SimLogger.java

**Summary:** This module has got several get/set methods thru which it calculates functions (self explanatory, see function definitions in the following sequence order)

1. SimLogger()

It is a private constructor that prevents any other class from instantiating.

1. SimLogger getInstance()

It is used for digital signature on singleton class

1. void enableFileLog()

It enables file log writing

1. void enablePrintLog()

It enables print log using *printLogEnabled = true*

1. boolean isFileLogEnabled()

It checks if file log write is enabled(T) or not(F)

1. void disablePrintLog()

It disables the print log using *printLogEnabled = false*

1. void appendToFile(BufferedWriter bw, String line)

It appends the line at the end of file-write and then creates a newline for next rec write

1. File getConsoleTxtFile()

It writes text from console to file

1. void printLine(String msg)

If printLogEnabled is true, it prints msg to the console and to the file

1. void print(String msg)

Does the same as #7.

1. void fileInitialize(String outputFolder)

If the output file folder doesn’t exist, create one.

Create a textfile based on current millisecond and then two file handles for textfile and printfile.

1. void simStarted(String outFolder, String fileName)

It creates vmLoadList, fnMipsUtilList, fnNwUtilList along with taskMap and Utilization arrays.

Then, it creates centralize log file to write into it.

1. void addFNMipsUtilizationLog(double time, int hostId, int hostLevel, double fnMipsUtil)

It adds to the above list (created in #10) – fnMipsUtilList using above param.

1. void addFNNwUtilizationLog(double time, int hostId, int hostLevel, double fnBwUtil)

It adds to the above list (created in #10) – fnNwUtilList using above param.

1. PrintWriter getCentralizeLogPrinter()

It returns centerFileW, a file handle for printer

1. void addLog(double taskStartTime, int taskId, int taskType, int taskLength, int taskInputType,  
    int taskOutputSize)

It adds to the above list (created in #10) – taskMap using above param w/o taskId

1. void uploadStarted(int taskId, double taskUploadTime)

It starts uploading each task using UploadTime and taskId from taskMap

1. void uploaded(int taskId, int datacenterId, int hostId, int vmId, int vmType)

It uploads each task (using it’s taskId) from taskMap using the above params

1. void downloadStarted(int taskId, double taskDownloadTime)

It starts downloading each task using DownloadTime and taskId from taskMap

1. void downloaded(int taskId, double taskEndTime)

It downloads each task (using it’s taskId) from taskMap using the above params

1. void downloaded(int taskId, double taskEndTime, double cost)

It is an overloaded (overloading param being cost) function of #18 above. It downloads

each task (using it’s taskId) from taskMap using the above params

1. void taskRejected(int taskId, double taskRejectTime, TASK\_STATUS taskStatus)

It indicates each task (taskId) geting rejected at what time and with what status (see 13 below)

1. CREATED
2. UPLOADING
3. PROCESSING
4. DOWNLOADING
5. COMPLETED
6. REJECTED\_DUE\_TO\_VM\_CAPACITY
7. REJECTED\_DUE\_TO\_BANDWIDTH
8. UNFINISHED\_DUE\_TO\_BANDWIDTH
9. UNFINISHED\_DUE\_TO\_MOBILITY
10. ASSIGNED\_HOST
11. REJECTED\_DUE\_TO\_LACK\_OF\_NODE\_CAPACITY
12. REJECTED\_DUE\_TO\_LACK\_OF\_NETWORK\_BANDWIDTH
13. REJECTED\_DUE\_TO\_UNACCEPTABLE\_LATENCY
14. void rejectedDueToVMCapacity(int taskId, double taskRejectTime)

It indicates specific task (from taskMap) being rejected due to VM Capacity at a given reject time

1. void rejectedDueToBandwidth(int taskId, double taskRejectTime, int vmType)

It indicates specific task (from taskMap) being rejected due to Bandwidth at a given reject time and vmType (vm\_Cloud, vm\_Edge)

1. void failedDueToBandwidth(int taskId, double taskRejectTime)

It indicates specific task (from taskMap) failed due to Bandwidth at a given reject time

1. void failedDueToMobility(int taskId, double time)

It indicates specific task (from taskMap) failed due to Mobility at a given time

1. void addVmUtilizationLog(double time, double load)

It adds to vmLoadList (created in #10) using time & load

1. void addHostDistanceLog(int taskId, double dist)

It gets specific task (from taskMap) and then sets a specific distance (=dist)

1. void addUserDistanceLog(int taskId, double dist)

It gets specific task (from taskMap) and then sets a specific distance (=dist) from the user

**Note:** Same function as #26 doing different job.

1. void addHops(int taskId, int hops)

It gets specific task (from taskMap) and then sets hops

1. void addHopsBack(int taskId, int hops)

It gets specific task (from taskMap) and then sets hops back to the user.

1. void getTotalFogNodesCountInEachLevel()

It makes node set from SimManager, getNetworkModel(), getNetworkTopology(), getNodes();

Then from each of these node set, keep adding totalNodesNmuberInEachLevel using getLevel()

1. void addNodeUtilization(int hostId, EdgeHost host)

Fog node utilization -- Add hostId to utilizationArray if not there. Also count fog node in each level.

1. void addCloudletToLevel(int level)

Counts nbr of levels for cloud-vm.

1. void simStopped() -- It’s a big one described at the end.
2. double getTaskPerceivedDelay (int taskId)

It return perceived delay i.e. service time of a given task after completion of its execution to verify if the task execution response took longer than allowed time to reach the user.

1. boolean isPrintLogEnabled()

It returns printLogEnabled

1. void setPrintLogEnabled(boolean printLogEnabled)

It sets the printLogEnabled

1. String getFilePrefix()

It gets the file prefix.

1. void setFilePrefix(String filePrefix)

It sets the file prefix.

1. String getOutputFolder()

It gets the output folder

1. void setOutputFolder(String outputFolder)

It sets the output folder

1. Map<Integer, LogItem> getTaskMap()

It gets TaskMap with key=Integer and value=LogItem pair

1. void setTaskMap(Map<Integer, LogItem> taskMap)

It sets the TaskMap with the above pair

1. LinkedList<VmLoadLogItem> getVmLoadList()

It gets the VmLoadList as sequence of nodes with value and reference to the next & prev node list.

1. void setVmLoadList(LinkedList<VmLoadLogItem> vmLoadList)

It sets the VmLoadList from the above linked list

1. File getCenterLogFile()

It gets the file handle for the central log file

1. void setCenterLogFile(File centerLogFile)

It sets the file handle for the central log file

1. PrintWriter getCenterFileW()

It gets the PrintWriter class for central file write

1. void setCenterFileW(PrintWriter centerFileW)

It sets the PrintWriter class for central file write

1. ArrayList<Integer> getUtlizationArray()

It creates an integer arraylist for the utilization

1. void setUtlizationArray(ArrayList<Integer> utlizationArray)

It sets an integer arraylist for the utilization

1. SimLogger getSingleton()

This method gets a class having only one object, providing a single point of access and doesn’t allow instantiation outside of the class.

1. void setSingleton(SimLogger singleton)

It sets the singleton object using SimLogger class

1. int[] getTotalNodesNmuberInEachLevel()

It gets total nbr of nodes in each level in an integer array.

1. void setTotalNodesNmuberInEachLevel(int[] totalNodesNmuberInEachLevel)

It sets total nbr of nodes in each level using integer array of nodes per level.

1. int[] getLevelFogNodeCount()

It gets the nbr of nodes in each Fog level in an integer array.

1. void setLevelFogNodeCount(int[] levelFogNodeCount)

It sets the nbr of nodes in each Fog level using integer array of nodes per Fog level count.

1. int[] getLevelCloudletCount()

It gets the nbr of cloud level in an integer array.

1. void setLevelCloudletCount(int[] levelCloudletCount)

It sets the nbr of cloud level using integer array of cloud level count.

1. void setFileLogEnabled(boolean fileLogEnabled)

It enables the Log File if exists.

1. void simStopped() -- App\_Types(7) :
2. AUGMENTED\_REALITY,
3. HEALTH\_APP,
4. HEAVY\_COMP\_APP,
5. INFOTAINMENT\_APP,
6. COGNITIVE\_ASSISTANCE,
7. REMOTE\_HEALTHCARE,
8. MACHINE\_LEARNING
9. **Setting the counters of the following array variables.**

int[] uncompletedTask = new int[numOfAppTypes + 1];

int[] uncompletedTaskOnCloud = new int[numOfAppTypes + 1];

int[] uncompletedTaskOnCloudlet = new int[numOfAppTypes + 1];

int[] completedTask = new int[numOfAppTypes + 1];

int[] completedTaskOnCloud = new int[numOfAppTypes + 1];

int[] completedTaskOnCloudlet = new int[numOfAppTypes + 1];

int[] failedTask = new int[numOfAppTypes + 1];

int[] failedTaskOnCloud = new int[numOfAppTypes + 1];

int[] failedTaskOnCloudlet = new int[numOfAppTypes + 1];

double[] networkDelay = new double[numOfAppTypes + 1];

double[] wanDelay = new double[numOfAppTypes + 1];

double[] lanDelay = new double[numOfAppTypes + 1];

double[] serviceTime = new double[numOfAppTypes + 1];

double[] serviceTimeOnCloud = new double[numOfAppTypes + 1];

double[] serviceTimeOnCloudlet = new double[numOfAppTypes + 1];

double[] processingTime = new double[numOfAppTypes + 1];

double[] processingTimeOnCloud = new double[numOfAppTypes + 1];

double[] processingTimeOnCloudlet = new double[numOfAppTypes + 1];

double[] cost = new double[numOfAppTypes + 1];

int[] failedTaskDuetoBw = new int[numOfAppTypes + 1];

int[] failedTaskDuetoLanBw = new int[numOfAppTypes + 1];

int[] failedTaskDuetoWanBw = new int[numOfAppTypes + 1];

int[] failedTaskDuetoMobility = new int[numOfAppTypes + 1];

int[] rejectedTaskDueToVmCapacity = new int[numOfAppTypes + 1];

int[] rejectedTaskDueToLackofNodeCapacity = new int[numOfAppTypes+1];

int[] rejectedTaskDueToLackofNetworkBandwidth = new int[numOfAppTypes+1];

int[] rejectedTaskDueToUnacceptableLatency = new int[numOfAppTypes+1];

double[] totalDist = new double[numOfAppTypes + 1];

double[] totalUserDist = new double[numOfAppTypes +1];

int[] totalHops = new int[numOfAppTypes + 1];

int[] totalHopsBack = new int[numOfAppTypes + 1];

int[] numTasksPerAppType = new int[numOfAppTypes + 1];

double[] fogLayerAvgMipsUtil = {0, 0, 0, 0, 0, 0, 0}; // Shaik added

double[] fogLayerTotalMipsUtil = {0, 0, 0, 0, 0, 0, 0}; // Shaik added

double[] fogLayerEntryMipsCount = {0, 0, 0, 0, 0, 0, 0}; // Shaik added

double[] fogLayerAvgNwUtil = {0, 0, 0, 0, 0, 0, 0}; // Shaik added

double[] fogLayerTotalNwUtil = {0, 0, 0, 0, 0, 0, 0}; // Shaik added

double[] fogLayerEntryNwCount = {0, 0, 0, 0, 0, 0, 0}; // Shaik added

1. **open files and prepare for writing into the following log files –**

\_SUCCESS.log ; \_

\_FAIL.log ;

\_VM\_LOAD.log ;

\_HOST\_MIPS\_UTILIZATION.log ;

\_HOST\_NETWORK\_UTILIZATION.log ;

\_LOCATION.log ;

\_DISTANCES.log ;

\_DISTANCES\_BACK.log ;

\_HOPS.log ;

\_HOPS\_BACK.log ;

\_NUMHOSTS.log ;

\_NUMMSGS.log ;

\_NUMPUDDLES.log ;

\_ENERGY\_USAGE.log ;

ALL\_APPS\_GENERIC.log;

\_GENERIC.log ;

"CLAYSTESTFILE" + "\_" + i + ".log"; for(int i = 0; i < numOfAppTypes; i++)

Print warm up tasks and extract the result of each task and write it to the file if required

Track the number of successfully COMPLETED tasks

Track additional metrics per task.

If task is COMPLETED

Get info to calculate 'Average distance to host'

Get info to calculate 'Average number of hops to host'

Calculate – cost, serviceTime, networkDelay, processingTime,

if VmType = CLOUD\_VM

calculate -- wanDelay, serviceTimeOnCloud, processingTimeOnCloud

else

lanDelay, serviceTimeOnCloudlet, processingTimeOnCloudlet;

If task is REJECTED\_DUE\_TO\_VM\_CAPACITY, count them.

If task is REJECTED\_DUE\_TO\_LACK\_OF\_NODE\_CAPACITY, count them.

If task is REJECTED\_DUE\_TO\_BANDWIDTH or UNFINISHED\_DUE\_TO\_BANDWIDTH, count them.

if VmType = CLOUD\_VM

count failedTaskDuetoWanBw

else

count failedTaskDuetoLanBw

If task is REJECTED\_DUE\_TO\_LACK\_OF\_NETWORK\_BANDWIDTH, count them.

If task is REJECTED\_DUE\_TO\_UNACCEPTABLE\_LATENCY, count them

If task is UNFINISHED\_DUE\_TO\_MOBILITY, count them.

If task execution is left incomplete

if VmType = CLOUD\_VM

count uncompletedTaskOnCloud

reduce failedTaskOnCloud

else

count uncompletedTaskOnCloudlet

reduce failedTaskOnCloudlet

1. **Calculate the Sum of all those array variables declared in #A above.**
2. Calculate **server load (not for HAFA).**
3. Calculate Average **Mips utilization** of all fog nodes including capture metrics per layer.
4. Calculate Average **node utilization** **per fog layer**
5. Calculate Average **fog node utilization per layer**
6. Calculate Average **fog network utilization per layer**
7. Write location info of all mobile devices to file
8. Calculate averages for various metrics

**Following genericResult (1 thru 10, #K thru #V) are created** **for (int i = 0; i < numOfAppTypes ; i++)**

1. Write **genericResult1** using -- completedTask[i]

, failedTask[i]

, uncompletedTask[i]

, failedTaskDuetoBw[i]

, \_serviceTime

, \_processingTime

, \_networkDelay

, \_vmLoad

, \_cost

, rejectedTaskDueToVmCapacity[i]

, failedTaskDuetoMobility[i]

, \_fnMipsUtil

, \_fnNwUtil

, rejectedTaskDueToLackofNodeCapacity[i]

, rejectedTaskDueToLackofNetworkBandwidth[i]

, rejectedTaskDueToUnacceptableLatency[i])

, failedTask[numOfAppTypes]

, completedTask[numOfAppTypes])

followed by DELIMITED string.

1. Write **genericResult8** using -- failedTask[numOfAppTypes]

, completedTask[numOfAppTypes]

, failedTaskDuetoMobility[i])

, rejectedTaskDueToLackofNodeCapacity[i]

, rejectedTaskDueToLackofNetworkBandwidth[i] , rejectedTaskDueToUnacceptableLatency[i]

followed by DELIMITED string.

1. Write **genericResult2** using – completedTaskOnCloudlet[i]

, SimSettings.DELIMITER

, failedTaskOnCloudlet[i]

, uncompletedTaskOnCloudlet[i]

, failedTaskDuetoLanBw[i]

, \_serviceTimeOnCloudlet

, \_processingTimeOnCloudlet

, \_lanDelay;

followed by DELIMITED string.

1. Write **genericResult3 using --** completedTaskOnCloud[i]

, failedTaskOnCloud[i]

, uncompletedTaskOnCloud[i]

, failedTaskDuetoWanBw[i]

, \_serviceTimeOnCloud

, \_processingTimeOnCloud

, \_wanDelay;

followed by DELIMITED string.

1. Write **genericResult4** using -- distance

,hops

,avgNumHosts

,avgNumMsgs

,avgNumPuds;

followed by DELIMITED string.

1. Tasks executed per fog layer.

**Write genericResult5** using the following loop.

for(int level = 1; level <= getMaxLevels(); level++) {

genericResult5 += levelCloudletCount[level]) + DELIMITED string ;

}

1. Average fog node utilization per layer

**Write genericResult6** using following loop.

for (int index = 0; index < fogLayerAvgMipsUtil.length; index++) {

genericResult6 += fogLayerAvgMipsUtil[index]) + DELIMITED string ;

}

1. Average fog network utilization per layer

**Write genericResult7** using following loop.

for (int index = 0; index < fogLayerAvgNwUtil.length; index++) {

genericResult7 += fogLayerAvgNwUtil[index]) + DELIMITED string ;

}

1. Write **genericResult9** using -- distBack + by DELIMITED string + hopsBack

1. CalculateTotalIdleEnergy() and then followings are written ---

getTotalEnergy()),

getTotalRouterEnergy()),

getTotalFogNodeEnergy()),

getIdleEnergy())

followed by DELIMITED string.

1. Write **genericResult10 using --**

getTotalEnergy()),

getTotalRouterEnergy()),

getTotalFogNodeEnergy()).

followed by DELIMITED string.

1. Finally all these generericResults (1 thru 10) are written in genericBWs file
2. Print Energy\_Usage file using –

getTotalEnergy()),

getTotalRouterEnergy()),

getTotalFogNodeEnergy())

getIdleEnergy()

1. Prints -- "# of failed tasks due to vm capacity/LAN bw/mobility: "

Prints -- "# of failed tasks due to lack of node capacity/lack of network bandwidth/unacceptable latency: "

Prints -- "Submitted tasks: "

Prints – “Percentage of failed tasks: "

Prints -- "Average service time: "

Prints -- "Tasks executed per fog layer: "

Prints --"Average cost: $"

Prints -- "Processing Time: "

Prints --"Average Distance from task to host: "

Prints -- "Average Distance from host to user: "

Prints -- "Average number of hops from task to host: "

Prints -- "Average number of hops from host to user: "

Prints -- "Average number of prospective hosts considered for placement: "

Prints -- "Average number of messages exchanged for placement: "

Prints --"Average number of Puddles searched for placement: "

Prints --"Percentage of fog nodes executing atleast one task:"

Prints --"Level " + (i + 1) ": I = numOfAppTypes;

Prints -- "Average fog node utilization per layer:");

Prints --"Average fog network utilization per layer:");

Prints --"Average Fog server utilization: "

Prints --"Average Fog network utilization: "