CloudSim 3.0.3 Simulator Step by Step

Experiment Findings · March 2021

DOI: 10.13140/RG.2.2.17891.48161

4 authors, including:

Dalia Abdulkareem Shafiq
Taylor's University
15 PUBLICATIONS **471** CITATIONS
SEE PROFILE

Noor Zaman Jhanjhi
Taylor's University
632 PUBLICATIONS **14,220** CITATIONS
SEE PROFILE

Azween Abdullah
Taylor's University
12 PUBLICATIONS **25** CITATIONS
SEE PROFILE

CloudSim 3.0.3

The Manual

CloudSim Simulation Tool is the most popular simulator used by researchers and developers nowadays for the cloud-related issues in the research field. This manual will ease your learning by providing simple steps to follow up with installing and understanding this simulation tool.

**This manual is intentionally prepared to help the research community who are working in Cloud Computing domain.*

***Prepared By**: Dalia Abdulkareem Shafiq*
***Supervised By**: Noor Zaman Jhanjhi, Azween Abdullah*

Contact Details:

daliakareem7@gmail.com; noorzaman.jhanjhi@taylors.edu.my; azween.abdullah@taylors.edu.my

School of Computer Science & Engineering (SCE), Taylor's University, Malaysia.

**For any enquiry or help please contact Dalia Abdulkareem Shafiq.*

Index

Introduction to CloudSim

Simulation creates a virtual environment for testing and verifying research experiments for efficient and better solutions for the application. It is a scientific technique to make a model or a real-time system [1]. Thus it eliminates the need and expenses of computing facilities [2] for performance evaluation and modelling the research solution. This manual focuses mainly on CloudSim simulation tool and its benefits to researchers.

Purpose of CloudSim

This is known to be the most popular and widely used simulator for CC applications. It is known for its ability to handle large-scale platforms and combined stimulation of both private and public domains [2]. It is in the form of layered architecture as can be seen in figure 1 below.

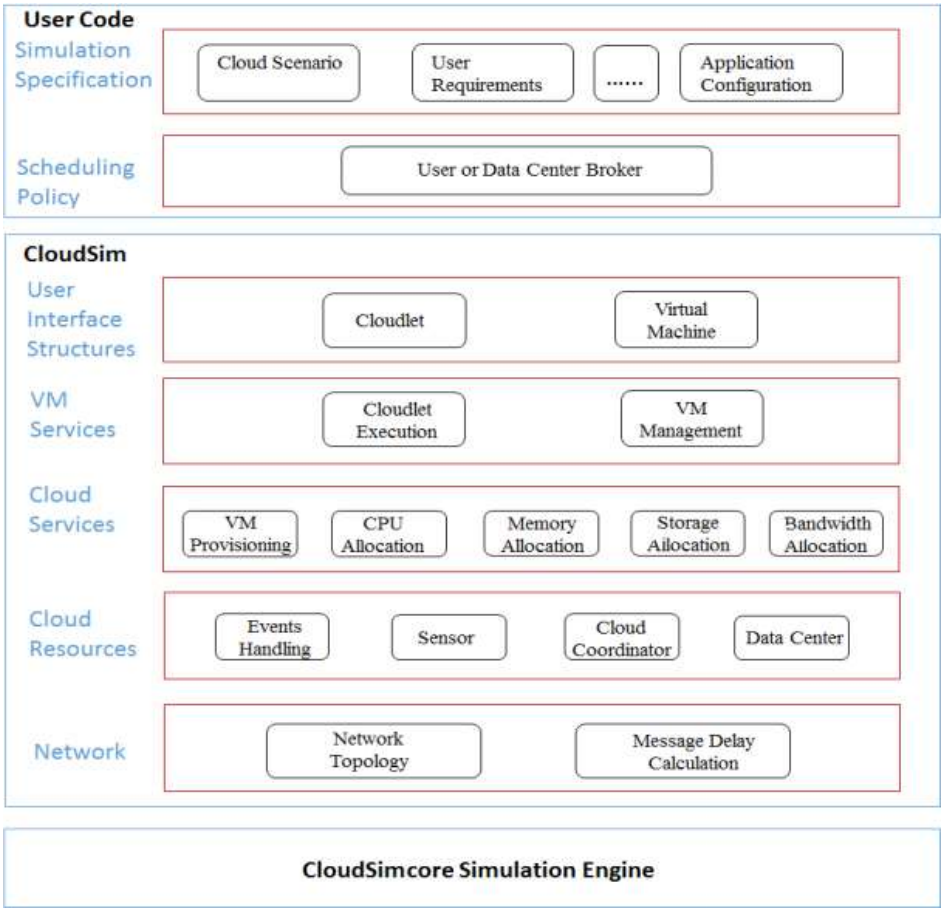


Figure 1: CloudSim Architecture [3]

CloudSim consists of three layers. The top-layer, named as the "User Code" layer which consists of the basic entities in the cloud. This is where the specifications of the simulation are defined such as no. of VMs, users [3], and the enhanced or proposed scheduling policy e.g. Round Robin, etc. In this tool, the

cloudlet is known to be the task, which is the user request. Thus, this layer provides the interface and other specifications related to the simulation experiment such as the data center location, etc. The second layer "CloudSim" also provides support to create the cloud-based environment such as implementation of user interface that includes the Cloudlet and Virtual Machines. This layer allows the users to configure the bandwidth, memory, and CPU of the cloud services. In addition, the layer can be used to simulate the network in cloud environments. The last layer is the "Simulation Engine" which is responsible for running events such as creation and communication of cloud components.

Features of CloudSim

Technologies such as cloud computing offer many services to users by accessing files online and eliminating the need of physical infrastructure. Enhancing those services have become a goal for researchers and developers worldwide. While it may be difficult for researchers to model and test such experiments, simulators such as CloudSim comes in handy. Following specific requirements, the cloud environment can easily be depicted using essential components in this simulator. Simulation can bring great benefit in the research community as it can reduce cost, effort, and time to configure the real cloud scenario. CloudSim has many features as listed below:

- Less time and effort to implement the cloud environments, especially to depict the Infrastructure as a Service (IaaS) delivery model [4] in the cloud technology.
- Allows modelling and simulation for large scale Data Centers; testing can be done for both heterogenous and homogeneous cloud environments.
- Enables developers and researchers to design and simulate energy-aware solutions.
- Provides availability to share virtualized resources by time and space.
- The toolkit can be extended to cater different users' policies in the cloud.

While CloudSim brings many benefits, it still lacks in few things such as the ability to simulate other cloud delivery models such as Platform as a Service (PaaS) and Software as a Service (SaaS). Also, it lacks in GUI as it highly relies of developer's programming skills.

Comparison with Other Tools

CloudAnalyst Similar to CloudSim, however it includes a graphical user interface (GUI) [2][5], which allows the user to separate the experiments into two exercises: program and simulation. It is a further enhancement to CloudSim whereby it provides geographical information about users as can be seen in figure 2 below in the simulation experiment. It also extracts the results in PDF or XML format [6] which makes the workload description more clear and detailed. The main benefit of this tool is to

make the simulation set up and configurations easier for users with low programming skills and for simulating the response time parameter [1]. The tool provides three load balancing policies by default such as Round Robin, Equally Spread Concurrent Execution (ESCE), and Throttled Algorithm.

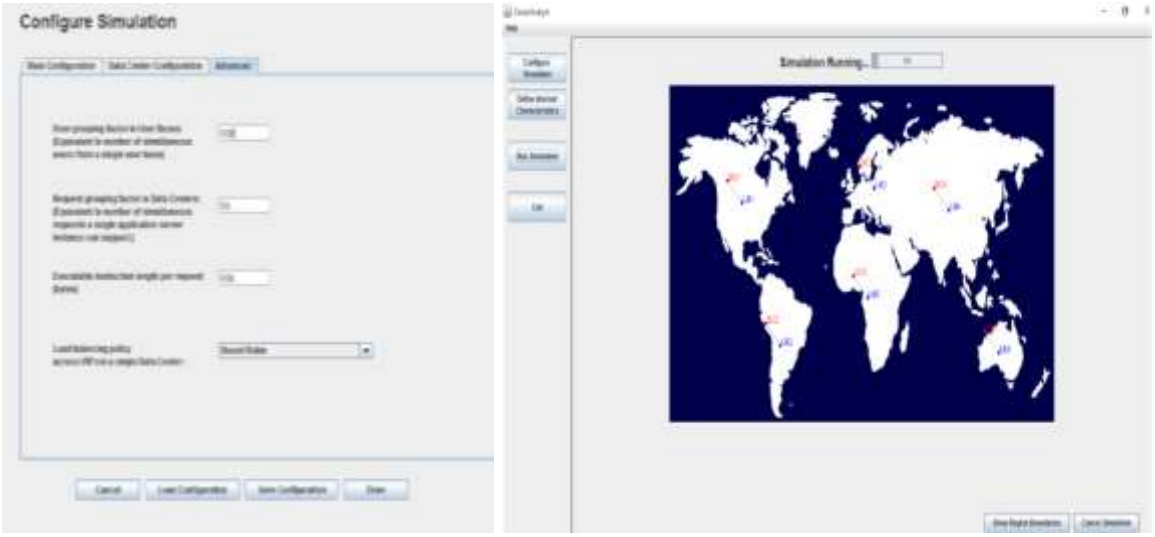


Figure 2: Advanced configuration and Geographical representation in CloudAnalyst [7]

CloudReports A simulation tool, with more focus towards the energy [6] aspect of CC. similar to CloudAnalyst, it makes simulation experiments easier for researchers with low programming skills and high repeatedly. It provides results in the form of charts which makes the results easier to understand and more presentable. This tool work best for the evaluation of cost and resource utilization and power consumption of the proposed solution [1].

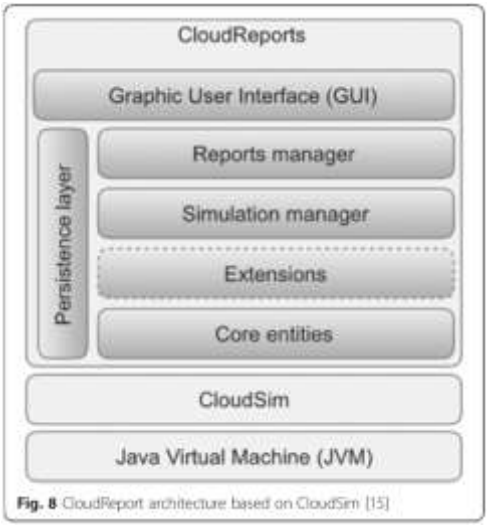


Figure 3: CloudReports Architecture [8]

In terms of testing and simulating performance parameters such as VM migration and response time CloudAnalyst is better than CloudSim, for load balancing parameter, CloudReports is not sufficient for the simulation, however, it is faster than CloudSim. All mentioned simulation tools can model and simulate the IaaS cloud environment [1].

Table 1: Comparison of Simulation Tools for Cloud Computing

Simulation Tool	**Availability**	**Programming Language**	**Graphical?**	**Result Output Format**
CloudSim	Open Source	Java	No	Text
CloudAnalyst		Java	Yes	XML, PDF
CloudReports		Java, JS	Yes	JavaScript, Text

CloudSim Packages

CloudSim offers many packages upon installation to help developers in creating a seamless simulation. These packages can be modified to allow developers and researchers to create their own policies and algorithms. The packages are listed and explained below:

Table 2: CloudSim Packages

Package	Description
org.cloudbus.cloudsim	This package is highly important in CloudSim as it includes the necessary components required to depict the cloud environment. In this package, developers can introduce their own Cloudlet Scheduler to determine policies to divide the available CPU resources of the VMs among Cloudlets. By default, the resources are shared in two ways: 1) Space-Shared (***CloudletSchedulerSpaceShared.java***): in this his policy a cloudlet is executed on a VM and once it's executed successfully and release the resources, another cloudlet can run on the VM. 2) Time-Shared (***CloudletSchedulerTimeShared.java***) : this method allows for dynamic distribution of the capacity of a CPU among VMs. Example: if two cloudlets are running parallel and a VM has a CPU of 1000 MIPS, each cloudlet will get half of the MIPS, giving both cloudlets same amount of time for execution. In this package, developers can modify the broker class, create a new cloudlet class to add specific parameters for object-oriented simulation.
org.cloudbus.cloudsim.core org.cloudbus.cloudsim.core.predicates	The core package is required to determine the events of the simulation, for example it include the CloudSim

	shutdown which waits for all entities to terminate and then end the simulation. Predicates are used to select events from the deferred queue.
org.cloudbus.cloudsim.distributions	This package consists of mathematical distributions such as Lomax Distribution, to distribute probability. Other distributions include Random Number Generator, Log normal, Exponential Distribution, Gamma Distribution, etc.
org.cloudbus.cloudsim.examples org.cloudbus.cloudsim.examples.network org.cloudbus.cloudsim.examples.network.datacenter org.cloudbus.cloudsim.examples.power org.cloudbus.cloudsim.examples.power.planetlab org.cloudbus.cloudsim.examples.power.random	CloudSim provides examples to ease the understanding of developers in these packages. the examples explain the basic concept for creating the cloud components and running them. The examples are written with the main() method and ready for execution.
org.cloudbus.cloudsim.lists	Components such as cloudlets, VMs, hosts, PeS should be stored in lists. This package provides a collection of operations on that can be done on such lists. For example, it can sort the cloudlets based on their length, find its position etc.
org.cloudbus.cloudsim.network org.cloudbus.cloudsim.network.datacenter	This package includes classes that are necessary for network-based simulation. It deals with network topology. While the network Data Center package consists of classes to define a network within a Data Center. This package deals with switches.
org.cloudbus.cloudsim.power org.cloudbus.cloudsim.power.lists org.cloudbus.cloudsim.power.models	This package deals with power-aware Data Center policies. This package can be extended by developers to introduce their own policies. The lists packages include collection of operations that can be done on VMs list that are power-enabled. The models can be extended by developers based on the system utilization.
org.cloudbus.cloudsim.provisioners	This package is essential as it defines the provisioning policy for Virtual Machines running on a host. Example

	it defines the bandwidth (BW), the available MIPS (PE), and the memory (RAM).
org.cloudbus.cloudsim.util	This package consists of classes related to specific mathematical functions. For example, MathUtil in java can be used to imply multiple functions such as Sum, absolute value and so on. It also provides a measure for the execution time.
workload.planetlab.20110303	Such package can act as a dataset in simulation as it includes PlanteLab Virtual Machine workload trace.

Creating Simple Cloud Components

There are four vital simulation components required to implement and depict the cloud environment. In a typical cloud environment, there are four essential entities that must be created to depict the virtual infrastructure of cloud computing concept which are the Cloudlets, Virtual Machines, Data Center and broker. The figure below illustrates how these components can interact with each other in a typical cloud environment.

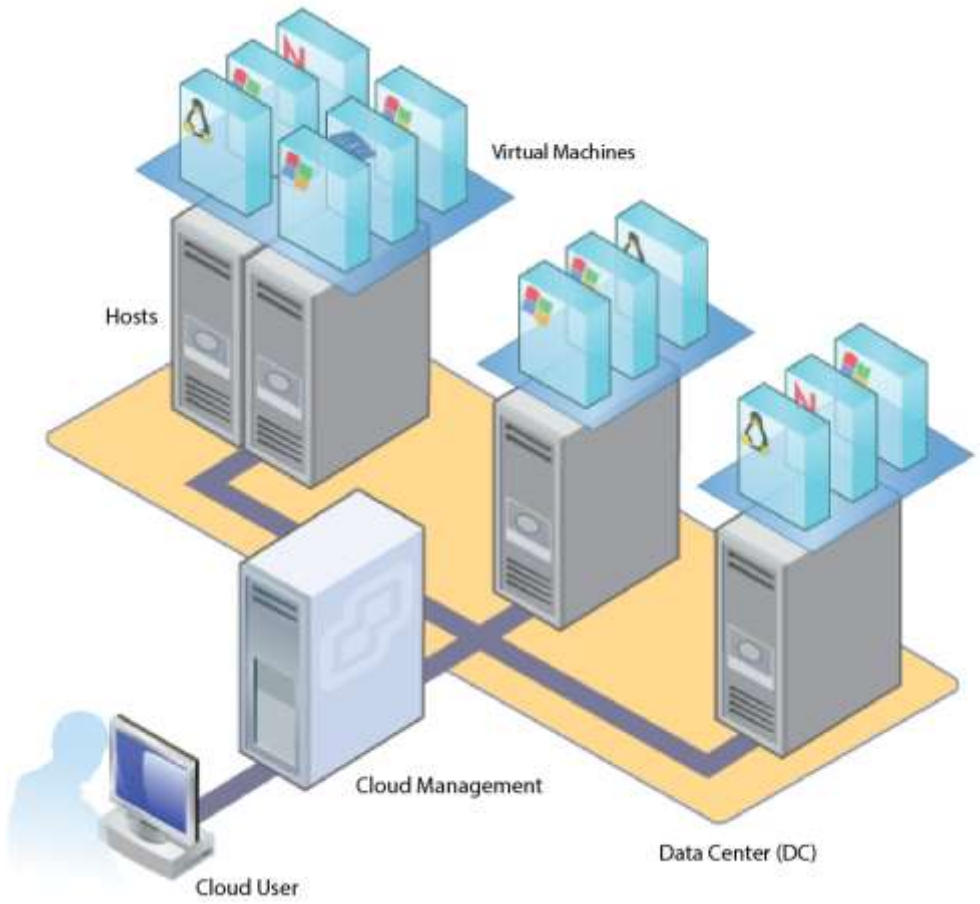


Figure 4: Cloud Components

The cloud user sends requests (cloudlets) through the cloud management to the Data Center (DC). The purpose of the cloud management is to distribute these requests and assign resources to users efficiently. In the DC, there can be more than 1 host, and, in each host, there can be more than one virtual machine. All the components are necessary to perform simulation.

The above cloud entities are illustrated in the figure 5 below which shows their interaction and relationship with each other.

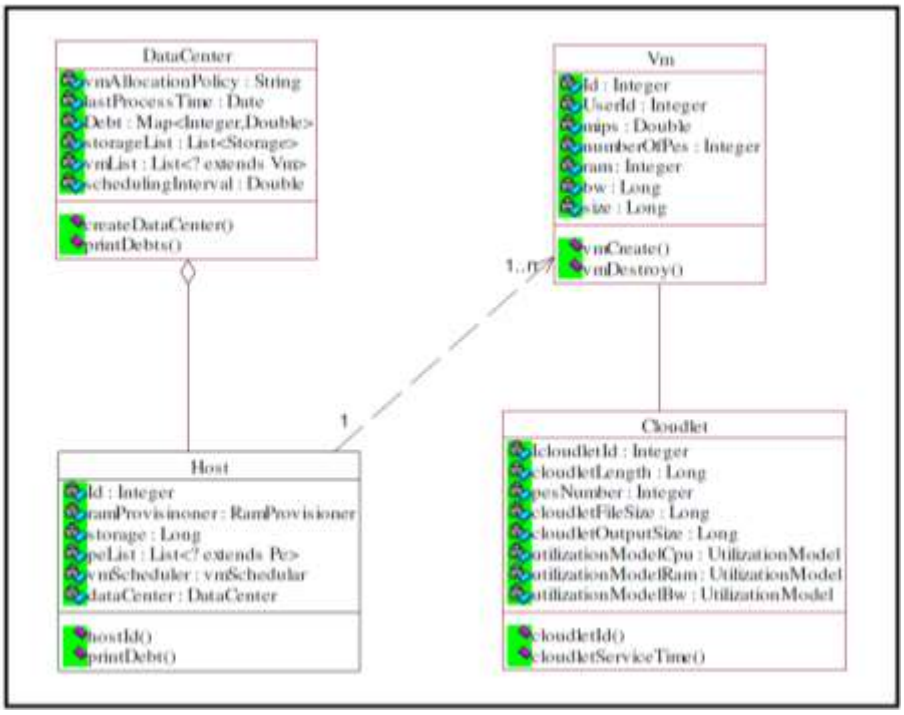


Figure 5: Class Diagram of CloudSim Components

In a typical simulation environment such as in CloudSim tool, there are four main components as listed below:

1. **Data Centre (DC)**: this represents the cloud computing environment. This where a list of Hosts will be created with different configurations for each host such as bandwidth, memory, storage, and MIPS. It is facility to house the physical machines which are converted to virtual machines in the cloud environment. The DC can have multiple hosts.
2. **Hosts**: 1 host can have 1 to many Virtual Machines as can be seen in the figure above. The number of hosts needs to be increased several times through testing of performance.
3. **Virtual Machines**: used to store tasks allocated to them. A fair distribution of tasks to VMs should be performed to provide a balanced load among the hosts. More than 1 VM is required for performance testing.
4. **Cloudlets**: these act as the application requests (incoming user tasks). Dependent on the proposed algorithm, different parameters will be considered for this such as deadline, length (size), and arrival time. cloudlets are used to represent user requests.

Below are the steps required to create these components along with a code snippet in Java language:

Step 1: *Initializing the CloudSim Package & Library*

This step is created in the `main()` method which is indicates this where the execution of the simulation starts. First the number of users is set. This number indicates the users in the current experiment, 1 user means there is 1 broker. Hence, this count is directly proportional to the number of brokers implemented. The `getInstance()` method is used to with a `Calendar` object to initialize the current time zone set by the java runtime environment. Hence, it indicates the starting time of the simulation. If it is null, then the time will be automatically taken from `Calendar.getInstance();` Lastly, we need to specify whether it is required to trace the simulation events or not. If the `trace_flag` is set to the CloudSim trace needs to be written.

```
// First step: Initialize the CloudSim package. It should be called
before creating any entities.
int num_user = 1;   // number of cloud users
Calendar calendar = Calendar.getInstance();
boolean trace_flag = false;  // mean trace events

// Initialize the CloudSim library
CloudSim.init(num_user, calendar, trace_flag);
```

Step 2: *Creating a Data Center*

A separate method is required to create a Data Center. Firstly, two lists are created to store the hosts and the cores (CPUs). More than 1 CPU can be created for a single machine with the help of `ArrayList<>` concept. The available MIPS for this host is initialized and characteristics of host and Data Center is created based on the developer's preference. Each host's PE must have its own instance of a `PeProvisioner`, if this class is extended, developers must guarantee that there's available MIPS in case of future allocations.

```
private static Datacenter createDatacenter(String name){

// Here are the steps needed to create a PowerDatacenter:
// 1. We need to create a list to store our machine
List<Host> hostList = new ArrayList<Host>();

// 2. A Machine contains one or more PEs or CPUs/Cores.
// In this example, it will have only one core.
List<Pe> peList = new ArrayList<Pe>();

int mips = 1000;
```

```
// 3. Create PEs and add these into a list.
peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id
 and MIPS Rating
//4. Create Host with its id and list of PEs and add them to the list of mac
hines
int hostId=0;
int ram = 2048; //host memory (MB)
long storage = 1000000; //host storage
int bw = 10000;

hostList.add(
      new Host(
            hostId,
            new RamProvisionerSimple(ram),
            new BwProvisionerSimple(bw),
            storage,
            peList,
            new VmSchedulerTimeShared(peList)
      )
); // This is our machine.

// 5. Create a DatacenterCharacteristics object that stores the
//    properties of a data center: architecture, OS, list of
//    Machines, allocation policy: time- or space-shared, time zone
//    and its price (G$/Pe time unit).
String arch = "x86";      // system architecture
String os = "Linux";          // operating system
String vmm = "Xen";          // hypervisor
double time_zone = 10.0;         // time zone this resource located
double cost = 3.0;          // the cost of using processing in this resource
double costPerMem = 0.05;       // the cost of using memory in this resource
double costPerStorage = 0.001; // the cost of using storage in this resource
double costPerBw = 0.0;             // the cost of using bw in this resource
LinkedList<Storage> storageList = new LinkedList<Storage>();    //we are not
 adding SAN devices by now

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPe
rBw);

// 6. Finally, we need to create a PowerDatacenter object.
Datacenter datacenter = null;
try {
      datacenter = new Datacenter(name, characteristics, new VmAllocationPo
      licySimple(hostList), storageList, 0);
      } catch (Exception e) {
      e.printStackTrace();
```

```
        }

        return datacenter;
}
```

Step 3: *Creating a Data Center Broker*

A broker acts as an intermediary broker between the cloud users and the Cloud Service Providers (CSPs). It is responsible to route user requests to the most appropriate Data Center.

```
private static DatacenterBroker createBroker(){
        DatacenterBroker broker = null;
        try {
                broker = new DatacenterBroker("Broker");
        } catch (Exception e) {
                e.printStackTrace();
                return null;
        }
        return broker; }
```

The output below shows that the simulation experiment has started with the CloudSim version, the Data Center has been initiated along with its broker and finally the necessary entities such as VMs and cloudlets are started and ready to be created.

Output (Step 1 – 3)

```
Starting CloudSim version 3.0
Datacenter_0 is starting...
Broker is starting...
Entities started.
```

Step 4: *Creating a Virtual Machine*

A Virtual Machine acts as a resource that should be utilized efficiently in simulation. First, a list to store the Virtual Machines is created, then the characteristics are initiated and finally the list is submitted to a broker.

```
//Fourth step: Create one virtual machine
vmlist = new ArrayList<Vm>();

//VM description
int vmid = 0;
int mips = 250;
long size = 10000; //image size (MB)
int ram = 512; //vm memory (MB)
```

```
long bw = 1000;
int pesNumber = 1; //number of cpus
String vmm = "Xen"; //VMM name

//create two VMs
Vm vm1 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new Clo
udletSchedulerTimeShared());

//add the VM to the vmList
vmlist.add(vm1);

//submit vm list to the broker
broker.submitVmList(vmlist);
```

The output below illustrates that the program has successfully created six of virtual machines (VMs) and describes which VM by including its ID is submitted to which Data Center on a particular host with the help of the Data Center broker.

Output (Step 4)

```
0.0: Broker: Cloud Resource List received with 1 resource(s)
0.0: Broker: Trying to Create VM #1 in Datacenter_0
0.0: Broker: Trying to Create VM #2 in Datacenter_0
0.0: Broker: Trying to Create VM #3 in Datacenter_0
0.0: Broker: Trying to Create VM #4 in Datacenter_0
0.0: Broker: Trying to Create VM #5 in Datacenter_0
0.0: Broker: Trying to Create VM #6 in Datacenter_0
0.1: Broker: VM #1 has been created in Datacenter #2, Host #0
0.1: Broker: VM #2 has been created in Datacenter #2, Host #0
0.1: Broker: VM #3 has been created in Datacenter #2, Host #0
0.1: Broker: VM #4 has been created in Datacenter #2, Host #0
0.1: Broker: VM #5 has been created in Datacenter #2, Host #0
0.1: Broker: VM #6 has been created in Datacenter #2, Host #0
```

Step 5: *Creating a Cloudlet*

A cloudlet depicts the user requests/jobs submitted in the simulation. First a list of cloudlets is created then the properties are initiated and finally the list is submitted to the broker.

```
//Fifth step: Create two Cloudlets
cloudletList = new ArrayList<Cloudlet>();

//Cloudlet properties
int id = 0;
pesNumber=1;
long length = 250000;
long fileSize = 300;
long outputSize = 300;
```

```
UtilizationModel utilizationModel = new UtilizationModelFull();

Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSiz
e, utilizationModel, utilizationModel, utilizationModel);
cloudlet1.setUserId(brokerId);

//add the cloudlets to the list
cloudletList.add(cloudlet1);

//submit cloudlet list to the broker
broker.submitCloudletList(cloudletList);
```

Step 6: *Bind cloudlets to Virtual Machines.*

It is always important to remember to bind cloudlets to VMs to make sure they are submitted to specific VMs as seen below.

```
//bind the cloudlets to the vms. This way, the broker
// will submit the bound cloudlets only to the specific VM
broker.bindCloudletToVm(cloudlet1.getCloudletId(),vm1.getId());
broker.bindCloudletToVm(cloudlet2.getCloudletId(),vm2.getId());
```

The output below shows cloudlets are being sent to VMs for execution and describes which cloudlets by including their IDs are submitted to which virtual machine.

Output

```
0.1: Broker: Sending cloudlet 0 to VM #0
400.1: Broker: Cloudlet 0 received
400.1: Broker: All Cloudlets executed. Finishing...
```

Step 7: *Running/stopping simulation & creating a new list of the submitted cloudlets to VMs*

After creation of the components above, the simulation is started. After the execution of tasks is completed a new list should be created to store the cloudlets received by the broker.

```
// Sixth step: Starts the simulation
CloudSim.startSimulation();

// Final step: Print results when simulation is over
List<Cloudlet> newList = broker.getCloudletReceivedList();
CloudSim.stopSimulation();
printCloudletList(newList);
```

The output below shows VMs are being destroyed after the completion of task execution and the broker is shutting down which indicates the end of the simulation.

Output

```
400.1: Broker: Destroying VM #0
Broker is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Broker is shutting down...
Simulation completed.
```

Installing CloudSim With NetBeans IDE 8.2

1. Before installing CloudSim tool, we need to install the following resources on a local system:

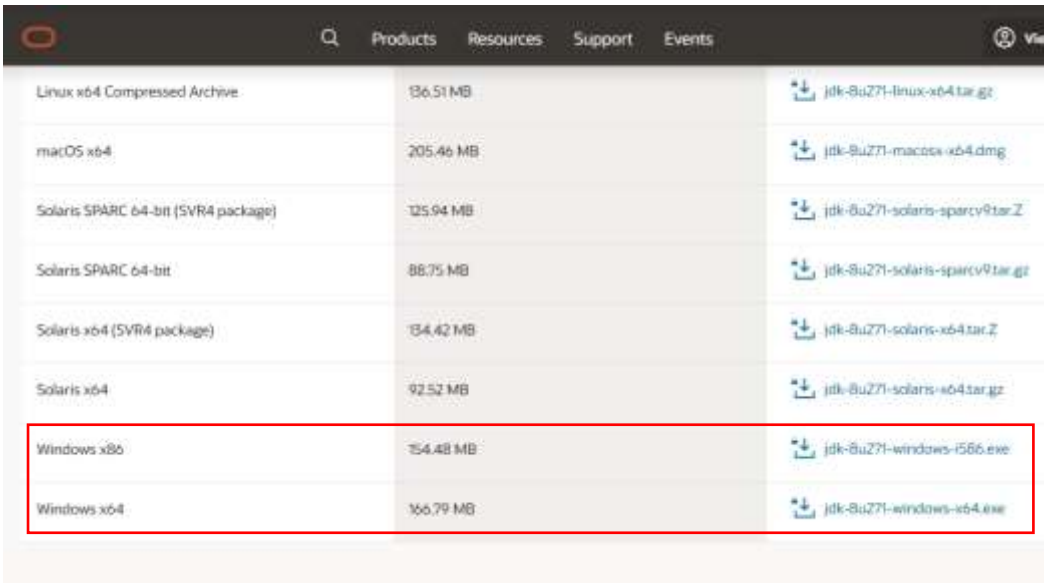
   a. **NetBeans IDE 8.2** for java developers:

   Follow this link to download as seen below: https://netbeans.org/downloads/old/8.2/

   b. **Java Development Kit (JDK)** - Java SE Development Kit 8u271 version is recommended:

   Follow this link to download as seen below (*either Windows x86 or x64 depending on your system*):

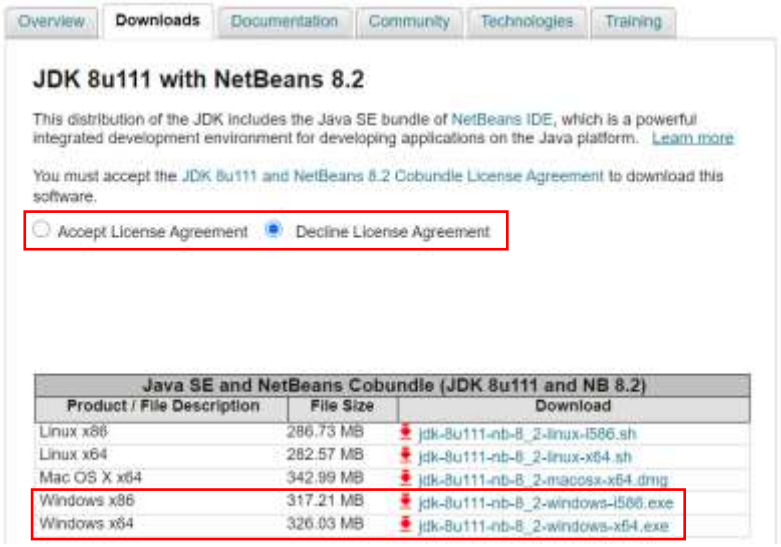
   https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html



   c. Another option is to download NetBeans with JDK preinstalled in it:

   Follow this link to download as seen below (make sure to accept the License agreement first):

https://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-3413139-esa.html



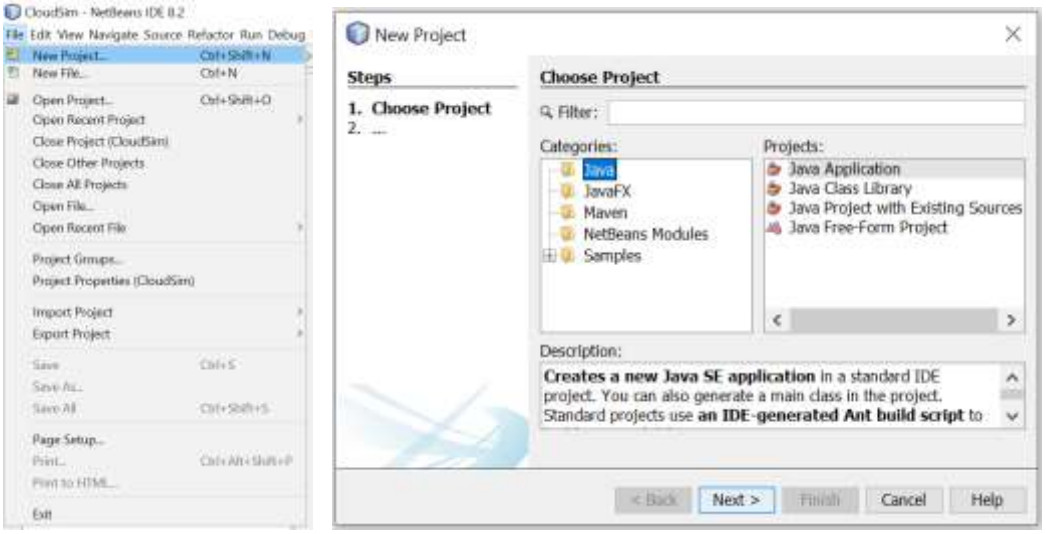
d. CloudSim *(3.0.3 version & above)* Project code:

Follow this link to download as seen below: http://www.cloudbus.org/cloudsim/

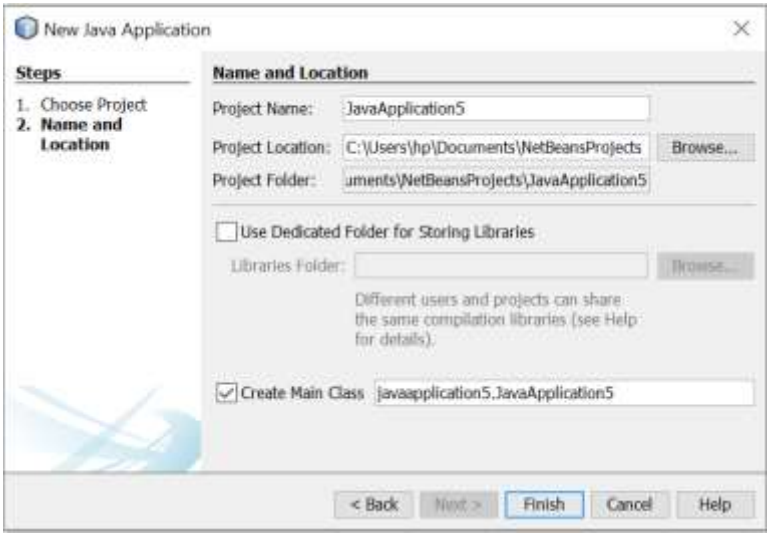
2. Unzip **NetBeans** and **CloudSim** to same destination folder on the local system. *(**Example**: Desktop or Documents)*

3. To install NetBeans: Find the installer executable file (jdk-8u111-nb-8_2-windows-x64.exe) and double-click the installer file to run it.

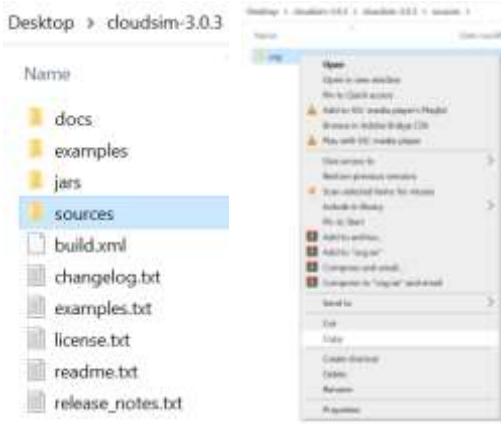
4. How to install CloudSim:
   a. Open NetBeans, go to File → New Project → Choose Java, then Java Application as seen below:



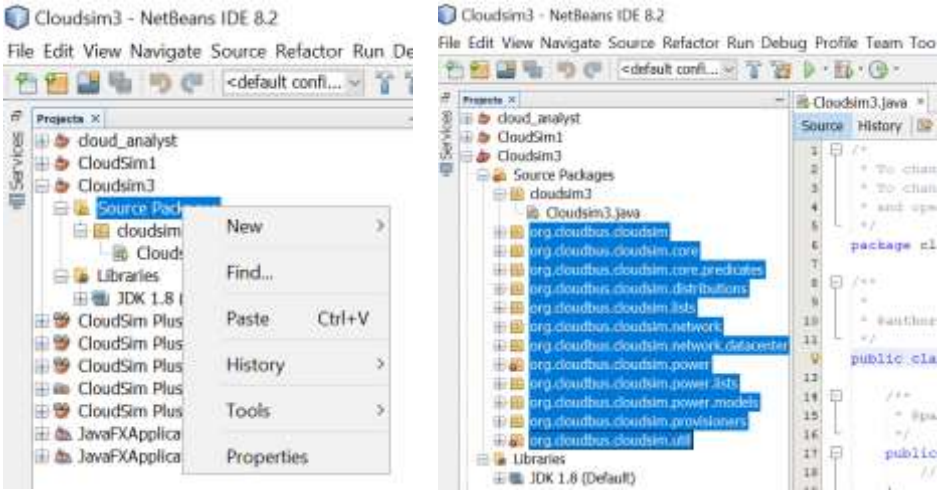
   b. Choose the project name and location (E.g., Project name: CloudSim3 and Project Location: NetBeansProjects), then click “Finish”.



   c. Find the CloudSim folder downloaded in step 1 (d) → go to Sources folder → copy the “org” and folder as seen below.

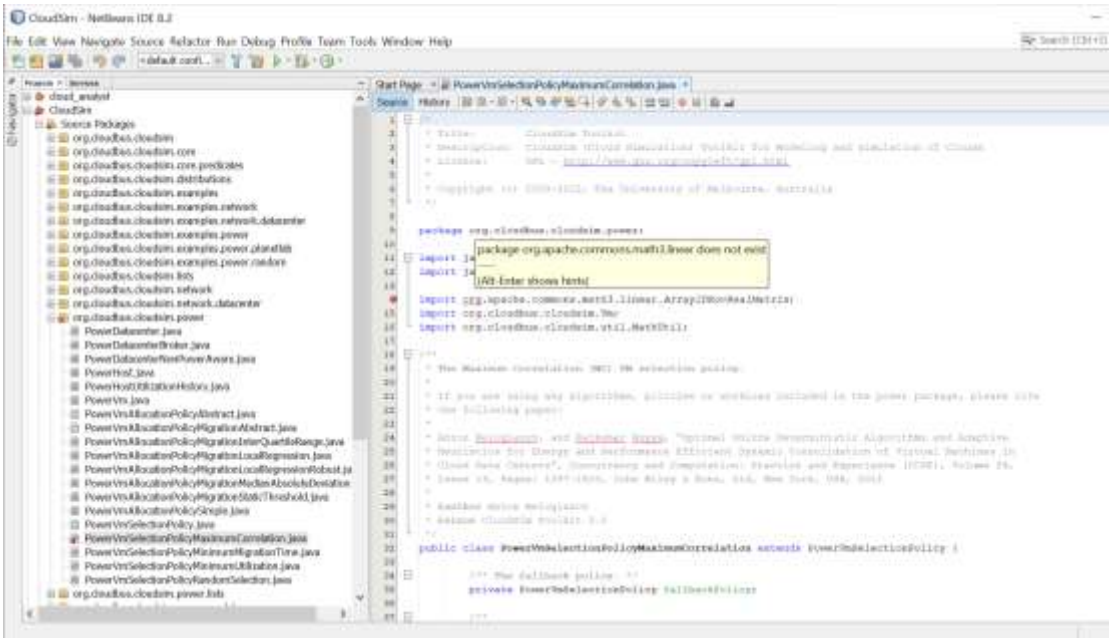


d. Go back to NetBeans window → right click on the "Source Packages", then choose Paste as seen below.



Repeat the step to copy and paste the "org" folder from "Examples" folder.

5. Upon installing CloudSim, you may receive errors such as missing libraries import as seen below:

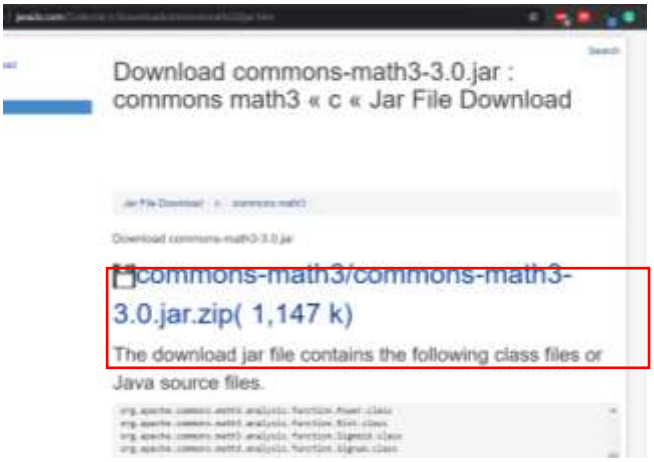


Here is how to resolve it:

a. First you need to download the missing jar file "commons math3":

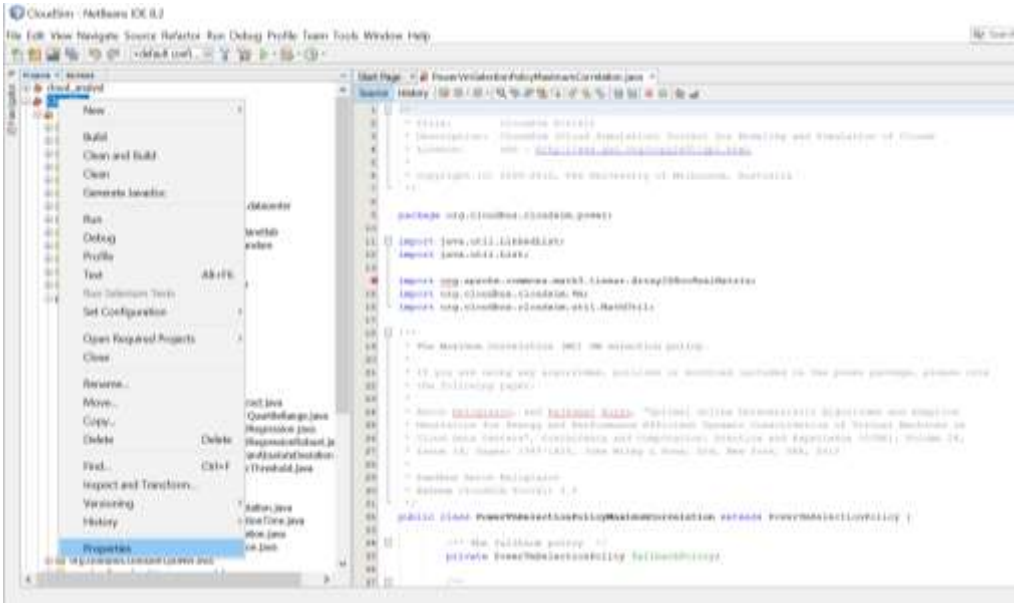
Follow this link to download as seen below:

http://www.java2s.com/Code/Jar/c/Downloadcommonsmath330jar.htm

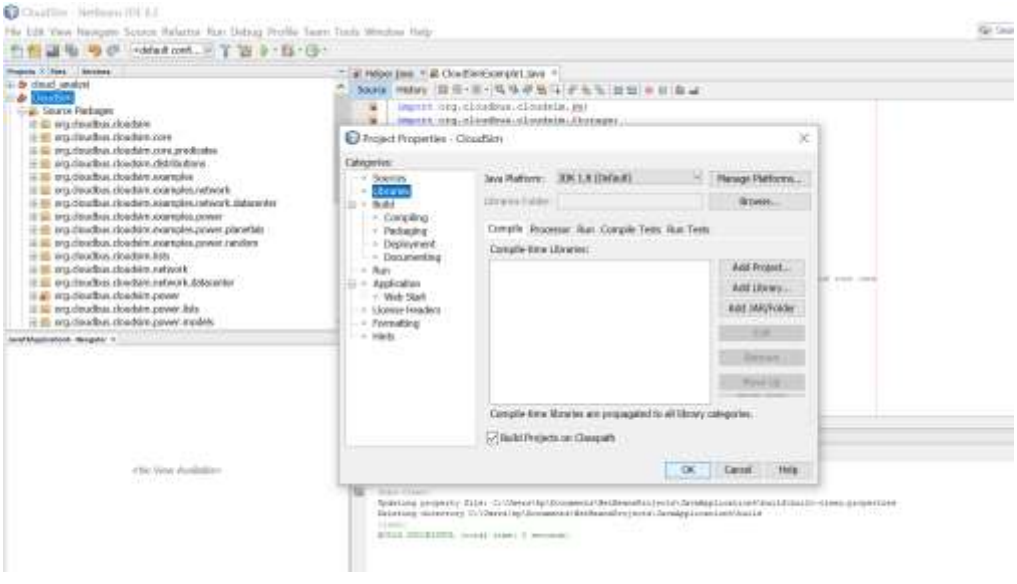


b. Unzip the folder & place it as you desire in the local system (recommended outside the project file)

c. Go to NetBeans, right-click on the "CloudSim" project, choose Properties from the menu as seen below:



d. Click on Libraries from the categories, then choose Add JAR/Folder from the right



Creating a Dataset with CloudSim 3.0.3

In research, datasets are often required for richer outcome of results. While this feature is not built in the CloudSim tool, it still can be achieved using File concept in Java language. To demonstrate this, CloudSimExample1.java source code will be used. The source code is found from the packages extracted in step 4 (d) above.

Purpose of the example 1: to show how a Data Center is created with one host. To demonstrate how to save the output in files, we ran two cloudlets on it instead of one using the CloudSim tool.

Required Import libraries: the following imports are essential to create a dataset from the simulation results of example 1 in NetBeans. The libraries are required for file creation, writing to file, and checking of errors.

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
```

Class Components:

Variables: the following variables are important to store the output of the program in rows. For this purpose, ArrayList concept is used. Both of these rows will consist of 7 columns; hence the range of values (size of the array) is 7 as seen below.

```
//the first and second rows consists of 7 columns.
private static double[] Row1 = new double[7];
private static double[] Row2 = new double[7];
```

Method: the below method is required to write the results of the simulation using ArrayList concept. Note that it is important to close the file and writer with the close() method to ensure successful writing and no further modifications can be done to the files.

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

// Create file (example .txt or .csv)
try {
        File results = new File("Output.txt");
                if (results.createNewFile()) {
                System.out.println("File created: " + results.getName());
```

```
                } else {
                System.out.println("File already exists.");
                results.delete();
                }
        }
catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
}
```

Full Source code: the source code below is to show how the results of example 1 is saved in an external file.

```
package org.cloudbus.cloudsim.examples;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

/**
```

```
 * A simple example showing how to create a datacenter with one host and run
 one
 * cloudlet on it.
 */
public class writeresults {

       /** The cloudlet list. */
        private static List<Cloudlet> cloudletList;

        /** The vmlist. */
        private static List<Vm> vmlist;

         //the first and second rows in excel consists of 7 columns.
         private static final double[] Row1 = new double[7];
         private static final double[] Row2 = new double[7];

        /**
        * Creates main() to run this example.
        *
        * @param args the args
        */
        @SuppressWarnings("unused")
        public static void main(String[] args) {

        Log.printLine("Starting CloudSimExample1...");

         // Create file (example .txt or .csv)
         try {
             /** specific location:
File file = new File("C:\\Users\\hp\\Documents\\NetBeansProjects\\Cloudsim3\
\Output2.csv"); */
             //automatically the file is created in the project folder
             File file = new File("Output2.csv");
             if (file.createNewFile()) {
                 Log.printLine("File created: " + file.getName() + "\n");
             } else {
                 Log.printLine("Deleting/overwritting file...\n");
                 file.delete();
             }
         } catch (IOException e) {
                 System.out.println("An error occurred.");
         }
         try {
              // First step: Initialize the CloudSim package. It should be ca
              lled
              // before creating any entities.
               int num_user = 1; // number of cloud users
               Calendar calendar = Calendar.getInstance();
```

```
boolean trace_flag = false; // mean trace events

// Initialize the CloudSim library
CloudSim.init(num_user, calendar, trace_flag);

// Second step: Create Datacenters
// Datacenters are the resource providers in CloudSim. We need
at least one of them to run a CloudSim simulation
Datacenter datacenter0 = createDatacenter("Datacenter_0");

// Third step: Create Broker
DatacenterBroker broker = createBroker();
int brokerId = broker.getId();

// Fourth step: Create one virtual machine
vmlist = new ArrayList<>();

// VM description
int vmid = 0;
int mips = 1000;
long size = 10000; // image size (MB)
int ram = 512; // vm memory (MB)
long bw = 1000;
int pesNumber = 1; // number of cpus
String vmm = "Xen"; // VMM name

// create VM
Vm vm = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size,
vmm, new CloudletSchedulerTimeShared());

// add the VM to the vmList
vmlist.add(vm);

// submit vm list to the broker
broker.submitVmList(vmlist);

// Fifth step: Create one Cloudlet
cloudletList = new ArrayList<>();

Random rnd = new Random();
// Cloudlet properties
int id = 0;
long length1 = rnd.nextInt(400000);
long length2 = rnd.nextInt(400000);
long fileSize = 300;
long outputSize = 300;
UtilizationModel utilizationModel = new UtilizationModelFull();
```

```
            Cloudlet cloudlet1 = new Cloudlet(id, length1, pesNumber, fileS
            ize, outputSize, utilizationModel, utilizationModel, utilizatio
            nModel);
             cloudlet1.setUserId(brokerId);
             cloudlet1.setVmId(vmid);

             id++;
            Cloudlet cloudlet2 = new Cloudlet(id, length2, pesNumber, fileS
            ize, outputSize, utilizationModel, utilizationModel, utilizatio
            nModel);
             cloudlet2.setUserId(brokerId);
             cloudlet2.setVmId(vmid);

            //row 1 column 1 and 2
            Row1[0] = 0;
            Row1[1] = length1;

            //row 2 column 1 and 2
            Row2[0] = 1;
            Row2[1] = length2;

             // add the cloudlet to the list
             cloudletList.add(cloudlet1);
             cloudletList.add(cloudlet2);

             // submit cloudlet list to the broker
             broker.submitCloudletList(cloudletList);

             // Sixth step: Starts the simulation
             CloudSim.startSimulation();
             CloudSim.stopSimulation();

             //Final step: Print results when simulation is over
             List<Cloudlet> newList = broker.getCloudletReceivedList();
             printCloudletList(newList);

             Log.printLine("CloudSimExample1 finished!");
        } catch (NullPointerException e) {
             Log.printLine("Unwanted errors happen");
        }
        }

        /**
         * Creates the datacenter.
         *
         * @param name the name
         *
         * @return the datacenter
```

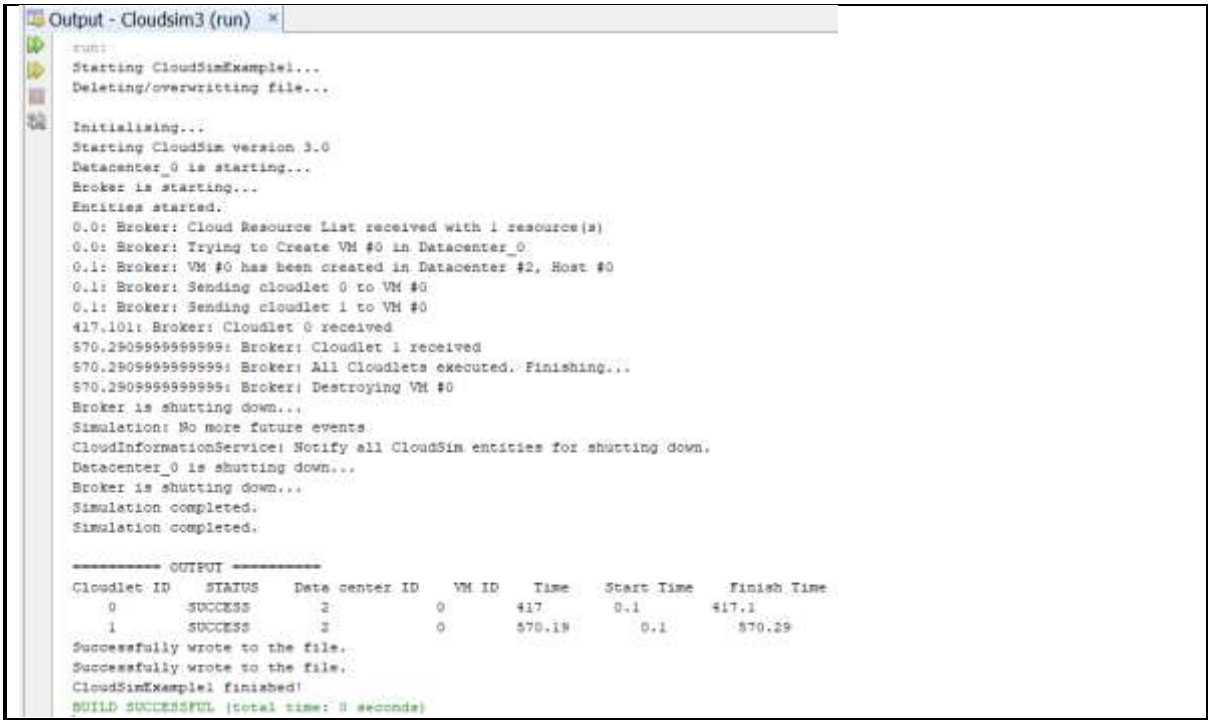
```
         */
        private static Datacenter createDatacenter(String name) {
        // Here are the steps needed to create a PowerDatacenter:
        // 1. We need to create a list to store
        // our machine
        List<Host> hostList = new ArrayList<>();

        // 2. A Machine contains one or more PEs or CPUs/Cores.
        // In this example, it will have only one core.
        List<Pe> peList = new ArrayList<>();

        int mips = 1000;

        // 3. Create PEs and add these into a list.
      peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to stor
      e Pe id and MIPS Rating

      // 4. Create Host with its id and list of PEs and add them to the lis
      t of machines
        int hostId = 0;
        int ram = 2048; // host memory (MB)
        long storage = 1000000; // host storage
        int bw = 10000;

        hostList.add(
                new Host(
                    hostId,
                    new RamProvisionerSimple(ram),
                    new BwProvisionerSimple(bw),
                    storage,
                    peList,
                    new VmSchedulerTimeShared(peList)
                )
        ); // This is our machine

        // 5. Create a DatacenterCharacteristics object that stores the
        // properties of a data center: architecture, OS, list of
        // Machines, allocation policy: time- or space-shared, time zone
        // and its price (G$/Pe time unit).
        String arch = "x86"; // system architecture
        String os = "Linux"; // operating system
        String vmm = "Xen";
        double time_zone = 10.0; // time zone this resource located
        double cost = 3.0; // the cost of using processing in this resource
        double costPerMem = 0.05;//the cost of using memory in this resource
        double costPerStorage = 0.001; // the cost of using storage in this
resource
        double costPerBw = 0.0; // the cost of using bw in this resource
```

```
        LinkedList<Storage> storageList = new LinkedList<>(); // we are not
adding SAN devices by now

      DatacenterCharacteristics characteristics = new DatacenterCharacteris
      tics(arch, os, vmm, hostList, time_zone, cost, costPerMem,
      costPerStorage, costPerBw);

        // 6. Finally, we need to create a PowerDatacenter object.
        Datacenter datacenter = null;
        try {
            datacenter = new Datacenter(name, characteristics, new VmAlloca
            tionPolicySimple(hostList), storageList, 0);
        } catch (Exception e) {}
        return datacenter;
        }

      // We strongly encourage users to develop their own broker policies,
      to submit vms and cloudlets according
      // to the specific rules of the simulated scenario
        /**
         * Creates the broker.
         *
         * @return the datacenter broker
         */
        private static DatacenterBroker createBroker() {
            DatacenterBroker broker = null;
            try {
                  broker = new DatacenterBroker("Broker");
            } catch (Exception e) {
                  return null;
            }
               return broker;
        }

        /**
         * Prints the Cloudlet objects.
         *
         * @param list list of Cloudlets
         */
        private static void printCloudletList(List<Cloudlet> list) {
        int size = list.size();
        Cloudlet cloudlet;

        String indent = "    ";
        Log.printLine();
        Log.printLine("========== OUTPUT ==========");
        Log.printLine("Cloudlet ID" + indent + "STATUS" + indent
```

```
                + "Data center ID" + indent + "VM ID" + indent + "Time" + ind
        ent + "Start Time" + indent + "Finish Time");

          DecimalFormat dft = new DecimalFormat("###.##");
          for (int i = 0; i < size; i++) {
               cloudlet = list.get(i);
               Log.print(indent + cloudlet.getCloudletId() + indent + indent);

               if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) {
                     Log.print("SUCCESS");

                     Log.printLine(indent + indent + cloudlet.getResourceId()
                                 + indent + indent + indent + cloudlet.getVmI
                                 d() + indent + indent
                                 + dft.format(cloudlet.getActualCPUTime()) +
                                 indent
                                 + indent + dft.format(cloudlet.getExecStartT
                                 ime()) + indent + indent
                                 + dft.format(cloudlet.getFinishTime()));

               //if it's cloudlet 1 add values to row 1 in file, else to row2
               if(cloudlet.getCloudletId() == 0){
                   Row1[2] = cloudlet.getVmId();
                   Row1[3] = cloudlet.getActualCPUTime();
                   Row1[4] = cloudlet.getExecStartTime();
                   Row1[5] = cloudlet.getFinishTime();
               } else {
                   Row2[2] = cloudlet.getVmId();
                   Row2[3] = cloudlet.getActualCPUTime();
                   Row2[4] = cloudlet.getExecStartTime();
                   Row2[5] = cloudlet.getFinishTime(); }
              }
          }
                   //use the write method to append the values to file
                   WriteResultsToFile(Row1);
                   WriteResultsToFile(Row2);
          }
          // Create method to write results to external file
          private static void WriteResultsToFile(double[] results)
          {
              //create a file writter
              FileWriter fw;
              try {
/** Constructs a FileWriter object given a File object.
If the second argument is true, then bytes will be written to the end of th
file rather than the beginning.
You may pass true as second parameter to the constructor of FileWriter to in
struct the writer to append the data instead of rewriting the file. */
```

```
        /** specific location
            fw = new FileWriter("C:\\Users\\hp\\Documents\\NetBeansProjects\
\Cloudsim3\\Output2.csv", true); */
            fw = new FileWriter("Output2.csv", true);
                try (PrintWriter pw = new PrintWriter(fw)) {
                //since we will write only 6 columns, the range for
                "FOR LOOP" is less than 7
                     for(int i = 0; i < 6; i++){
                        pw.print(results[i]);
              //to separate the values by comma, when opened in .txt format.
                        pw.print(",");
                        }
                        pw.println();
                        //Flush the output to the file
                        pw.flush();
                        //Close the Print Writer
                        pw.close();
                }

                //Close the File Writer
                fw.close();
                System.out.println("Successfully wrote to the file.");
        } catch (IOException e) {
                // TODO Auto-generated catch block
        }
        }
}
```

Output:

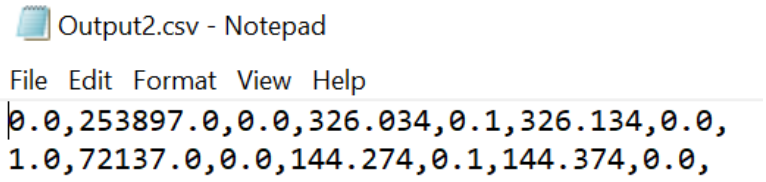


Results:

.CSV file

	A	B	C	D	E	F	G
1	0	253897	0	326.034	0.1	326.134	0
2	1	72137	0	144.274	0.1	144.374	0

.txt file



Our recent research

Our research is focused on seven key areas within the fields of Computing and Engineering fields. The following research domains aims to produce researchers from various disciplines to excel in an integrated IT solution in each area listed below:

- Cyber Security
- Internet of Things (IoT)
- Software Engineering
- Wireless Networks
- Computer Network Security
- Security & Privacy
- Unmanned Aerial Vehicle (UAV)

For more details, please refer to the manual's last section (our research), which provide details for the last three years selective research, for more details may access to the ResearchGate profile.

Acknowledgment

"This work was supported by Taylor's University through its Scholarship Master Program in School of Computer Science & Engineering (SCE).

References

[1] S. R. Pakize, S. masood Khademi, and A. Gandomi, “Comparison Of CloudSim, CloudAnalyst And CloudReports Simulator in Cloud Computing,” *Int. J. Comput. Sci. Netw. Solut.*, vol. 2, no. 5, pp. 19–27, 2014.

[2] K. Nazeer and N. Banu, “Cloud Computing Simulation Tools -A Study,” *Intern. J. Fuzzy Math. Arch. Int. J.*, vol. 7, no. 1, pp. 13–25, 2015.

[3] S. Vahora and R. Patel, “CloudSim-A Survey on VM Management Techniques,” *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 4, no. 1, pp. 128–133, 2015, doi: 10.17148/ijarcce.2015.4126.

[4] A. Singh, “Cloudsim - Cloudsim Tutorials,” 17-Dec-2019. [Online]. Available: https://www.cloudsimtutorials.online/cloudsim/. [Accessed: 16-Feb-2021].

[5] V. Thapar, “A Comparative Study of Cloud Simulation Tools,” *Int. J. Comput. Sci. Eng.*, vol. 9, no. 06, pp. 385–392, 2017.

[6] J. Byrne *et al.*, “A Review of Cloud Computing Simulation Platforms & Related Environments,” *Proc. 7th Int. Conf. Cloud Comput. Serv. Sci.*, no. Closer, pp. 651–663, 2017, doi: 10.5220/0006373006790691.

[7] S. P. Singh, A. Sharma, and R. Kumar, “Analysis of load balancing algorithms using cloud analyst,” *Int. J. Recent Technol. Eng.*, vol. 7, no. 6, pp. 684–687, 2019.

[8] K. Bahwaireth, L. Tawalbeh, E. Benkhelifa, Y. Jararweh, and M. A. Tawalbeh, “Experimental comparison of simulation tools for efficient cloud and mobile cloud computing applications,” *Eurasip J. Inf. Secur.*, no. 1, 2016, doi: 10.1186/s13635-016-0039-y.

Appendix (our research)

1. A. P. Singh et al., "A Novel Patient-Centric Architectural Framework for Blockchain-Enabled Healthcare Applications," in IEEE Transactions on Industrial Informatics, doi:10.1109/TII.2020.3037889
2. S. M. Muzammal, R. K. Murugesan and N. Z. Jhanjhi, "A Comprehensive Review on Secure Routing in Internet of Things: Mitigation Methods and Trust-based Approaches," in IEEE Internet of Things Journal, doi:10.1109/JIOT.2020.3031162
3. D. A. Shafiq, N. Jhanjhi and A. Abdullah, "Proposing A Load Balancing Algorithm For The Optimization Of Cloud Computing Applications," 2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), Karachi, Pakistan, 2019, pp. 1-6, doi: 10.1109/MACS48846.2019.9024785.
4. Thamer Tabbakh, Noha Alotaibi, Nawaf Bin Darwish, Zahrah A. Almusaylim, Sundos Alabdulkarim and N.Z. Jhanjhi, Optoelectronics and Optical Bio-Sensors, in Optoelectronics, OpenTech publisher, **2021**, DOI: 10.5772/intechopen.96183.
5. Himanshi Babbar, Shalli Rani, Mehedi Masud, Sahil Verma, Divya Anand, NZ Jhanjhi, Load Balancing Algorithm for Migrating Switches in Software-defined Vehicular Networks, in Computers, Materials & Continua, Tech Science Press, doi:10.32604/cmc.2021.014627, Vol.67, No.1, 2021, pp.1301-1316, (2021)
6. Ata Ullah, Muhammad Azeem, Humaira Ashraf, A. Alaboudi, Mamoona Humayun, NZ Jhanjhi, Secure Healthcare Data Aggregation and Transmission in IoT - A Survey, in IEEE Access, (2021) , DOI: 10.1109/ACCESS.2021.3052850.
7. Sandeep Verma, Satnam Kaur, Danda B. Rawat, Chen Xi , Linss T Alex , and NZ Jhanjhi, NZ Jhanjhi, Intelligent Framework using Internet of Things (IoT)-based Wireless Sensor Networks for Wildfire Detection, in IEEE Access,(**2021**).
8. Y. Hafeez, S. Ali, N. Jhanjhi, M. Humayun, A. Nayyar et al., Towards Distributed Components Selection and Acceptance Testing for Faults Detection using Fuzzy Approach, in Computers, Materials & Continua, Tech Science Press (2021), Vol.67, No.2, 2021, pp.1979-1996, doi:10.32604/cmc.2021.014830.
9. Sohan Kumar Pande, Sanjaya Kumar Panda, Satyabrata Das, Kshira Sagar Sahoo, Ashish Kr Luhach, NZ Jhanjhi, Roobaea Alroobaea and Sivakumar Sivanesan, A Resource Management Algorithm for Virtual Machine Migration in Vehicular Cloud Computing, in Computers, Materials & Continua, Tech Science Press (2021), Vol.67, No.2, 2021, pp.2647-2663, doi:10.32604/cmc.2021.015026.
10. R. Sujatha, Jyotir Moy Chatterjee, NZ Jhanjhi, Sarfraz Nawaz Brohi, Performance of deep learning vs machine learning in plant leaf disease detection, Microprocessors and Microsystems, Volume 80, 2021, 103615, ISSN 0141-9331, https://doi.org/10.1016/j.micpro.2020.103615.
11. S.Sankar, S Ramasubbareddy, A Kr. Luhach, G G Deverajan, W S Alnumay, NZ Jhanjhi, U Ghosh, P Sharma, Energy Efficient Optimal Parent Selection in RPL Using Firefly Optimization Algorithm for Internet of Things, in Transactions on Emerging Telecommunication Technologies (2020), https://doi.org/10.1002/ett.4171.
12. Zaman, N., Rafique, K., & Ponnusamy, V. (2021). ICT Solutions for Improving Smart Communities in Asia. IGI Global. http://doi:10.4018/978-1-7998-7114-9
13. Zaman, N., Rafique, K., & Ponnusamy, V. (2021). ICT Solutions for Improving Smart Communities in Asia. IGI Global. http://doi:10.4018/978-1-7998-7114-9
14. Ponnusamy, V., Rafique, K., & Zaman, N. (2020). Employing Recent Technologies for Improved Digital Governance. IGI Global. http://doi:10.4018/978-1-7998-1851-9
15. S.H. Kok, Azween Abdullah, NZ Jhanjhi, Evaluation Metric for Crypto-Ransomware Detection Using Machine Learning, Journal of Information Security and Applications, (2020), vol, 55, 102646, https://doi.org/10.1016/j.jisa.2020.102646.
16. NZ Jhanjhi, Mamoona Humayun and Saleh N Almuayqil, Industry 4.0: Cyber Security and Privacy Issues in Industrial Internet of Things, Computer Systems Science and Engineering, 2021, In Press.

17. Rana Muhammad Waseem, Farrukh Zeeshan Khan, Muneer Ahmad, Anum Naseem, NZ Jhanjhi, Uttam Ghosh, Performance Evaluation of AOMDV on Realistic and Efficient VANet Simulations, in Wireless Personal Communication WPC, Springer (2021), In Press.
18. G. Madhu, A. Govardhan, B. Sunil Srinivas, Kshira Sagar Sahoo, NZ Jhanjhi, K.S. Vardhan and B. Rohit, Imperative Dynamic Routing Between Capsules Network for Malaria Classification, in Computers, Materials & Continua, Tech Science Press (2021), In Press.
19. Maryam Shafiq, Humaira Ashraf, Ata Ullah, Mehedi Masud, Muhammad Azeem, NZ Jhanjhi and Mamoona Humayun, Robust Cluster-Based Routing Protocol for IoT-Assisted Smart Devices in WSN, in Computers, Materials & Continua, Tech Science Press (2021), In Press
20. Amir Latif, R.M., Hussain, K., Jhanjhi, N.Z. *et al.* A remix IDE: smart contract-based framework for the healthcare sector by using Blockchain technology. *Multimed Tools Appl* (2020). https://doi.org/10.1007/s11042-020-10087-1
21. Seungjin, L., Abdullah, A., & Jhanjhi, N. Z. (2020). A Review on Honeypot-based Botnet Detection Models for Smart Factory. *International Journal of Advanced Computer Science and Applications*, *11*(10.14569).
22. A. Almusaylim, Z., Jhanjhi, N. Comprehensive Review: Privacy Protection of User in Location-Aware Services of Mobile Cloud Computing. *Wireless Pers Commun* **111,** 541–564 (2020). https://doi.org/10.1007/s11277-019-06872-3
23. Zahrah A. Almusaylim, Abdulaziz Alhumam, N.Z. Jhanjhi, Proposing a Secure RPL based Internet of Things Routing Protocol: A Review, Ad Hoc Networks, Volume 101, 2020, 102096, ISSN 1570-8705, https://doi.org/10.1016/j.adhoc.2020.102096.
24. Navid Ali Khan, N.Z. Jhanjhi, Sarfraz Nawaz Brohi, Raja Sher Afgun Usmani, Anand Nayyar, Smart traffic monitoring system using Unmanned Aerial Vehicles (UAVs), Computer Communications, Volume 157, 2020, Pages 434-443, ISSN 0140-3664, https://doi.org/10.1016/j.comcom.2020.04.049.
25. M. Lim, A. Abdullah, N. Z. Jhanjhi, M. Khurram Khan and M. Supramaniam, "Link Prediction in Time-Evolving Criminal Network With Deep Reinforcement Learning Technique," in IEEE Access, vol. 7, pp. 184797-184807, 2019, doi: 10.1109/ACCESS.2019.2958873.
26. C. Diwaker et al., "A New Model for Predicting Component-Based Software Reliability Using Soft Computing," in IEEE Access, vol. 7, pp. 147191-147203, 2019, doi: 10.1109/ACCESS.2019.2946862.
27. Kamat, P., Gautam, A. S., Tavares, J., Mishra, B., Kumar, R., Zaman, N., & Khari, M. (2018). Recent trends in the era of cybercrime and the measures to control them. *Handbook of e-business security*, 243-258.
28. Hussain, S.J., Irfan, M., Jhanjhi, N.Z. *et al.* Performance Enhancement in Wireless Body Area Networks with Secure Communication. *Wireless Pers Commun* **116,** 1–22 (**2021**). https://doi.org/10.1007/s11277-020-07702-7
29. S. J. Hussain, U. Ahmed, H. Liaquat, S. Mir, N. Jhanjhi and M. Humayun, "IMIAD: Intelligent Malware Identification for Android Platform," 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 2019, pp. 1-6, doi: 10.1109/ICCISci.2019.8716471.
30. Khan N.A., Brohi S.N., Jhanjhi N. (2020) UAV's Applications, Architecture, Security Issues and Attack Scenarios: A Survey. In: Peng SL., Son L., Suseendran G., Balaganesh D. (eds) Intelligent Computing and Innovation on Data Science. Lecture Notes in Networks and Systems, vol 118. Springer, Singapore. https://doi.org/10.1007/978-981-15-3284-9_86
31. M. Saleh, N. Jhanjhi, A. Abdullah and Fatima-tuz-Zahra, "Proposing a Privacy Protection Model in Case of Civilian Drone," 2020 22nd International Conference on Advanced Communication Technology (ICACT), Phoenix Park, PyeongChang, Korea (South), 2020, pp. 596-602, doi: 10.23919/ICACT48636.2020.9061508.
32. Navid Ali Khan, Noor Zaman Jhanjhi, Sarfraz Nawaz Brohi, Anand Nayyar, Chapter Three - Emerging use of UAV's: secure communication protocol issues and challenges, Editor(s): Fadi Al-Turjman, Drones in Smart-Cities, Elsevier, 2020, Pages 37-55, ISBN 9780128199725, https://doi.org/10.1016/B978-0-12-819972-5.00003-3.
33. Fatima-tuz-Zahra, N. Jhanjhi, S. N. Brohi and N. A. Malik, "Proposing a Rank and Wormhole Attack Detection Framework using Machine Learning," 2019 13th International Conference on Mathematics,

Actuarial Science, Computer Science and Statistics (MACS), Karachi, Pakistan, 2019, pp. 1-9, doi: 10.1109/MACS48846.2019.9024821.

34. S. Saeed and A. B. Abdullah, "Investigation of a Brain Cancer with Interfacing of 3-Dimensional Image Processing," 2019 International Conference on Information Science and Communication Technology (ICISCT), Karachi, Pakistan, 2019, pp. 1-6, doi: 10.1109/CISCT.2019.8777404.
35. Khalid Hussain, NZ Jhanjhi, Hafiz Mati-ur-Rahman, Jawad Hussain, Muhammad Hasan Islam, Using a systematic framework to critically analyze proposed smart card based two factor authentication schemes, Journal of King Saud University - Computer and Information Sciences, 2019, ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2019.01.015.
36. Ren, A., Liang, C., Hyug, I., Broh, S., & Jhanjhi, N. Z. (**2020**). A Three-Level Ransomware Detection and Prevention Mechanism. *EAI Endorsed Transactions on Energy Web*, *7*(26).
37. Mamoona Humayun, NZ Jhanjhi, Ahmed Alsayat, Vasaki Ponnusamy, Internet of things and ransomware: Evolution, mitigation and prevention, Egyptian Informatics Journal, 2020, ISSN 1110-8665, https://doi.org/10.1016/j.eij.2020.05.003.
38. Humayun, M., & Jhanjhi, N. Z. (2019). Exploring The Relationship Between GSD, Knowledge Management, Trust And Collaboration. *Journal of Engineering Science and Technology*, *14*(2), 820-843.
39. V. Singhal et al., "Artificial Intelligence Enabled Road Vehicle-Train Collision Risk Assessment Framework for Unmanned Railway Level Crossings," in IEEE Access, vol. 8, pp. 113790-113806, 2020, doi: 10.1109/ACCESS.2020.3002416.
40. Humayun, M., Jhanjhi, N. Z., & Alamri, M. Z. Smart Secure and Energy Efficient Scheme for E-Health Applications using IoT: A Review. *International Journal of Computer Science and Network Security*, *20*(4), 55-74.
41. B. Hamid, N. Jhanjhi, M. Humayun, A. Khan and A. Alsayat, "Cyber Security Issues and Challenges for Smart Cities: A survey," 2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), Karachi, Pakistan, 2019, pp. 1-7, doi: 10.1109/MACS48846.2019.9024768.
42. S. Jacob et al., "An Adaptive and Flexible Brain Energized Full Body Exoskeleton With IoT Edge for Assisting the Paralyzed Patients," in IEEE Access, vol. 8, pp. 100721-100731, 2020, doi: 10.1109/ACCESS.2020.2997727.
43. A. Almusaylim, Z.; Jhanjhi, N.; Alhumam, A. Detection and Mitigation of RPL Rank and Version Number Attacks in the Internet of Things: SRPL-RP. Sensors MDPI, 2020, https://doi.org/10.3390/s20215997,20, 5997
44. I. A. Chesti, M. Humayun, N. U. Sama and N. Jhanjhi, "Evolution, Mitigation, and Prevention of Ransomware," 2020 2nd International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 2020, pp. 1-6, doi: 10.1109/ICCIS49240.2020.9257708.
45. S. Saeed, N. Jhanjhi, M. Naqvi, M. Humayun and S. Ahmed, "Ransomware: A Framework for Security Challenges in Internet of Things," 2020 2nd International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 2020, pp. 1-6, doi: 10.1109/ICCIS49240.2020.9257660.
46. Fatima-tuz-Zahra, N. Jhanjhi, S. N. Brohi, N. A. Malik and M. Humayun, "Proposing a Hybrid RPL Protocol for Rank and Wormhole Attack Mitigation using Machine Learning," 2020 2nd International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 2020, pp. 1-6, doi: 10.1109/ICCIS49240.2020.9257607.
47. D. K. Alferidah and N. Jhanjhi, "Cybersecurity Impact over Bigdata and IoT Growth," 2020 International Conference on Computational Intelligence (ICCI), Bandar Seri Iskandar, Malaysia, 2020, pp. 103-108, doi: 10.1109/ICCI51257.2020.9247722.
48. M. Humayun, N. Jhanjhi, M. Alruwaili, S. S. Amalathas, V. Balasubramanian and B. Selvaraj, "Privacy Protection and Energy Optimization for 5G-Aided Industrial Internet of Things," in IEEE Access, vol. 8, pp. 183665-183677, 2020, doi: 10.1109/ACCESS.2020.3028764.
49. S. K. Mishra et al., "Energy-Aware Task Allocation for Multi-Cloud Networks," in IEEE Access, vol. 8, pp. 178825-178834, 2020, doi: 10.1109/ACCESS.2020.3026875.

50. S. Ali et al., "Towards Pattern-Based Change Verification Framework for Cloud-Enabled Healthcare Component-Based," in IEEE Access, vol. 8, pp. 148007-148020, 2020, doi: 10.1109/ACCESS.2020.3014671.
51. Kok, S. H., Abdullah, A., Jhanjhi, N. Z., & Supramaniam, M. (2019). A Review of Intrusion Detection System using Machine Learning Approach. *International Journal of Engineering Research and Technology*, *12*(1), 8-15.
52. Ubing, A. A., Jasmi, S. K. B., Abdullah, A., Jhanjhi, N. Z., & Supramaniam, M. (2019). Phishing Website detection: An improved accuracy through feature selection and ensemble learning. *International Journal Of Advanced Computer Science And Applications*, *10*(1), 252-257.
53. K. Hussain, S. J. Hussain, N. Jhanjhi and M. Humayun, "SYN Flood Attack Detection based on Bayes Estimator (SFADBE) For MANET," 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 2019, pp. 1-4, doi: 10.1109/ICCISci.2019.8716416.
54. Alamri, M., Jhanjhi, N. Z., & Humayun, M. (2019). Blockchain for Internet of Things (IoT) Research Issues Challenges & Future Directions: A Review. *Int. J. Comput. Sci. Netw. Secur*, *19*, 244-258.
55. Kok, S. H., Abdullah, A., Jhanjhi, N. Z., & Supramaniam, M. (2019). Prevention of crypto-ransomware using a pre-encryption detection algorithm. *Computers*, *8*(4), 79
56. Lim M, Abdullah A, Jhanjhi N, Supramaniam M. Hidden Link Prediction in Criminal Networks Using the Deep Reinforcement Learning Technique. COMPUTERS. 2019; 8(1):8.
57. M. Almulhim and N. Zaman, ''Proposing secure and lightweight authentication scheme for IoT based E-health applications,'' in Proc. 20th Int. Conf. Adv. Commun. Technol., Chuncheon-si Gangwon-do, South Korea, Feb. 2018, pp. 481–487.
58. Saeed, S., Jhanjhi, N. Z., Naqvi, M., & Humayun, M. (2019). Analysis of Software Development Methodologies. *International Journal of Computing and Digital Systems*, *8*(5), 446-460.
59. A. Alsubaie, M. Alaithan, M. Boubaid and N. Zaman, "Making learning fun: Educational concepts & logics through game," 2018 20th International Conference on Advanced Communication Technology (ICACT), Chuncheon-si Gangwon-do, Korea (South), 2018, pp. 454-459, doi: 10.23919/ICACT.2018.8323792.
60. Ahmad, M., Zaman, N., & Al-Amin, M. (2017). An experimental research in health informatics for enhancing ovarian cancer identification in ovarian imaging analysis using fuzzy histogram equalization. *Journal of Medical Imaging and Health Informatics*, *7*(6), 1385-1390.
61. F. A. Almusalli, N. Zaman and R. Rasool, "Energy efficient middleware: Design and development for mobile applications," 2017 19th International Conference on Advanced Communication Technology (ICACT), Bongpyeong, 2017, pp. 541-549, doi: 10.23919/ICACT.2017.7890149.
62. Khan, A., Jhanjhi, N. Z., Humayun, M., & Ahmad, M. (2020). The Role of IoT in Digital Governance. In *Employing Recent Technologies for Improved Digital Governance* (pp. 128-150). IGI Global.
63. Elijah, A. V., Abdullah, A., Jhanjhi, N., Supramaniam, M., & Abdullateef, B. (2019). 'Ensemble and deep-learning methods for two-class and multi-attack anomaly intrusion detection: An empirical study. *Int. J. Adv. Comput. Sci. Appl.*, *10*, 520-528.
64. S. S. A. Bukhari, M. Humayun, S. A. A. Shah and N. Z. Jhanjhi, "Improving Requirement Engineering Process for Web Application Development," 2018 12th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), Karachi, Pakistan, 2018, pp. 1-5, doi: 10.1109/MACS.2018.8628422.