User manual, Design Document, Assumptions

**Generating log file**

* **To run the tool from the command line for generating the log file**

1. Open generate.sh under LogSimulator directory and set the JAVA\_HOME variable to the exact path on the system for Java SDK
2. Make sure that the path to the logsimulator.jar file is correct in the query.sh
3. Change the permissions to executable mode (eg – 775)
4. Run the file using ./generate.sh <YOUR DATA DIRECTORY PATH>

* **To run the tool from an IDE**

1. Import the LogSimulator into an IDE such as Eclipse.
2. Open the Generate.java file, right click and click on run java configurations
3. Provide a valid data path where the log file should be stored and choose apply
4. Run the Generate.java file

**Querying log file for cpu usage**

* **To run the tool from an IDE**

1. Import the LogQuerier into an IDE such as Eclipse.
2. Open the QueryCpuUsage.java file, right click and click on run java configurations
3. Provide a valid data path where the log file would be stored and choose apply
4. Run the QueryCpuUsage.java file

* **To run the tool from the command line for querying**

1. Open query.sh under LogQuerier directory and set the JAVA\_HOME variable to the exact path on the system for Java SDK
2. Make sure that the path to the jar file is correct in the query.sh
3. Change the permissions to executable (eg – 775)
4. Run the file using ./query.sh <YOUR DATA DIRECTORY PATH>

**Assumptions**

* The data path supplied by the user is a directory
* JAR files are assumed to have been placed in the Project folders
* The date for which the user queries cpu usage for is 2010-10-31
* For running the shell scripts, the user will have to have a linux machine or a virtual machine with Java 1.8 sdk installed on it
* No memory restriction, right now the query tool will at least consume around 90mb of RAM to internally store the file as a hash map
* The query tool can take however long to initialize. Right now the tool takes close to 4 seconds to initialize.

**Design Documentation**

The crux of the search/query tool lies in the fact that the whole file could be loaded in memory before the user even starts typing the query. The data structure used to store the file is explained below

**HashMap<”Server ip”,<HashMap<”cpu id”, int[][]>>>**

There is an external hash map whose key is the server IP. The reason the external hash map was chosen with the IP as the key was to focus on only the server we want. The next level of hash map involves looking at the CPU ID(either 0 or 1) as the key. This way we will have to look at just 1\*24\*60 cpu usages. Next step involved iterating over the 2D array, the reason a 2D array was chosen was to directly hit the row(hour) followed by iterating over the column(minutes). So for example

QUERY 192.168.1.10 1 2014-10-31 00:00 2014-10-31 00:05

The above query works as explained below:

1. First step we look for a key “192.168.1.10” in the external hash map.
2. Search for “1” in the value after first search
3. Since hour = 0, hit the 0th row directly and then iterate 5 elements usages to get the result

**Output**

Time taken for initialization 3.957 secs

>QUERY 192.168.1.12 0 2014-10-31 00:00 2014-10-31 00:05

CPU0 usage on 192.168.1.12:

(2014-10-31 00:00 21%),(2014-10-31 00:01 86%),(2014-10-31 00:02 13%),(2014-10-31 00:03 97%),(2014-10-31 00:04 18%),(2014-10-31 00:05 25%)

Time taken for results 0.004 secs

>QUERY 192.168.1.10 1 2014-10-31 00:00 2014-10-31 00:05

CPU1 usage on 192.168.1.10:

(2014-10-31 00:00 71%),(2014-10-31 00:01 70%),(2014-10-31 00:02 39%),(2014-10-31 00:03 2%),(2014-10-31 00:04 96%),(2014-10-31 00:05 14%)

Time taken for results 0.001 secs

>EXIT