**SparkFaultBench**

**Design Document**

**2016.9**

Contents

[SparkFaultBench System Design 4](#_Toc1157610567)

[1. Introduction 4](#_Toc1971963988)

[1.1 Purpose 4](#_Toc1627619298)

[1.2 Vision 4](#_Toc940108949)

[2. Architecture 4](#_Toc1420776431)

[3. Detail Design 6](#_Toc80471765)

[3.1 Spark SQL 6](#_Toc2054321686)

[3.1.1 Description 6](#_Toc1101628470)

[3.1.2 Test goal 6](#_Toc532835456)

[3.1.3 Data Generator 7](#_Toc180946825)

[3.1.4 Class design 8](#_Toc1835227963)

[3.2 Spark Mllib 10](#_Toc1410796436)

[3.2.1 classification 10](#_Toc294592714)

[3.2.1.1 Description 10](#_Toc1324938959)

[3.2.1.2 Test goal 11](#_Toc1906583141)

[3.2.1.3 Data generation 11](#_Toc1986275663)

[3.2.1.4 Class Design 11](#_Toc1896640424)

[3.2.2 LogisticRegressionwithLBFGS 12](#_Toc797331099)

[3.2.2.1 Description 12](#_Toc856826965)

[3.2.2.2 Test goal 13](#_Toc519766977)

[3.2.2.3 Data Generation 13](#_Toc8511377)

[3.2.2.4 Generate test workload 14](#_Toc980108898)

[3.2.2.5 Class Design 16](#_Toc908078142)

[3.2.3 DecisionTreeClassification 16](#_Toc1945986339)

[3.2.3.1 Description 16](#_Toc1754043507)

[3.2.3.2 Test Goal 16](#_Toc692475549)

[3.2.3.3 Data Generation 16](#_Toc291418907)

[3.2.3.4 Generate test workload 17](#_Toc310685205)

[3.2.3.5 Class Design 18](#_Toc334442999)

[3.2.4 RandomForestClassification 18](#_Toc1065119973)

[3.2.4.1 Description 18](#_Toc1149974603)

[3.2.4.2 Test goal 18](#_Toc1492053566)

[3.2.4.3 Data Generation 18](#_Toc889600313)

[3.2.4.4 Generate Test Workload 19](#_Toc630110253)

[3.2.4.5 Class Design 20](#_Toc284678867)

[3.2.5 MLlib-GaussianMixture 20](#_Toc162893097)

[3.2.5.1 Description 20](#_Toc710582018)

[3.2.5.2 Test goal 20](#_Toc191516905)

[3.2.5.3 Data generation 21](#_Toc1264521567)

[3.2.5.4 Class Design 21](#_Toc1243417474)

[3.3 Spark Graphx 21](#_Toc372463731)

[3.3.1 Data Volume 21](#_Toc952265882)

[3.3.2 Cluster Scale 21](#_Toc506730262)

[3.3.3 Application Description 21](#_Toc667056445)

[3.3.4 Test Objective 22](#_Toc129721193)

[3.3.5 Data Generation Type 22](#_Toc265829755)

[3.3.6 Possible Error 23](#_Toc505848460)

SparkFaultBench System Design

# Introduction

SparkFaultBench is an open-source tools to test spark and find the potential problems of spark automatically.

In this project, we mainly test Spark SQL, Spark MLlib, Spark Graphx.

With our test, we find some errors when the component run with some special data sets and some potential problems about spark memory management and spark job schedule problems.As we all know, spark is widely used,but there are many problems of spark and there are no one suitable tools to test spark,so it’s a big idea that we

Implement a tools to test spark to help the spark developers find more information.

## 1.1 Purpose

This document serves as the blueprint for the software development and implementation of SparkFaultBench.

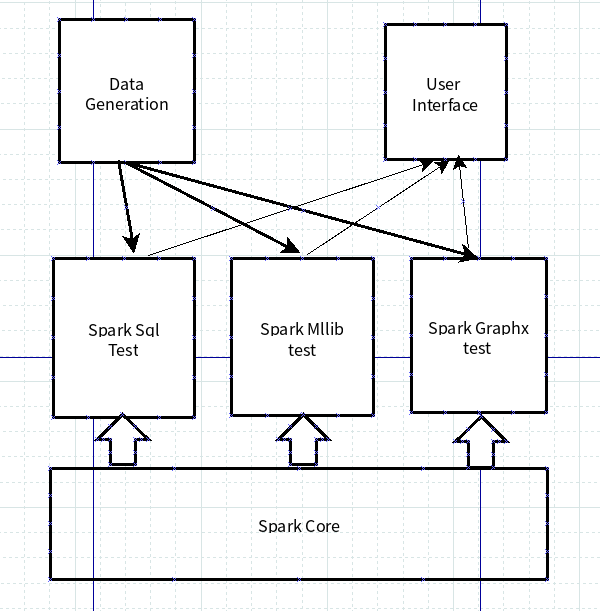
With the tools, we find many run errors of spark and some potential problems.

## 1.2 Vision

The vision for this project is to become a standard platform to test spark, besides, we want to make a unified tool to test big data platform.

# Architecture

The architecture of the whole system is shown below.



System Architecture

With the system architecture diagram, we can see that the system contains three parts:Data Generation module, Test module,User Interface module with the support of spark core.

Data Generation module: we implement many data sets with many different generation model and different data size.

Spark Component test: we can test some spark component such as Spark sql ,Spark mllib,Spark Graphx.

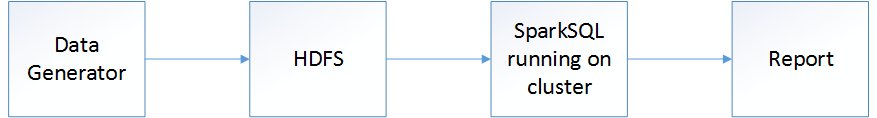
User Interface: Users can interface with our system to choose which data set to produce and which component to test.

# Detail Design

**3.1 Spark SQL**

### 3.1.1 Description

Spark SQL is Apache Spark’s module for working with structured data. In order to test Spark SQL’s performance, we design four steps to test Spark SQL module.

Figure 3.1 Running Flow chart

* Generate random local data for test
* Write the data to HDFS.
* Run the Spark program on cluster
* Generate a report for test result

## 3.1.2 Test goal

Test the spark SQL performance by running different data, in order to find out the fault in spark system.

## 3.1.3 Data Generator

According to the paper “A Comparison of Approaches to Large-Scale DataAnalysis” (http://database.cs.brown.edu/sigmod09/benchmarks-sigmod09.pdf), we create two tables named Rankings and UserVisits. The schema of these two tables is as follows:

Table 3.1 table schema

|  |  |  |
| --- | --- | --- |
| Table | type | name |
| Rankings | VARCHAR(100) | pageURL |
| INT | pageRank |
| INT | avgDuration |
| UserVisits | VARCHAR(16) | sourceIP |
| VARCHAR(100) | destURL |
| DATE | visitDate |
| FLOAT | adRevenue |
| VARCHAR(64) | userAgent |
| VARCHAR(3) | countryCode |
| VARCHAR(6) | languageCode |
| VARCHAR(32) | searchWord |
| INT | duration |

After designing table schema, data generator is easy to design. The data generation class diagram shows below.

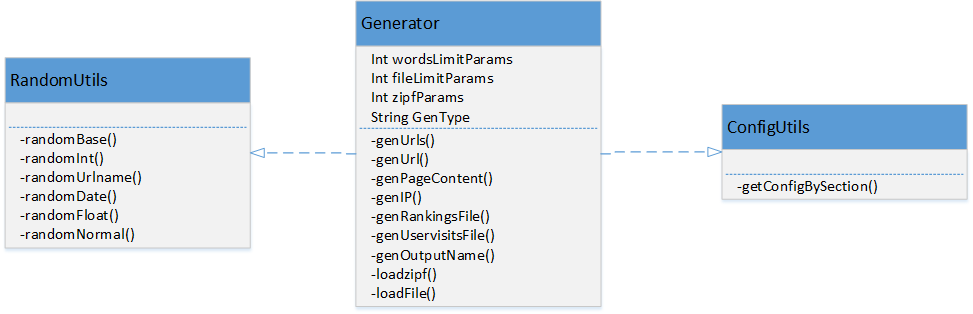


Figure 3.2 Data generation class diagram

Generator has the main function to run. It contains all generation functions for table fields. Besides, there are two interfaces for base functions, RandomUtils for generating base type data randomly, and ConfigUtils for loading outer parameters.

For details, we create 15 million UserVisits records and 10 thousand Rankings records. The pagerank, destURL fields are generated by zipfian distribution. The visitDate, adRevenue, duration fields are generated uniformly at random. Those data are picked in common situation.

However, in some particular situation, some abnormal data will be found. For example, almost all visitors linked to the same URL, most of visitors were visit through the same source IP address. We call skewed data in those situation. For some SQL queries, normal data and skewed data will display different performance. According to above , we generate not only normal data, but also skewed data which will be loaded for testing in next step

## 3.1.4 Class design

With the development of SQL standards, there are so many SQL grammars now, so we pick three typical grammars and a mix query(include 3 grammars above) to test Spark SQL performance, those are selection query, aggregation query and join query and mix query

Table 3.2 SQL Query

|  |  |
| --- | --- |
| Category | SQL Command |
| Selection | SELECT \* FROM uservisits WHERE adRevenue > X |
| Aggregation | SELECT destinationURL, sum(adRevenue) AS total  FROM uservisits  GROUP BY destinationURL  ORDER BY total DESC |
| Join | SELECT sourceipaddr, url, adrevenue  FROM rankings  INNER JOIN uservisits  ON url = destinationURL  ORDER BY adrevenue DESC LIMIT 100 |
| Mix | SELECT sourceIPAddr, totalRevenue, avgPageRank  FROM (SELECT sourceIPAddr,  AVG(pageRank) as avgPageRank,  SUM(adRevenue) as totalRevenue  FROM Rankings AS R, Uservisits AS UV  WHERE R.url = UV.destinationURL  AND UV.avgTimeOnSite BETWEEN 30 AND 70  GROUP BY UV.sourceIPAddr)  ORDER BY totalRevenue DESC LIMIT 10 |

The Spark SQL test class design shows below.

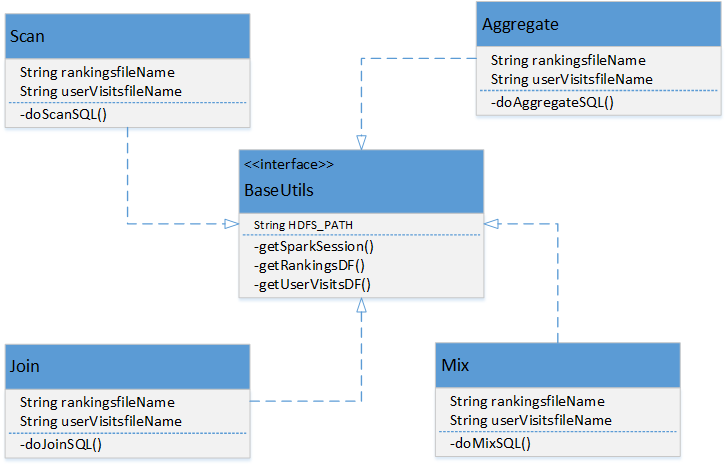


Figure 3.3 Spark SQL class diagram

Actually, we create four class to implement four SQL queries, there are named “Scan”, ”Aggregate”, “Join” and “Mix”, Besides, a general interface for loading files and setting parameters. In order to display the different performance between skewed data and normal data, each SQL command has to run twice, one for normal data, one for the other.

**3.2 Spark Mllib**

**3.2.1 classification**

We mainly test KMeans and NaiveBayes in the Spark Mllib classification part.

**3.2.1.1 Description**

k-means clustering: aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

NaiveBayes:Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

**3.2.1.2 Test goal**

Basic goal:Check if there are some error when this algorithm run on various DataSet;

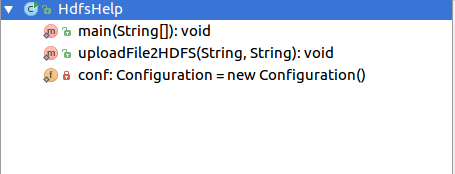
Furthermore:find if there are some potential performance problems of Kmeans and NaiveBayes on spark

**3.2.1.3 Data generation**

Use Scala program’s random generator produce random number between 0 and 100.

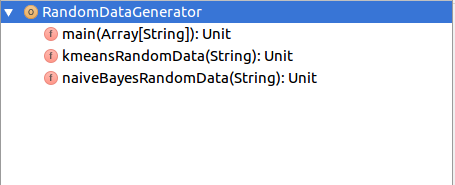
**3.2.1.4 Class Design**

We use Class HdfsHelp as a Utils class to uplaod lcoal file to the Hadoop HDFS system, the class structure is as below.

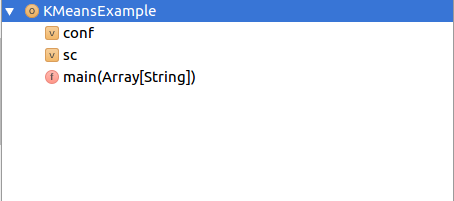


We use the class below to generate the dataSet as below with a Scala random data generator.

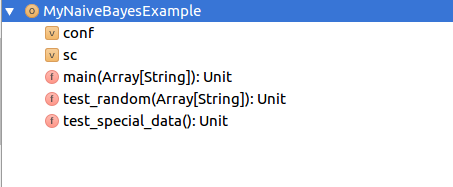
|  |  |  |  |
| --- | --- | --- | --- |
|  | Small size | Big size | Generator |
| Kmeans | 57M | 570M | Random |
| NaiveBayes | 10M | 1G | Random |



We use the class below to test KMeans in spark mllib.



We use the class below to test Naive



**3.2.2 LogisticRegressionwithLBFGS**

**3.2.2.1 Description**

Logistic Regression with LBFGS is a classification algorithm with LBFGS. The application divides data into training set and testing set, it uses training set to generate a model by the algorithm and then predicts the class of the test data belongs to according the model.

**3.2.2.2 Test goal**

To measure the performance of Logistic Regression with LBFGS algorithm in Spark with difference scale of data and different setting of parameter.

Measuring scalability: if operation time linearly increase with the amount of samples or not.

Measuring stability and reliability: if there is out of memory, memory leak or other errors when samples and parameters of the algorithm change.

**3.2.2.3 Data Generation**

The input data of Logistic Regression with LBFGS is labeled data in the LIBSVM format, which has the form: label index1:value1 index2:value2 ...

In binary logistic regression, labels include values 0 and 1, index is a characteristic value, a record has a large dimension of characteristics.

Test data can be in the real world or be composite of manually generated data.

To test the performance of Spark in different configuration of data. We design the data with changing five elements -- number of dimensions, number of samples, number of negative samples which leads to data skew and percent of not “NA” data.

Data standardization: In LBFGS version of logistic regression, the data is properly standardized.

Data regularization: the data should be regularized to avoid overfitting, especially when the number of training examples is small. In LBFGS version of logistic regression, it chooses L2 regularization as default, And we can choose L1 regularization by setting in program.

Manually generated data:

The manually generating data program address is https://github.com/JerryLead/SparkFaultBench/blob/master/dataGenerated/mllib/swt/genEnormous.py.

Data generate from the real world:

Data source is <http://stat-computing.org/dataexpo/2009/the-data.html>. Download 1987.csv, 1988.csv, 1989.csv, 1990.csv, 1991.csv, 1992.csv, 1993.csv.

Choose characteristic of “Canceled” which just consists of 0 and 1 as label. Each “\*.csv” file has same dimension of 29. Join them together and form each column to LIBSVM format. Finally, the LIBSVM format text has 7340025 samples. To test with a smaller number of samples too, form 1987.csv which has 1048576 samples to LIBSVM format. The LIVSVM format program address is https://github.com/JerryLead/SparkFaultBench/blob/master/dataGenerated/mllib/swt/formatLibSVM.py.

**3.2.2.4 Generate test workload**

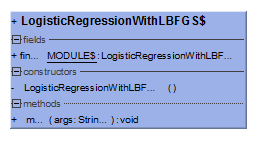
Parameters which can be set in Logistic Regression with LBFGS algorithm:

|  |  |  |
| --- | --- | --- |
| parameter | significance | default |
| numclass | number of possible outcomes for k classes classification | 2 |
| numIterations | the maximal number of iterations for L-BFGS | 100 |
| convergencoTol | the convergence tolerance of iterations for L-BFGS | 1E-6 |
| numRegparam | the regularization parameter | 1.0 |
| threshold | the threshold that separates positive predictions from negative predictions in Binary Logistic Regression. | 0.5 |
| numcorrection | the number of corrections used in the LBFGS update | 10 |

Design the following test groups with different amount of samples, dimensions, Inclination of samples, number of classes and normalized or not.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| samples | dimension | negative:positive | percent of not “NA” | normalized | parameters | | | | |
| classes | iterations | convergencoTol | regparam | correction |
| 100 | 10 | 1:1 | 60 | no | 10 | 100 | 1E-6 | 1.0 | 10 |
| 100 | 100 | 1:1 | 60 | no | 10 | 100 | 1E-6 | 1.0 | 10 |
| 100 | 200 | 1:1 | 60 | no | 10 | 100 | 1E-6 | 1.0 | 10 |
| 10000 | 100 | 9:1 | 60 | no | 10 | 100 | 1E-6 | 1.0 | 10 |
| 10000 | 100 | 9:1 | 10 | no | 10 | 100 | 1E-6 | 1.0 | 10 |
| 10000 | 100 | 1:1 | 60 | no | 10 | 100 | 1E-6 | 1.0 | 10 |
| 1000000 | 100 | 1:1 | 60 | no | 10 | 100 | 1E-6 | 1.0 | 10 |
| 7340025 | 29 | 54:1 | 63.8 | No | 10 | 100 | 1E-6 | 1.0 | 10 |
| 1048576 | 29 | 71:1 | 64.0 | No | 10 | 100 | 1E-6 | 1.0 | 10 |
| 7340025 | 29 | 54:1 | 63.8 | Yes | 10 | 100 | 1E-6 | 1.0 | 10 |
| 7340025 | 29 | 54:1 | 63.8 | No | 2 | 100 | 1E-6 | 1.0 | 10 |

**3.2.2.5 Class Design**



**3.2.3 DecisionTreeClassification**

**3.2.3.1 Description**

Training data Based on classification tree structure, selecting the optimal partition properties each time, generating learned regression tree model, then predict the class of the test data belongs to according the model and get mean squared error and depth and nodes of the tree.

**3.2.3.2 Test Goal**

To measure the performance of Decision Tree Classification algorithm in Spark with difference scale of data and different setting of parameter.

Measuring scalability: if operation time linearly increase with the amount of samples or not.

Measuring stability and reliability: if there is out of memory, memory leak or other errors when samples and parameters of the algorithm change.

**3.2.3.3 Data Generation**

The input data of Decision Tree Classification is labeled data in the LIBSVM format, which has the form: label index1:value1 index2:value2 ...

Test data can be in the real world or be composite of manually generated data.

We generated the data with changing five elements -- number of dimensions, number of samples, number of negative samples which leads to data skew and percent of not “NA” data. There is no need to standard or regularize the data.

**3.2.3.4 Generate test workload**

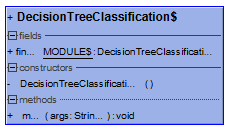
Parameters which can be set in Decision Tree Classification algorithm:

|  |  |  |
| --- | --- | --- |
| parameter | significance | suggested |
| maxDepth | Maximum depth of the tree | 5 |
| maxBins | Maximum number of bins used for splitting features | 32 |

Design the following test groups with different amount of samples, dimensions, inclination of samples and max depth. Record operation time, mean squared error and depth and nodes of the tree.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| samples | dimension | negative:positive | percent of not “NA” | parameters | |
| maxDepth | maxBins |
| 100 | 10 | 1:1 | 60 | 5 | 32 |
| 100 | 100 | 1:1 | 60 | 5 | 32 |
| 100 | 200 | 1:1 | 60 | 5 | 32 |
| 10000 | 100 | 9:1 | 60 | 5 | 32 |
| 10000 | 100 | 9:1 | 10 | 5 | 32 |
| 10000 | 100 | 9:1 | 60 | 40 | 32 |
| 10000 | 100 | 1:1 | 60 | 5 | 32 |
| 1000000 | 100 | 1:1 | 60 | 5 | 32 |

**3.2.3.5 Class Design**



**3.2.4 RandomForestClassification**

**3.2.4.1 Description**

Use Random Forest Classification algorithm to generate learned regression tree model, then predict the class of the test data belongs to according the model and get mean squared error and depth and trees in the forest.

**3.2.4.2 Test goal**

To measure the performance of Random Forest Classification algorithm in Spark with difference scale of data and different setting of parameter.

Measuring scalability: if operation time linearly increase with the amount of samples or not.

Measuring stability and reliability: if there is out of memory, memory leak or other errors when samples and parameters of the algorithm change.

**3.2.4.3 Data Generation**

The input data of Random Forest Classification algorithm is labeled data in the LIBSVM format, which has the form: label index1:value1 index2:value2 ...

Test data can be in the real world or be composite of manually generated data.

We generated the data with changing five elements -- number of dimensions, number of samples, number of negative samples which leads to data skew and percent of not “NA” data. There is no need to standard or regularize the data.

**3.2.4.4 Generate Test Workload**

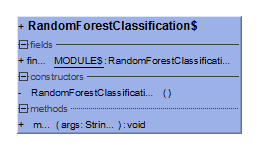
Parameters which can be set in Decision Tree Classification algorithm:

|  |  |  |
| --- | --- | --- |
| parameter | significance | Default/suggested |
| impurity | Criterion used for information gain calculation | gini |
| numClasses | Number of classes for classification. | 2 |
| numTrees | Number of trees in the random forest | 2 |
| maxDepth | Maximum depth of the tree | 5 |
| maxBins | Maximum number of bins used for splitting features | 32 |

Design the following test groups with different amount of samples, dimensions, inclination of samples and number of trees. Record operation time, mean squared error and depth and trees in the forest.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| samples | dimension | negative:positive | percent of not “NA” | parameters | | | |
| MaxDepth | MaxBins | classes | trees |
| 100 | 10 | 1:1 | 60 | 5 | 32 | 2 | 2 |
| 100 | 100 | 1:1 | 60 | 5 | 32 | 2 | 2 |
| 100 | 200 | 1:1 | 60 | 5 | 32 | 2 | 2 |
| 10000 | 100 | 9:1 | 60 | 5 | 32 | 2 | 2 |
| 10000 | 100 | 9:1 | 10 | 5 | 32 | 2 | 2 |
| 10000 | 100 | 9:1 | 60 | 5 | 32 | 2 | 10 |
| 10000 | 100 | 9:1 | 60 | 5 | 32 | 2 | 100 |
| 10000 | 100 | 1:1 | 60 | 5 | 32 | 2 | 2 |
| 1000000 | 100 | 1:1 | 60 | 5 | 32 | 2 | 2 |
| 1000000 | 100 | 1:1 | 60 | 30 | 32 | 2 | 100 |

**3.2.4.5 Class Design**



**3.2.5 MLlib-GaussianMixture**

**3.2.5.1 Description**

GaussianMixture model is used for quantifying things accurately by [Gaussian](C:/Users/kly/AppData/Local/youdao/dict/Application/6.3.69.8341/resultui/frame/javascript:void(0);) [probability-density](C:/Users/kly/AppData/Local/youdao/dict/Application/6.3.69.8341/resultui/frame/javascript:void(0);) [function](C:/Users/kly/AppData/Local/youdao/dict/Application/6.3.69.8341/resultui/frame/javascript:void(0);). Multivariate Gaussian Mixture Model(GMM) consisting of k Gaussians, where points are drawn from each Gaussian=i...k with probability w(i); mu(i) and sigma(i) are the respective mean and covariance for each Gaussian distribution i=i...k.

**3.2.5.2 Test goal**

To measure the performance of GaussianMixture in Spark with difference scale of data and different setting of parameter.

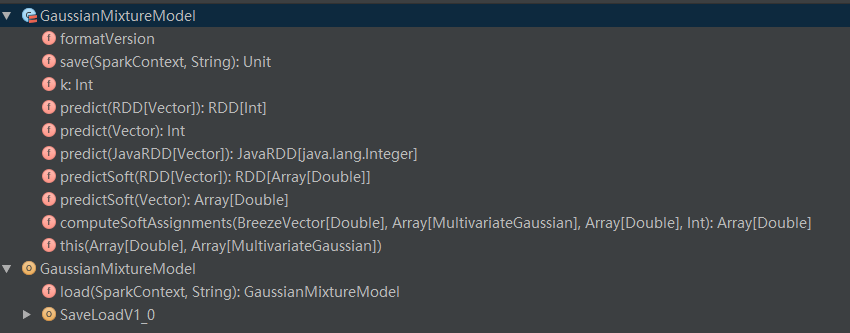
Measuring scalability: if operation time linearly increase with the amount of samples or not.

Measuring stability and reliability: if there is out of memory, memory leak or other errors when samples and parameters of the algorithm change.

**3.2.5.3 Data generation**

Use python script produce random number between -10 and 10 with amount of 100000,1000000, and 100000000.

**3.2.5.4 Class Design**



**3.3 Spark Graphx**

**3.3.1 Data Volume**

Generate graph data as we want, and the default number of vertices in the graph is 100000.

**3.3.2 Cluster Scale**

Master: System CentOS6, 60G Disk, 6 Vcpus, 6G Memory.

Slaves(1-5): System CentOS6, 100G Disk, 6 Vcpus, 6G Memory.

**3.3.3 Application Description**

GraphBenchmark can test four algorithms, including PageRank, ConnectedComponents, SSSP and TriangleCounting.

①PageRank is a way of measuring the importance of website pages. It works by counting the number and quality of links to a page to determine a rough estimate of how important the website is .

②ConnectedComponents computes the number of connected components. A connected component (or just component) of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.

③SSSP, Shortest path problem, find the shortest path between a vertice with other vertices, implemented by Dijkstra.

④TriangleCounting computes the number of triangles in a graph, often used in Community. The community is more stabilize with more triangles.

**3.3.4 Test Objective**

①Find change of the running time, memory and CPU cores used in algorithms with increasing the data volume.

②Find the possible shortages of algorithms

③Find the possible errors of algorithms with increasing the data volume.

**3.3.5 Data Generation Type**

We generate a graph whose vertex out degree distribution is log normal.

logNormalGraph( sc: SparkContext, numVertices: Int, numEParts: Int = 0, mu: Double = 4.0, sigma: Double = 1.3, seed: Long = -1)

@param sc Spark Context

@param numVertices number of vertices in generated graph

@param numEParts (optional) number of partitions

@param mu (optional, default: 4.0) mean of out-degree distribution

@param sigma (optional, default: 1.3) standard deviation of out-degree distribution

@param seed (optional, default: -1) seed for RNGs, -1 causes a random seed to be chosen

@return Graph object

**3.3.6 Possible Error**

java heap space : when test graph.pageRank(0.0001) with the graph data with 100000 vertices generated by logNormalGraph().