresult&timeout=6000'

Running a spark code on jobserver involves in extending the SparkJob class and overriding some of the functions in that class. In order to achieve this, a sbt project with a Scala src is created. And a fat jar is created on the successful compilation of the project.

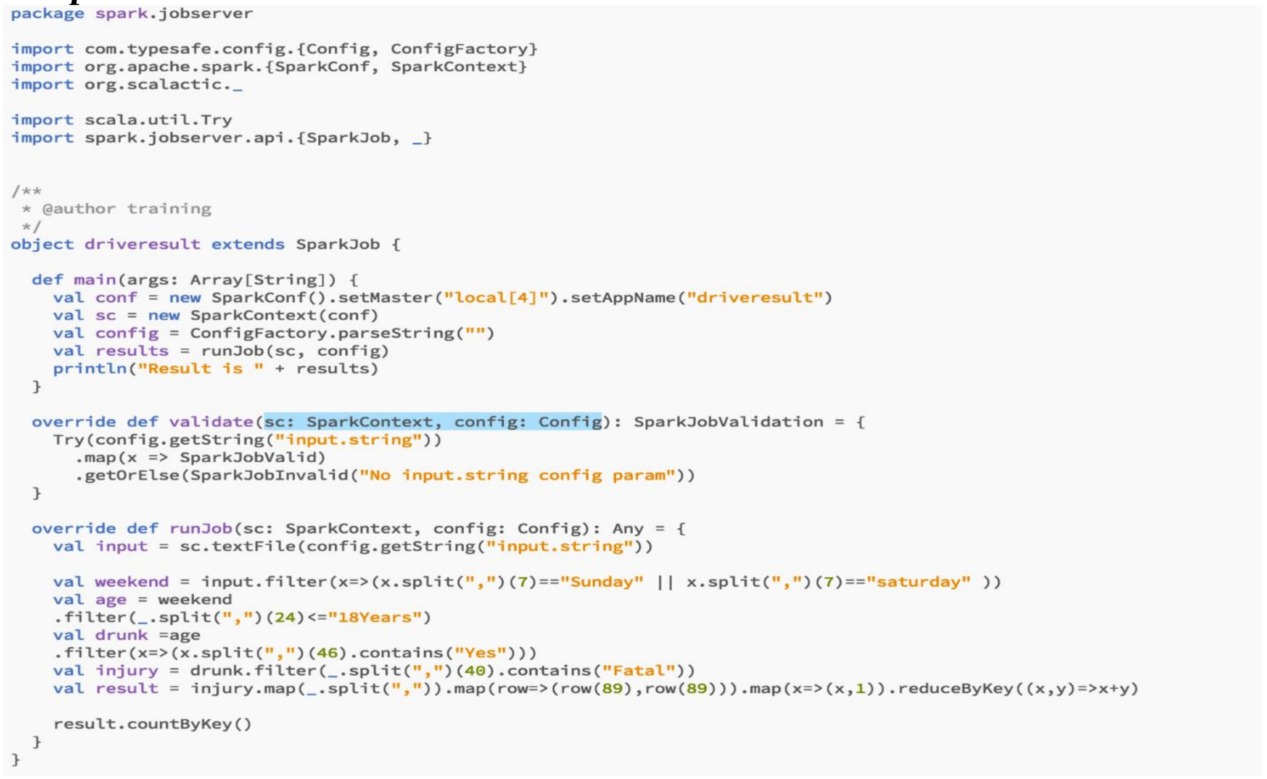
Example screen illustrates the scala project setup.



```
▼ bigdata                              --         Folder        Today, 2:24 PM
    build.sbt                          1 KB       Scala         Today, 2:24 PM
    build.sbt~                         1 KB       Document      Today, 2:24 PM
  ▼ project                            --         Folder        Today, 2:24 PM
      assembly.sbt                     57 bytes   Scala         Today, 2:24 PM
      assembly.sbt~                    56 bytes   Document      Today, 2:24 PM
      build.properties                 18 bytes   Properties    Today, 2:24 PM
    ▼ project                          --         Folder        Today, 2:24 PM
      ► target                         --         Folder        Today, 2:24 PM
    ▼ target                           --         Folder        Today, 2:24 PM
      ► config-classes                 --         Folder        Today, 2:24 PM
      ▼ scala-2.12                     --         Folder        Today, 2:24 PM
        ► sbt-1.0                      --         Folder        Today, 2:24 PM
      ► streams                        --         Folder        Today, 2:24 PM
  ▼ src                                --         Folder        Today, 2:24 PM
    ▼ main                             --         Folder        Today, 2:24 PM
      ▼ scala                          --         Folder        Today, 2:24 PM
        ▼ spark                        --         Folder        Today, 2:24 PM
          ▼ jobserver                  --         Folder        Today, 2:24 PM
              driveresult.scala        1 KB       Scala         Today, 2:24 PM
              driveresult.scala~       1 KB       Document      Today, 2:24 PM
  ▼ target                             --         Folder        Today, 2:24 PM
    ▼ scala-2.11                       --         Folder        Today, 2:24 PM
        bigdata_2.11-1.0.jar           13 KB      Java JAR file Today, 2:24 PM
      ► classes                        --         Folder        Today, 2:24 PM
      ► resolution-cache               --         Folder        Today, 2:24 PM
    ▼ streams                          --         Folder        Today, 2:24 PM
      ► $global                        --         Folder        Today, 2:24 PM
      ► compile                        --         Folder        Today, 2:24 PM
```

The SparkJob class mainly comprises of validate and run Job functions which can be override to fit custom functionality. Config object in the jobserver can used to pass any command line arguments or config properties to be sent over as post request in the job server call.

Sample code:



```scala
package spark.jobserver

import com.typesafe.config.{Config, ConfigFactory}
import org.apache.spark.{SparkConf, SparkContext}
import org.scalactic._

import scala.util.Try
import spark.jobserver.api.{SparkJob, _}


/**
 * @author training
 */
object driveresult extends SparkJob {

  def main(args: Array[String]) {
    val conf = new SparkConf().setMaster("local[4]").setAppName("driveresult")
    val sc = new SparkContext(conf)
    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 input = sc.textFile(config.getString("input.string"))

    val weekend = input.filter(x=>(x.split(",")(7)=="Sunday" || x.split(",")(7)=="saturday" ))
    val age = weekend
    .filter(_.split(",")(24)<="18Years")
    val drunk =age
    .filter(x=>(x.split(",")(46).contains("Yes")))
    val injury = drunk.filter(_.split(",")(40).contains("Fatal"))
    val result = injury.map(_.split(",")).map(row=>(row(89),row(89))).map(x=>(x,1)).reduceByKey((x,y)=>x+y)

    result.countByKey()
  }
}
```

Sample build script:

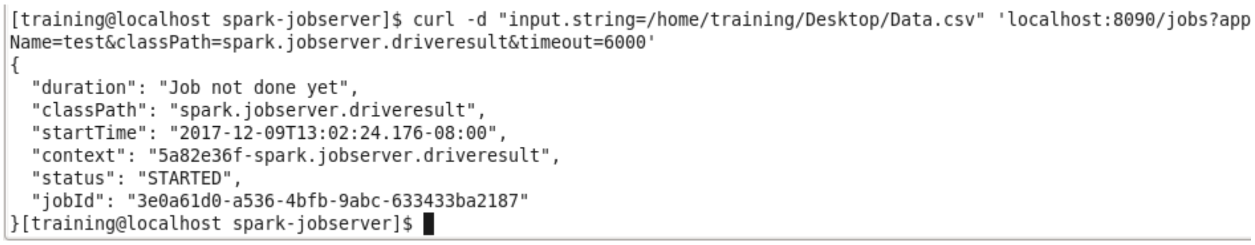
```
lazy val root = (project in file(".")).settings(name :="bigdata", version := "1.0",
scalaVersion := "2.11.1")
resolvers += "Job Server Bintray" at "https://dl.bintray.com/spark-jobserver/maven"
libraryDependencies ++= Seq("spark.jobserver" %% "job-server-api" % "0.8.0" % "provided",
"org.apache.spark" %% "spark-core" %"1.6.0" % "provided")


assemblyMergeStrategy in assembly := {
  case PathList("javax", "servlet", xs @ _*) => MergeStrategy.last
  case PathList("javax", "activation", xs @ _*) => MergeStrategy.last
  case PathList("org", "apache", xs @ _*) => MergeStrategy.last
  case PathList("com", "google", xs @ _*) => MergeStrategy.last
  case PathList("com", "esotericsoftware", xs @ _*) => MergeStrategy.last
  case PathList("com", "codahale", xs @ _*) => MergeStrategy.last
  case PathList("com", "yammer", xs @ _*) => MergeStrategy.last
  case "about.html" => MergeStrategy.rename
  case "META-INF/ECLIPSEF.RSA" => MergeStrategy.last
  case "META-INF/mailcap" => MergeStrategy.last
  case "META-INF/mimetypes.default" => MergeStrategy.last
  case "plugin.properties" => MergeStrategy.last
  case "log4j.properties" => MergeStrategy.last
  case x =>
    val oldStrategy = (assemblyMergeStrategy in assembly).value
    oldStrategy(x)
}
```

The reference to the job-server dependency library to be included in the build.sbt is presented in the readme of the jobserver and some reference is taken[4]on how to build and deploy custom application on job server.

Building the jar involves two steps, compiling the project shown above by issuing the “sbt compile” command followed by “sbt package” to create the fata jar.
I have referred to these websites[5] in order to create the project structure along with the fat jar.
Once the fat jar is created, it can be moved to the spark-jobserver/job-server-tests/target/scala/scala-2.11/ location and from there it can be uploaded for the scheduling using below command.

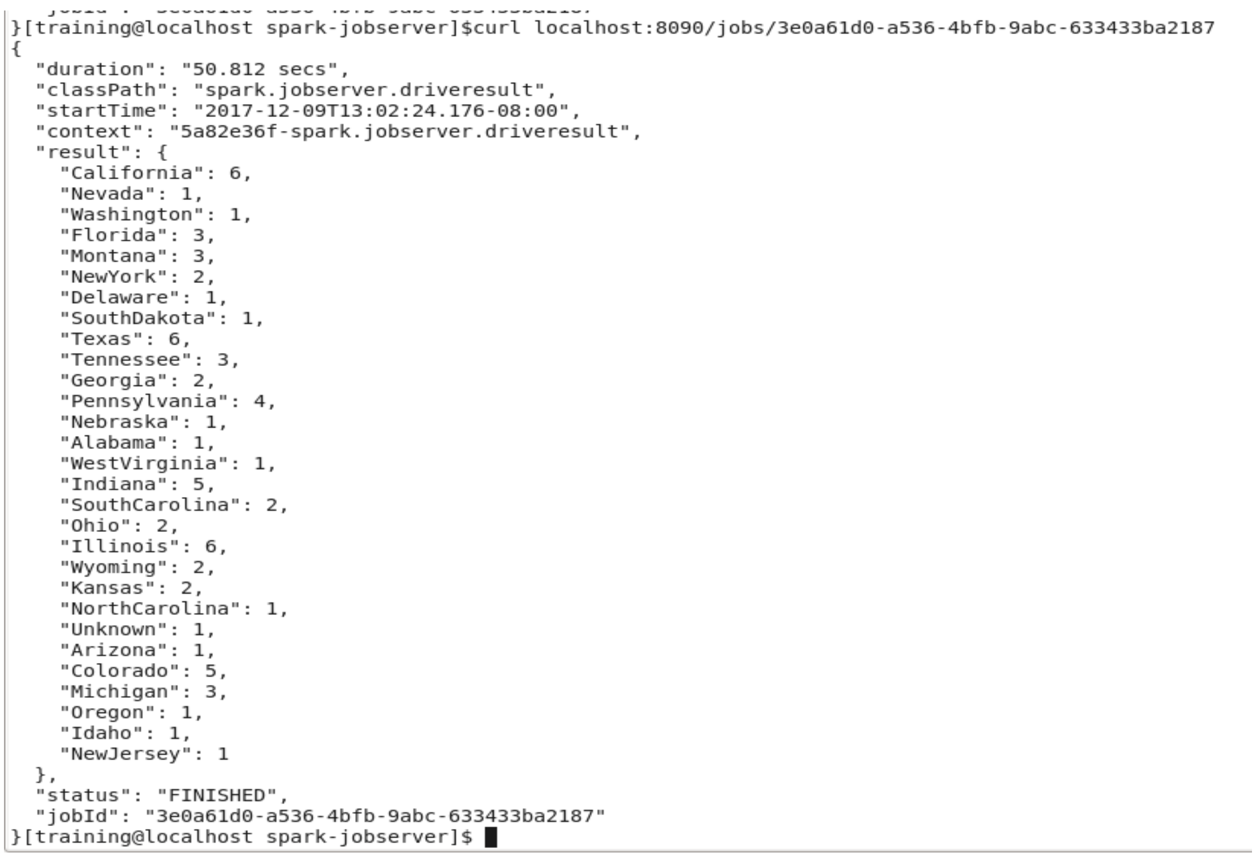
```
curl--data-binary @job-server-tests/target/scala-2.11/bigdata.jar
localhost:8090/jars/test
```

There are many ways the trigger job from the jobserver, if the rest call need not wait for the completion of the spark job we can issue the below command which returns a job id which can later be curled to see the completion of the job.



```
[training@localhost spark-jobserver]$ curl -d "input.string=/home/training/Desktop/Data.csv" 'localhost:8090/jobs?app
Name=test&classPath=spark.jobserver.driveresult&timeout=6000'
{
  "duration": "Job not done yet",
  "classPath": "spark.jobserver.driveresult",
  "startTime": "2017-12-09T13:02:24.176-08:00",
  "context": "5a82e36f-spark.jobserver.driveresult",
  "status": "STARTED",
  "jobId": "3e0a61d0-a536-4bfb-9abc-633433ba2187"
}[training@localhost spark-jobserver]$ 
```

To check for the completion of the job we can curl the below to get the output.



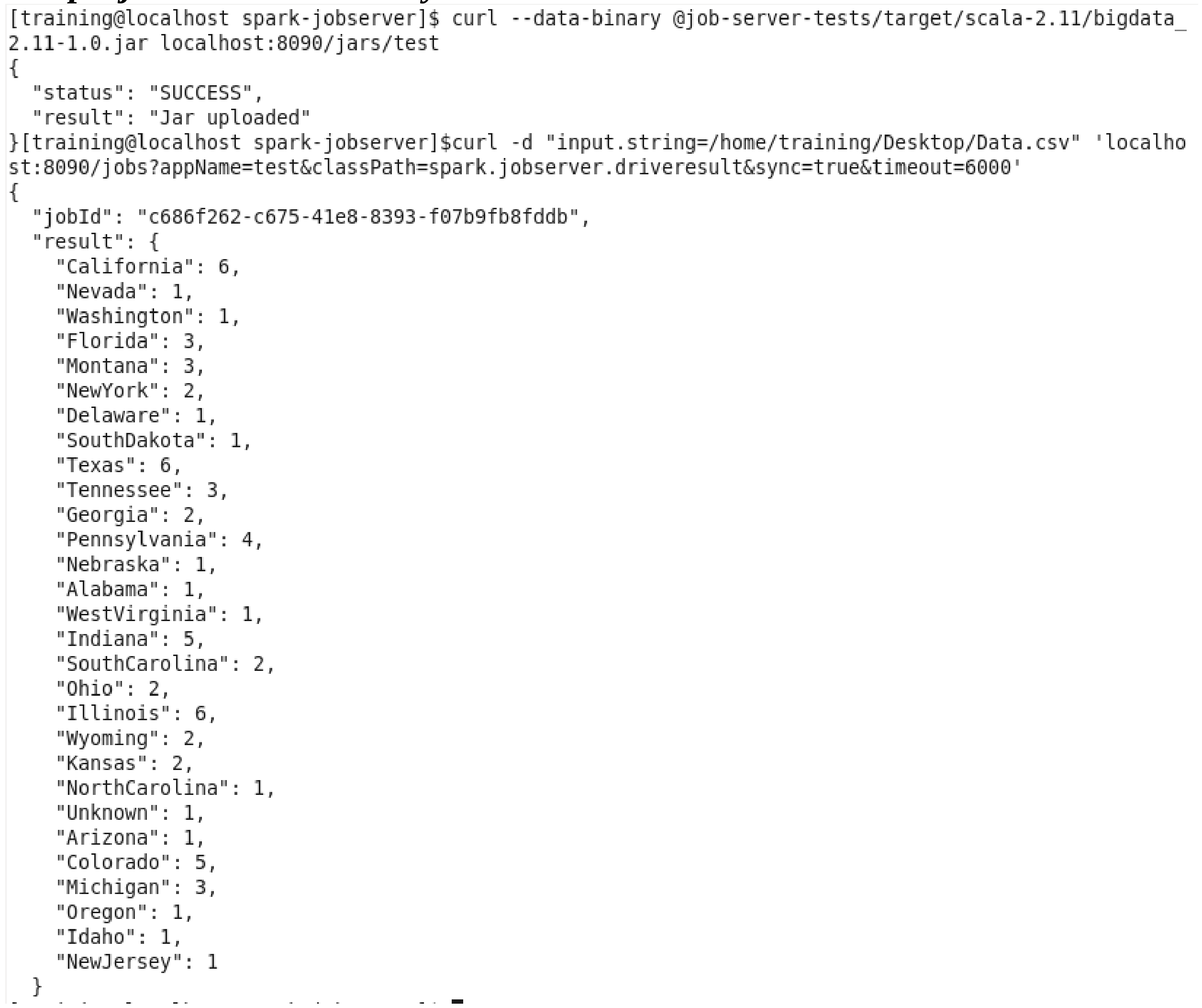
```
}[training@localhost spark-jobserver]$curl localhost:8090/jobs/3e0a61d0-a536-4bfb-9abc-633433ba2187
{
  "duration": "50.812 secs",
  "classPath": "spark.jobserver.driveresult",
  "startTime": "2017-12-09T13:02:24.176-08:00",
  "context": "5a82e36f-spark.jobserver.driveresult",
  "result": {
    "California": 6,
    "Nevada": 1,
    "Washington": 1,
    "Florida": 3,
    "Montana": 3,
    "NewYork": 2,
    "Delaware": 1,
    "SouthDakota": 1,
    "Texas": 6,
    "Tennessee": 3,
    "Georgia": 2,
    "Pennsylvania": 4,
    "Nebraska": 1,
    "Alabama": 1,
    "WestVirginia": 1,
    "Indiana": 5,
    "SouthCarolina": 2,
    "Ohio": 2,
    "Illinois": 6,
    "Wyoming": 2,
    "Kansas": 2,
    "NorthCarolina": 1,
    "Unknown": 1,
    "Arizona": 1,
    "Colorado": 5,
    "Michigan": 3,
    "Oregon": 1,
    "Idaho": 1,
    "NewJersey": 1
  },
  "status": "FINISHED",
  "jobId": "3e0a61d0-a536-4bfb-9abc-633433ba2187"
}[training@localhost spark-jobserver]$ 
```

Output Description:
The above result gives the state where accident happened on weekends and the count of people in who are teenagers, drunk and the injuries were fatal.

Output verification:
After getting the result, I manually verified them against my original data.

The other way to set “sync=true” in the input request so that the call waits for the completion of the spark job. This is important for the Gatling script as we measure the performance of the actual job run.

Sample job server run with sync=true



```
[training@localhost spark-jobserver]$ curl --data-binary @job-server-tests/target/scala-2.11/bigdata_
2.11-1.0.jar localhost:8090/jars/test
{
  "status": "SUCCESS",
  "result": "Jar uploaded"
}[training@localhost spark-jobserver]$curl -d "input.string=/home/training/Desktop/Data.csv" 'localho
st:8090/jobs?appName=test&classPath=spark.jobserver.driveresult&sync=true&timeout=6000'
{
  "jobId": "c686f262-c675-41e8-8393-f07b9fb8fddb",
  "result": {
    "California": 6,
    "Nevada": 1,
    "Washington": 1,
    "Florida": 3,
    "Montana": 3,
    "NewYork": 2,
    "Delaware": 1,
    "SouthDakota": 1,
    "Texas": 6,
    "Tennessee": 3,
    "Georgia": 2,
    "Pennsylvania": 4,
    "Nebraska": 1,
    "Alabama": 1,
    "WestVirginia": 1,
    "Indiana": 5,
    "SouthCarolina": 2,
    "Ohio": 2,
    "Illinois": 6,
    "Wyoming": 2,
    "Kansas": 2,
    "NorthCarolina": 1,
    "Unknown": 1,
    "Arizona": 1,
    "Colorado": 5,
    "Michigan": 3,
    "Oregon": 1,
    "Idaho": 1,
    "NewJersey": 1
  }
```

Performance/scale characteristics:

Installing and running simulations with Gatling

Gatling is a performance testing tool to run simulations. It helps in simulating the load in real time, we can setup scripts to fire up multiple requests or control the test to pause the requests within a given test.
We have used the open source library of Gatling with sbt plugin. The source of the project can be downloaded[6]

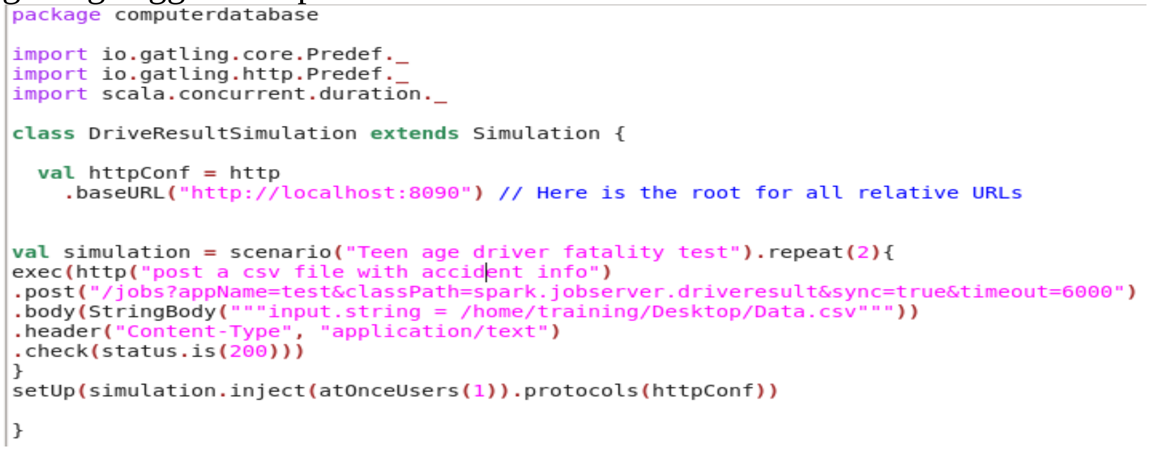
Setting up the project is easy once sbt is installed in the system. The general folder structure of the repo has test and src folders and any simulation can be added under scala test folder(src/test/scala/computerdatabase)
Once the repo is cloned, sbt can be run on the main folder of the repo to start the scala shell where the script can be triggered.

The Gatling script essentially fires of the post request explained above in the job server and then we fire of the gatling script with the below command.

sbt:gatling-sbt-plugin-demo> gatling:testOnly computerdatabase.DriveResultSimulation

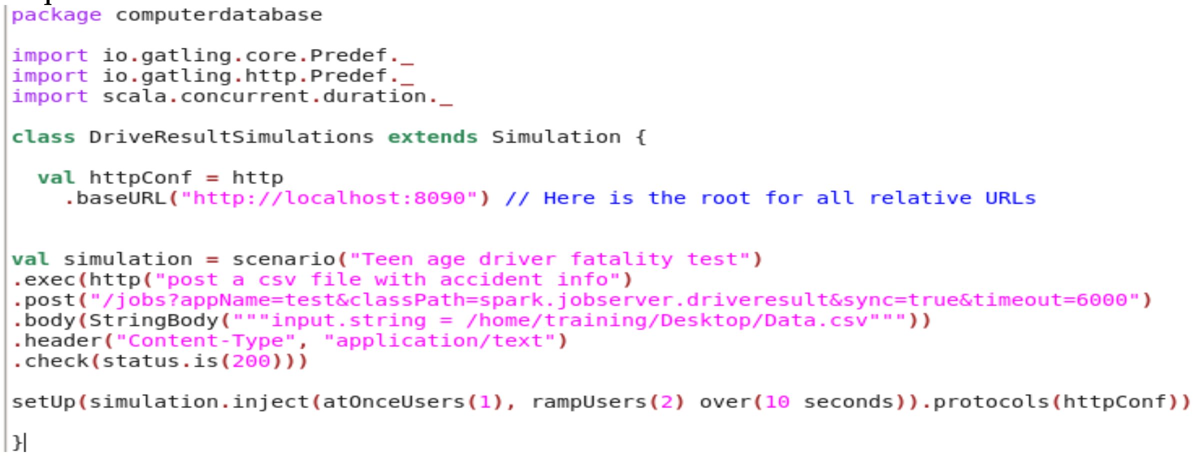
I had three variations to the script where the scala code, it was tested for a single user for a given input, the same input was tested with requests getting fired back-to back and the third simulation had two requests getting fired at the same time and later ramped up by a third request with in a span of 10seconds.

The below code snippet illustrates the simulation for single and multiple requests getting triggered in parallel.



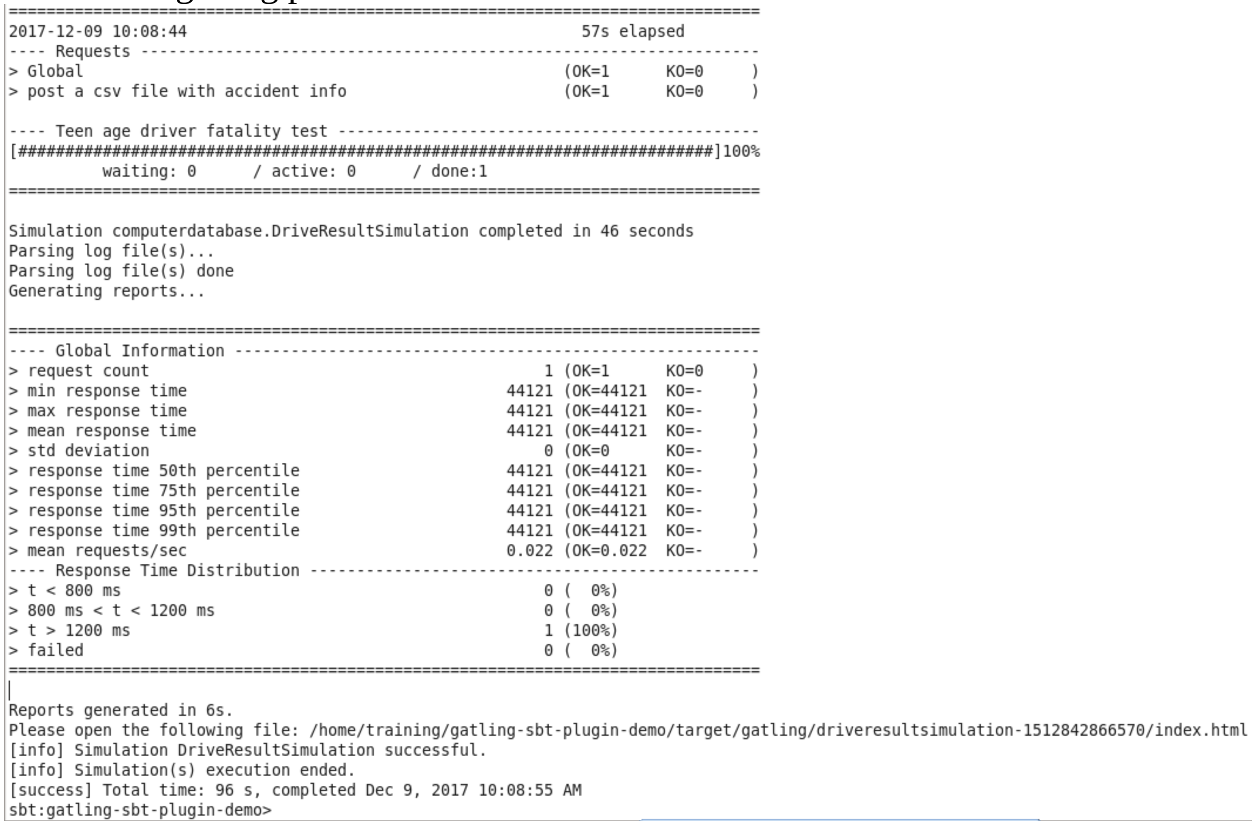
```
package computerdatabase

import io.gatling.core.Predef._
import io.gatling.http.Predef._
import scala.concurrent.duration._

class DriveResultSimulation extends Simulation {

  val httpConf = http
    .baseURL("http://localhost:8090") // Here is the root for all relative URLs


val simulation = scenario("Teen age driver fatality test").repeat(2){
exec(http("post a csv file with accident info")
.post("/jobs?appName=test&classPath=spark.jobserver.driveresult&sync=true&timeout=6000")
.body(StringBody("""input.string = /home/training/Desktop/Data.csv"""))
.header("Content-Type", "application/text")
.check(status.is(200)))
}
setUp(simulation.inject(atOnceUsers(1)).protocols(httpConf))

}
```

The below code snippet issustrates the simualtion where requests are fired in sequence.

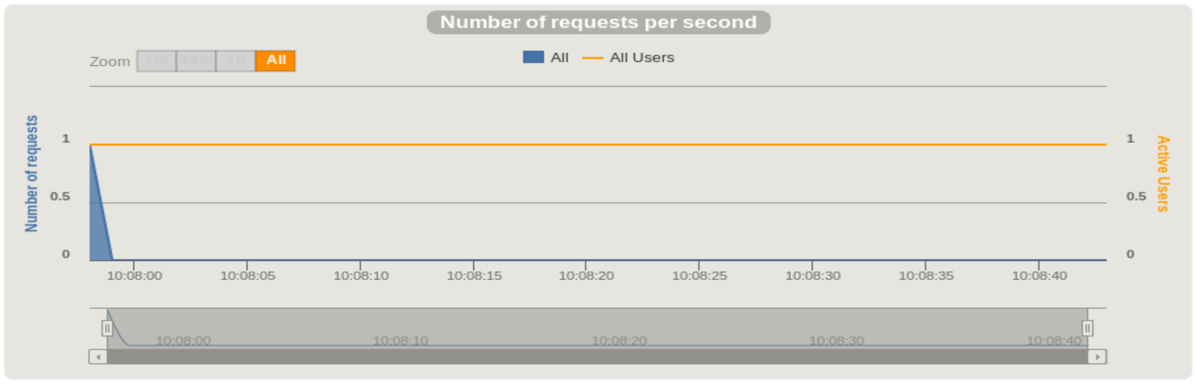


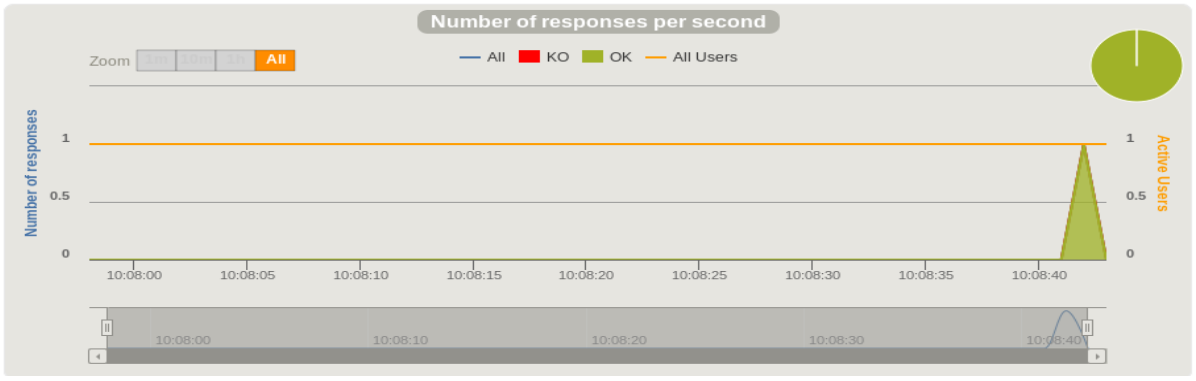
```
package computerdatabase

import io.gatling.core.Predef._
import io.gatling.http.Predef._
import scala.concurrent.duration._

class DriveResultSimulations extends Simulation {

  val httpConf = http
    .baseURL("http://localhost:8090") // Here is the root for all relative URLs


val simulation = scenario("Teen age driver fatality test")
.exec(http("post a csv file with accident info")
.post("/jobs?appName=test&classPath=spark.jobserver.driveresult&sync=true&timeout=6000")
.body(StringBody("""input.string = /home/training/Desktop/Data.csv"""))
.header("Content-Type", "application/text")
.check(status.is(200)))

setUp(simulation.inject(atOnceUsers(1), rampUsers(2) over(10 seconds)).protocols(httpConf))

}
```

Here is the gatling performance run for one user.



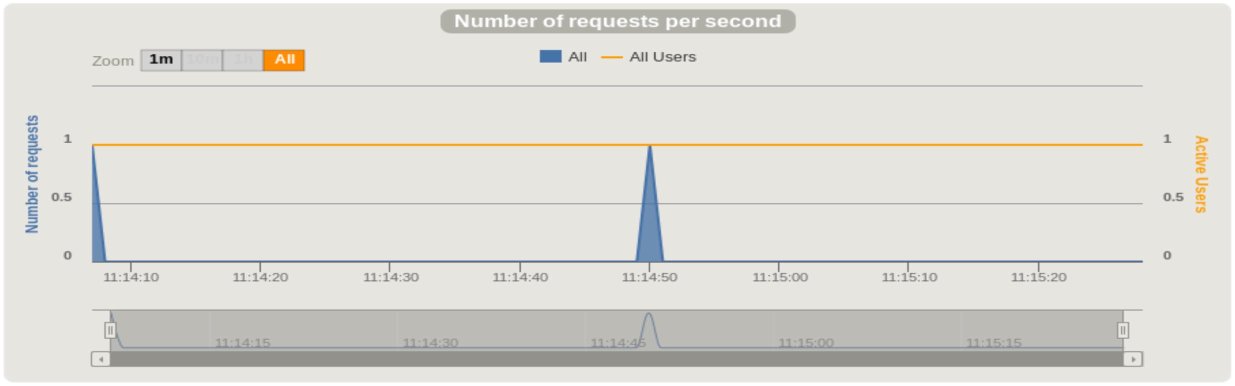
```
================================================================================
2017-12-09 10:08:44                                           57s elapsed
---- Requests ------------------------------------------------------------------
> Global                                                   (OK=1      KO=0     )
> post a csv file with accident info                       (OK=1      KO=0     )

---- Teen age driver fatality test ---------------------------------------------
[##########################################################################]100%
          waiting: 0      / active: 0      / done:1
================================================================================

Simulation computerdatabase.DriveResultSimulation completed in 46 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

================================================================================
---- Global Information --------------------------------------------------------
> request count                                          1 (OK=1      KO=0     )
> min response time                                  44121 (OK=44121  KO=-     )
> max response time                                  44121 (OK=44121  KO=-     )
> mean response time                                 44121 (OK=44121  KO=-     )
> std deviation                                          0 (OK=0      KO=-     )
> response time 50th percentile                      44121 (OK=44121  KO=-     )
> response time 75th percentile                      44121 (OK=44121  KO=-     )
> response time 95th percentile                      44121 (OK=44121  KO=-     )
> response time 99th percentile                      44121 (OK=44121  KO=-     )
> mean requests/sec                                  0.022 (OK=0.022  KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                             0 (  0%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            1 (100%)
> failed                                                 0 (  0%)
================================================================================

Reports generated in 6s.
Please open the following file: /home/training/gatling-sbt-plugin-demo/target/gatling/driveresultsimulation-1512842866570/index.html
[info] Simulation DriveResultSimulation successful.
[info] Simulation(s) execution ended.
[success] Total time: 96 s, completed Dec 9, 2017 10:08:55 AM
sbt:gatling-sbt-plugin-demo>
```

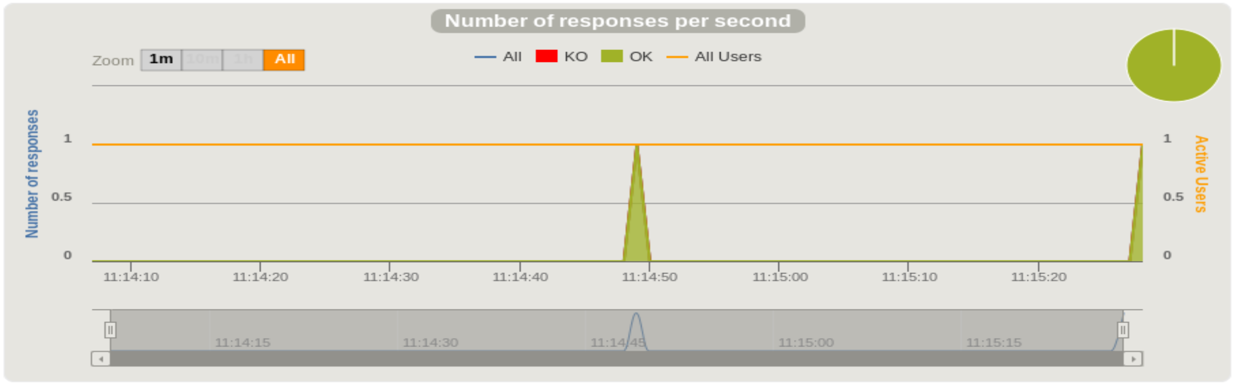




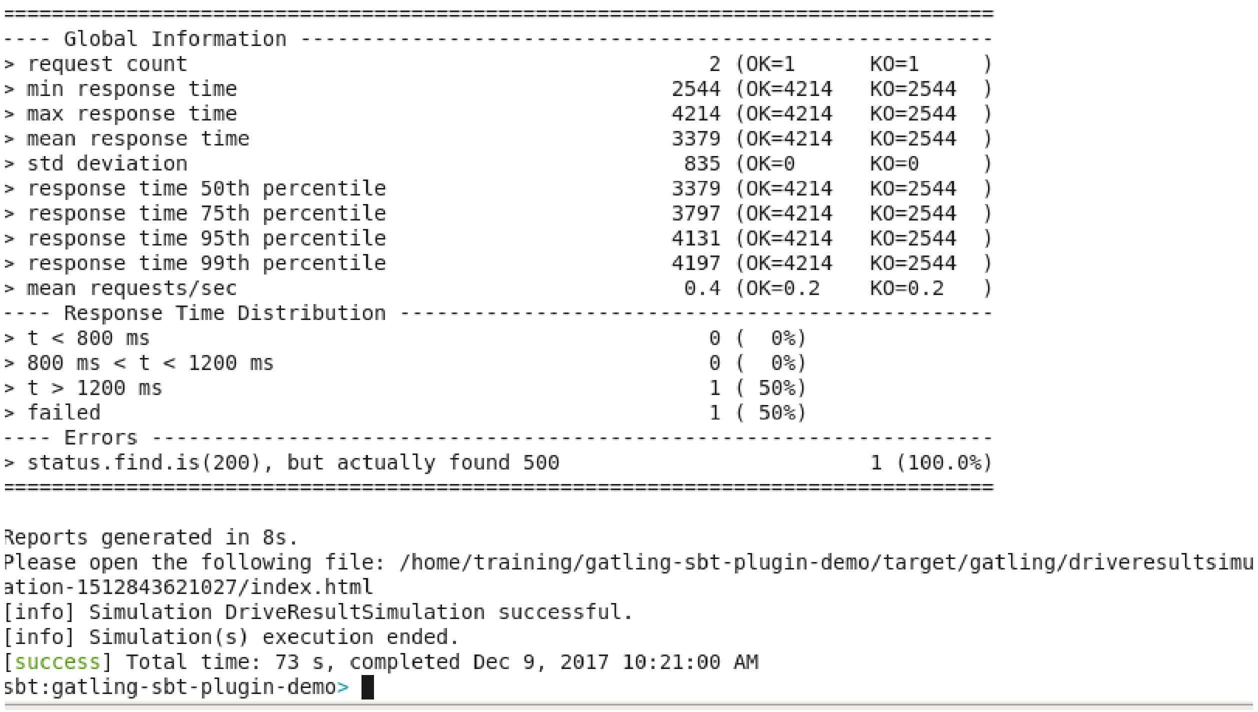
Here is the gatling performance run for two requests back to back.

```
Simulation computerdatabase.DriveResultSimulation completed in 81 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

================================================================================
---- Global Information --------------------------------------------------------
> request count                                          2 (OK=2      KO=0     )
> min response time                                  38339 (OK=38339  KO=-     )
> max response time                                  41714 (OK=41714  KO=-     )
> mean response time                                 40027 (OK=40027  KO=-     )
> std deviation                                       1688 (OK=1688   KO=-     )
> response time 50th percentile                      40027 (OK=40027  KO=-     )
> response time 75th percentile                      40870 (OK=40870  KO=-     )
> response time 95th percentile                      41545 (OK=41545  KO=-     )
> response time 99th percentile                      41680 (OK=41680  KO=-     )
> mean requests/sec                                  0.024 (OK=0.024  KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                             0 (  0%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            2 (100%)
> failed                                                 0 (  0%)
================================================================================

Reports generated in 4s.
Please open the following file: /home/training/gatling-sbt-plugin-demo/target/gatling/driveresultsimulation-151284672
7442/index.html
[info] Simulation DriveResultSimulation successful.
[info] Simulation(s) execution ended.
[success] Total time: 232 s, completed Dec 9, 2017 11:15:36 AM
sbt:gatling-sbt-plugin-demo> 
```

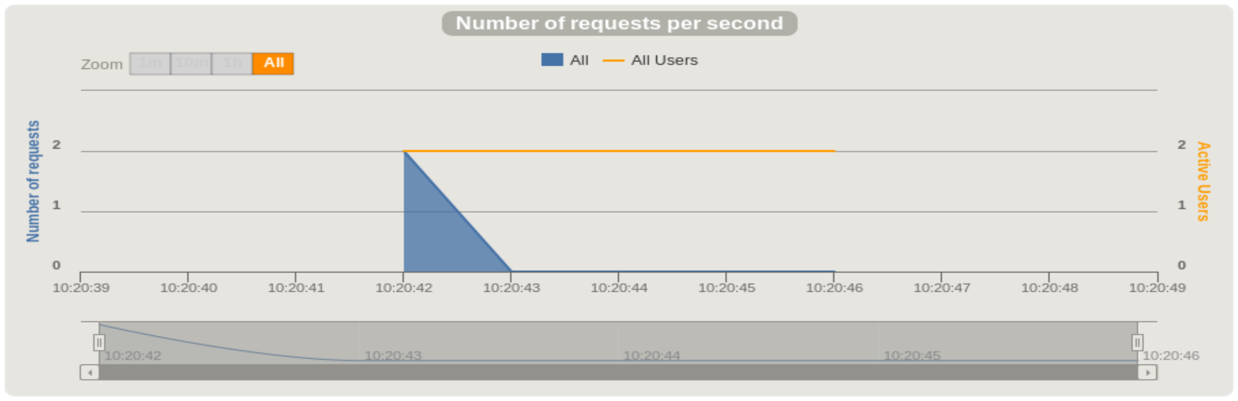


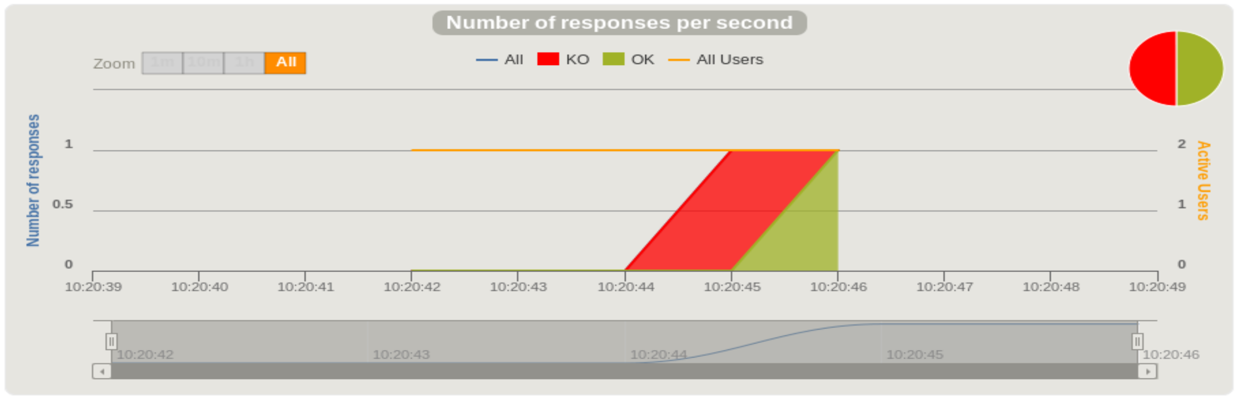


Here is the gatling report when couple of requests are fired at the same time, one request ended up failing.



```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                          2 (OK=1      KO=1      )
> min response time                                   2544 (OK=4214   KO=2544   )
> max response time                                   4214 (OK=4214   KO=2544   )
> mean response time                                  3379 (OK=4214   KO=2544   )
> std deviation                                        835 (OK=0      KO=0      )
> response time 50th percentile                       3379 (OK=4214   KO=2544   )
> response time 75th percentile                       3797 (OK=4214   KO=2544   )
> response time 95th percentile                       4131 (OK=4214   KO=2544   )
> response time 99th percentile                       4197 (OK=4214   KO=2544   )
> mean requests/sec                                    0.4 (OK=0.2    KO=0.2    )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                             0 (  0%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            1 ( 50%)
> failed                                                 1 ( 50%)
---- Errors --------------------------------------------------------------------
> status.find.is(200), but actually found 500                         1 (100.0%)
================================================================================

Reports generated in 8s.
Please open the following file: /home/training/gatling-sbt-plugin-demo/target/gatling/driveresultsimu
ation-1512843621027/index.html
[info] Simulation DriveResultSimulation successful.
[info] Simulation(s) execution ended.
[success] Total time: 73 s, completed Dec 9, 2017 10:21:00 AM
sbt:gatling-sbt-plugin-demo> 
```





As shown for the above diagrams the spark performance is consistent over multiple runs. The only thing that couldn't be established is the time it would have taken for parallel calls, considering the run is done on a single node it couldn't be achieved. The available memory in the virtual machine played a prominent role as the same Gatling run was not consistent as there are errors over different runs.

If at all the same experiment was run on a multi node cluster the results would have varied and it would be interesting to see the parallel requests and the endurance of spark nodes on ramped up requests. This would be something that would be done as the next step in this project. Changing the input file size is another good testing means, this would have given much more context on the limitations of the spark memory in a standalone mode. Unit testing of the spark code is something that would have been taken up, this is to make sure the code runs before creating the jar is created and deployed on to the job server.

Considering the complexity of the code in the current project is less, it was ok to test after creating the jar file, if the complexity of code did increase, it would have been essential to unit test each functionality individually.

Conclusion:

The project gives a glimpse of spark's power to extract, filter and aggregate data at scale to arrive at some useful analytics. It kind of tries to test sparks tipping point in terms of the scale of requests that causes the spark to perform poorly. Though we were able to address this to some extent a thorough analysis of same could have been done using a better hardware and better scheduling mechanism. There is lot of scope to understand how the spark job server internally allocates the jobs on spark nodes and changes can be made that jobs can be scheduled in a more elegant manner to make sure concurrent requests will not end up in failing but can queued to performed later in case of memory constraints.

References:

[1] https://spark-summit.org/2014/wp-content/uploads/2014/06/Testing-Spark-Best-Practices-Anupama-Shetty-Neil-Marshall.pdf

[2] https://github.com/spark-jobserver/spark-jobserver

[3] https://www.youtube.com/watch?v=JKKqAgNkHZk

[4] https://www.slideshare.net/EvanChan2/productionizing-spark-and-the-spark-job-server

[5] http://queirozf.com/entries/creating-scala-fat-jars-for-spark-on-sbt-with-sbt-assembly-plugin --> creating fat jar with sbt

http://www.scala-sbt.org/0.12.4/docs/Getting-Started/Hello.html#build-definition --> creating sbt project structure

[6] https://github.com/gatling/gatling-sbt-plugin-demo