Configuration Manual

MSc Research Project
Cloud Computing

Shiva Ram Raja Gandhi Raja
Student ID: x21219095

School of Computing
National College of Ireland

Supervisor: Mr. Rashid Mijumbi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shiva Ram Raja Gandhi Raja
Student ID:	x21219095
Programme:	Cloud Computing
Year:	2022
Module:	MSc Research Project
Supervisor:	Mr. Rashid Mijumbi
Submission Due Date:	14/8/2023
Project Title:	Energy Efficient Task Scheduling Approach in Cloud Environments towards Green Cloud
Word Count:	902
Page Count:	7

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Shiva Ram Raja Gandhi Raja
Date:	14th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Shiva Ram Raja Gandhi Raja
x21219095

1 Introduction

With this report, one can understand the steps needed to run and to successfully deploy our research which is Energy Efficient Task Scheduling Approach in Cloud Environment using the CloudSim framework. This report will walk you through the hardware requirements and software requirements to achieve successful implementation of the project.

2 Pre-Requirements

Let's look into the prerequisites of this project, by satisfying these we will be well prepared on the configuration journey for our Energy-Efficient Task Scheduling project using CloudSim.

2.1 Hardware Requirements

For this project, we need a laptop or desktop with a minimum of i3 processor, 8 GB RAM, and 100 GB storage capacity is needed to run the CloudSim toolkit.

Table 1 shows the hardware setup I used for the project.

Table 1: Hardware used for the project

Parameter	**Value**
Processor	Intel i5
CPU	6 cores and 12 threads
RAM (GB)	8
Storage (TB)	1

2.2 Software Requirements

In this section, we will know the software we need to download and install before we start the project.

- Java JDK 1.8.0-311: Java JDK is necessary for the CloudSim project as it provides the essential tools and libraries for compiling, running, and developing Java-based simulation applications that model and analyze cloud computing environments and scenarios. You can install the version from `https://www.oracle.com/ie/java/technologies/javase/javase8u211-later-archive-downloads.html`

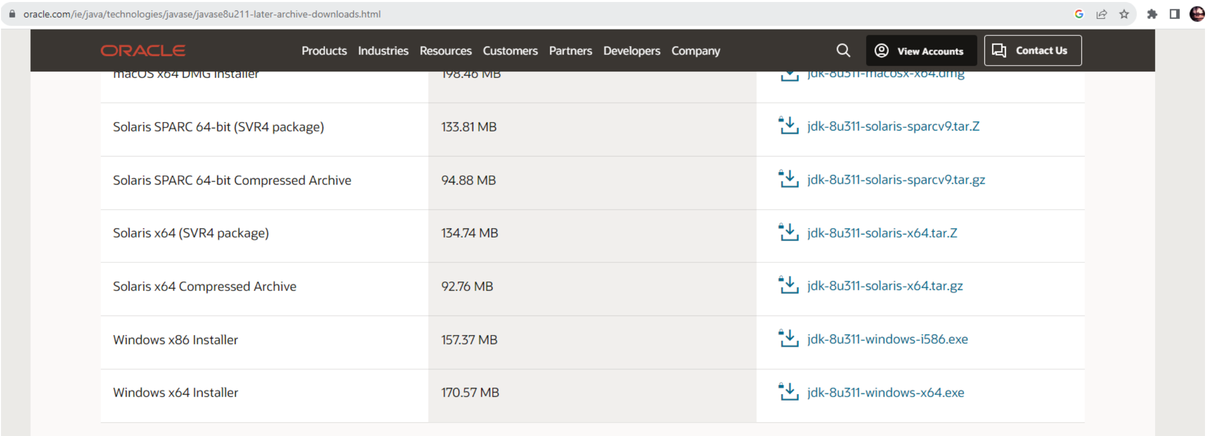


Figure 1: Download JDK file

- Eclipse 2022-09 (4.25.0): Eclipse IDE is valuable for the CloudSim project as it offers a robust and user-friendly integrated development environment, facilitating efficient coding, debugging, and project management throughout the simulation modeling and implementation process. You can install the version from `https://www.eclipse.org/downloads/packages/release/2022-09/r`

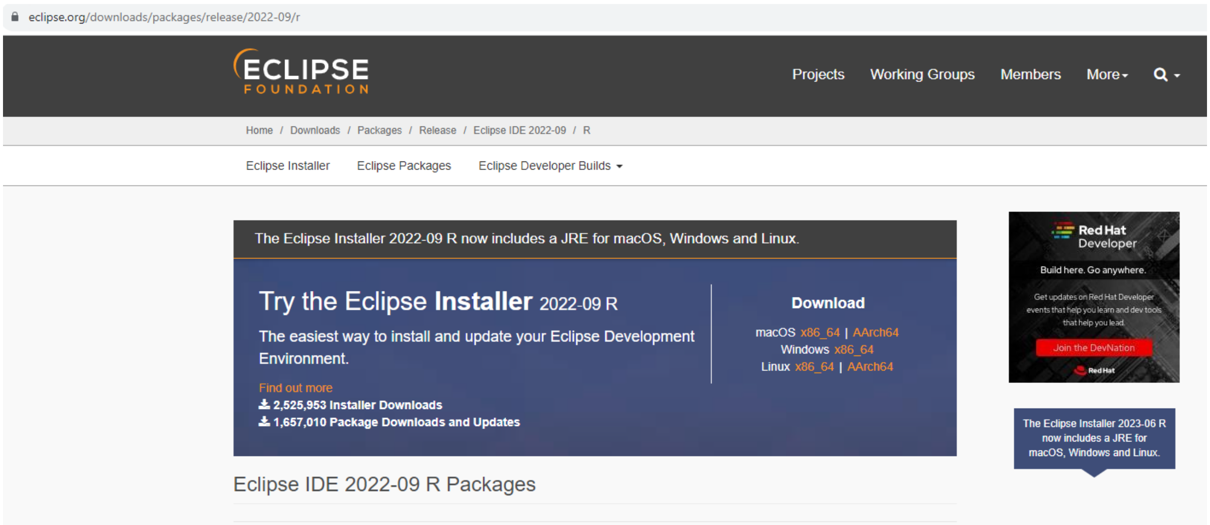


Figure 2: Download Eclipse IDE

- Cloudsim Toolkit: It is utilized to simplify the modeling and simulation of cloud computing systems, enabling researchers and developers to experiment, analyze, and optimize various cloud-based algorithms, policies, and architectures in a controlled and scalable environment. You can refer the code from `https://github.com/Cloudslab/cloudsim/releases/tag/cloudsim-3.0.3` and `https://github.com/Aniket144/Cloud_Simulation_Project`

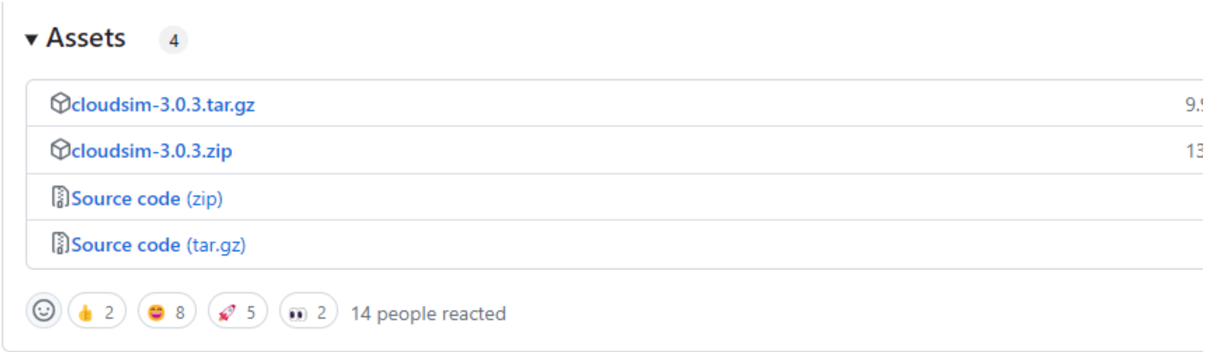


Figure 3: Download Cloudsim toolkit

3 Configuration setup

Once all the above-mentioned software has been installed, we need to create an environment variable for Java. To do that we need to follow the below steps.
1. Go to "systems setting".
2. Select "Advance system settings".
3. Go to "Environment variables".
4. Under System variables click "path".
5. Edit and add a new path of Java.

3.1 Importing CloudSim

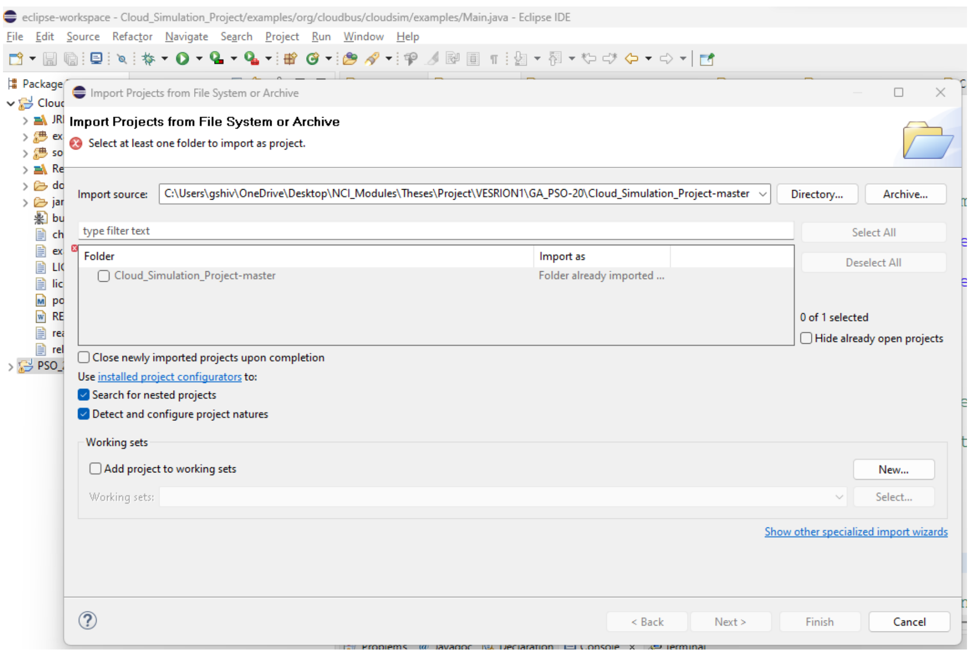


Figure 4: Importing Project

We will now import the cloudsim package in Eclipse and create a project. Figure 4 shows importing cloudsim packages.

To create a Java project in Eclipse IDE follow these below steps,
1. Click file on the left corner of the IDE.
2. Click open project from the file system.
3. Browse the "Import source" and select CloudSim package which you need to import.
4. Click finish to complete the import process.

4 Implementation, Configuring the code part

Let's look into how we will initialize CloudSim by integrating the proposed algorithm and we will also look into the parameters where we have made the configuration changes.

From the main.jav code set we will now configure our required scenario, Figure 5 shows the parameters for configuring VM. and Figure 6 shows parameters for cloudlets. We can change the VM and cloudlet configuration and set it as per our needs which gives a better simulation outcome.

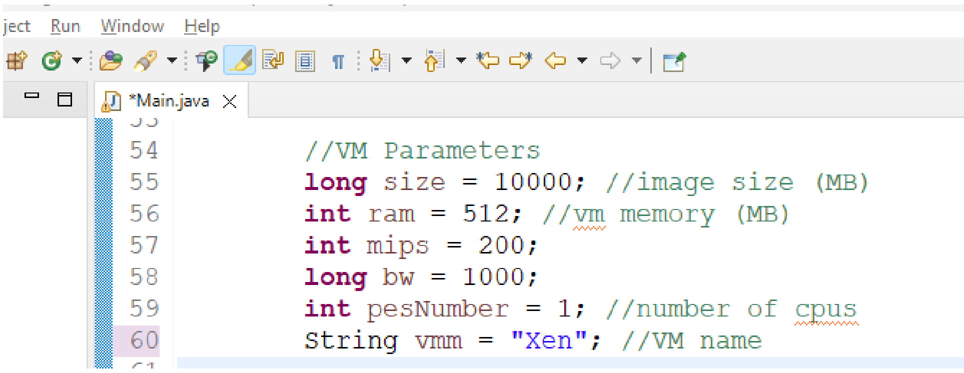


Figure 5: VM Configuration

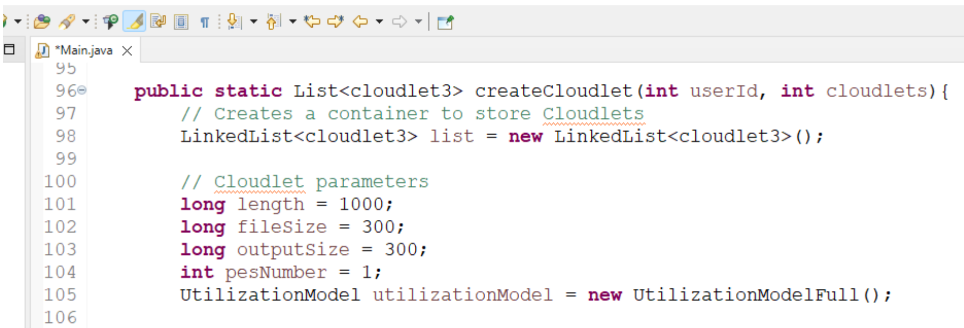


Figure 6: Cloudlet parameters

Once we set the VM and cloudlet configuration, we will now initialize the CloudSim library,

Figure 7 shows setting up and running a simulation using the CloudSim simulation framework. The code simulates a cloud computing environment where virtual machines execute cloudlets (tasks) on data centers.

```java
try {
    // First step: Initialize the CloudSim package. It should be called
    // before creating any entities.
    int num_user = 1;   // number of grid 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 list one o
    @SuppressWarnings("unused")
    Datacenter datacenter0 = createDatacenter("Datacenter_0");
    @SuppressWarnings("unused")
    Datacenter datacenter1 = createDatacenter("Datacenter_1");

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

    //Fourth step: Create VMs and Cloudlets and send them to broker
    vmlist = createVM(brokerId,10); //creating 10 vms
    cloudletList = createCloudlet(brokerId,20); // creating 20 cloudlets

```

Figure 7: Initializing the CloudSim package

Once cloudsim has been initialized and created all the required environments, now we will initialize our proposed algorithm GA-PSO. First, a Genetic Algorithm is applied to perform task scheduling, it will Initialize and evaluate the population and find the fittest population. And then cross-over and mutation method is applied to the fittest population and evaluated.

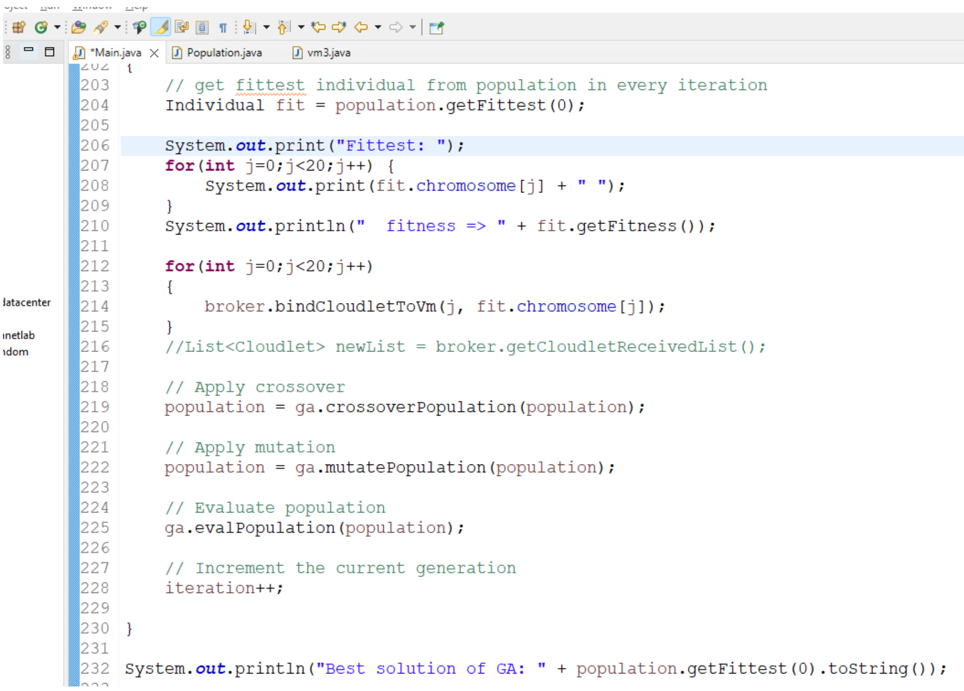
Figure 8 and Figure 9 shows the code where the population is being initialized and evaluated for the fittest population.

```java

// Initialize population
System.out.println("Population Initialization");
int chromosomeLength = 20;
Population population = ga.initPopulation(chromosomeLength);

// Evaluate population
ga.evalPopulation(population);

```

Figure 8: Initializing Population

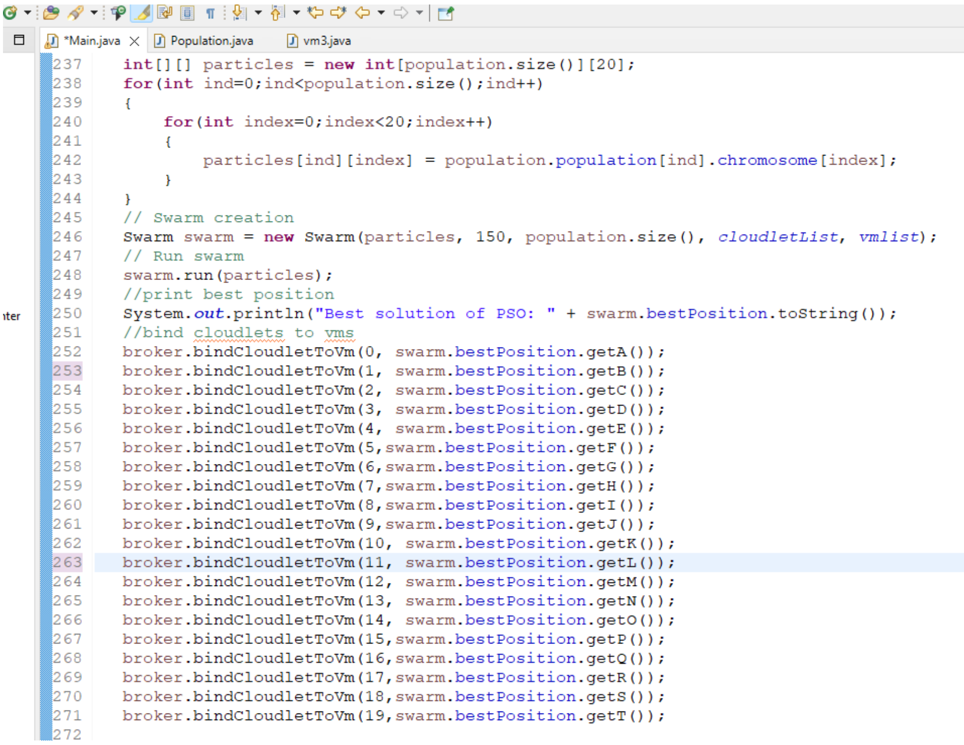
After finding the best solution of GA we will apply PSO over GA generated population. Using PSO on a population created by GA attempts to combine the advantages of both methods for more effective and efficient optimization. This hybrid technique is intended to solve the constraints of individual algorithms and offer a strong resolution for challenging optimization issues, such as the scheduling of tasks in a cloud environment that are energy-efficient.

Figure 10 shows Swarm creation and finding the best position for particles, then binding the cloudlets (tasks) to VMs depending on the best position evaluated by PSO.



```java
{
    // get fittest individual from population in every iteration
    Individual fit = population.getFittest(0);

    System.out.print("Fittest: ");
    for(int j=0;j<20;j++) {
        System.out.print(fit.chromosome[j] + " ");
    }
    System.out.println("  fitness => " + fit.getFitness());

    for(int j=0;j<20;j++)
    {
        broker.bindCloudletToVm(j, fit.chromosome[j]);
    }
    //List<Cloudlet> newList = broker.getCloudletReceivedList();

    // Apply crossover
    population = ga.crossoverPopulation(population);

    // Apply mutation
    population = ga.mutatePopulation(population);

    // Evaluate population
    ga.evalPopulation(population);

    // Increment the current generation
    iteration++;

}

System.out.println("Best solution of GA: " + population.getFittest(0).toString());
```

Figure 9: Get the fittest genetic algorithm population sequence



```java
int[][] particles = new int[population.size()][20];
for(int ind=0;ind<population.size();ind++)
{
    for(int index=0;index<20;index++)
    {
        particles[ind][index] = population.population[ind].chromosome[index];
    }
}
// Swarm creation
Swarm swarm = new Swarm(particles, 150, population.size(), cloudletList, vmlist);
// Run swarm
swarm.run(particles);
//print best position
System.out.println("Best solution of PSO: " + swarm.bestPosition.toString());
//bind cloudlets to vms
broker.bindCloudletToVm(0, swarm.bestPosition.getA());
broker.bindCloudletToVm(1, swarm.bestPosition.getB());
broker.bindCloudletToVm(2, swarm.bestPosition.getC());
broker.bindCloudletToVm(3, swarm.bestPosition.getD());
broker.bindCloudletToVm(4, swarm.bestPosition.getE());
broker.bindCloudletToVm(5,swarm.bestPosition.getF());
broker.bindCloudletToVm(6,swarm.bestPosition.getG());
broker.bindCloudletToVm(7,swarm.bestPosition.getH());
broker.bindCloudletToVm(8,swarm.bestPosition.getI());
broker.bindCloudletToVm(9,swarm.bestPosition.getJ());
broker.bindCloudletToVm(10, swarm.bestPosition.getK());
broker.bindCloudletToVm(11, swarm.bestPosition.getL());
broker.bindCloudletToVm(12, swarm.bestPosition.getM());
broker.bindCloudletToVm(13, swarm.bestPosition.getN());
broker.bindCloudletToVm(14, swarm.bestPosition.getO());
broker.bindCloudletToVm(15,swarm.bestPosition.getP());
broker.bindCloudletToVm(16,swarm.bestPosition.getQ());
broker.bindCloudletToVm(17,swarm.bestPosition.getR());
broker.bindCloudletToVm(18,swarm.bestPosition.getS());
broker.bindCloudletToVm(19,swarm.bestPosition.getT());
```

Figure 10: Steps for swarm creation, finding best position for particles and binding cloudlets to VMs

Figure 11 shows the execution output and expected result as our proposed hybrid techniques evaluation is comparatively lesser than the traditional algorithm and its output is shown in Figure 12. The output makespan, cost, and simulation time may change every time depending on systems resource availability, load, background tasks, hardware variability, and much more. If we notice in our theses report where we have evaluated and compared with graphs the output is different because we have referred the output of different execution which we ran previously. It is recommended to run simulations on dedicated machines or cloud instances with controlled and consistent resources.

```
<terminated> Main [Java Application] C:\Users\gshiv\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038\jre\bin\javaw.exe

========== OUTPUT ==========
Cloudlet ID    STATUS          VM ID       Time     Start Time    Finish Time
    0         SUCCESS          2          6.63        0.2           6.83
    9         SUCCESS          0          7.57        0.2           7.77
    3         SUCCESS          9          10.84       0.2           11.04
    1         SUCCESS          6          11.72       0.2           11.92
    10         SUCCESS          5          14.65       0.2           14.85
    7         SUCCESS          4          14.93       0.2           15.13
    2         SUCCESS          2          9.79        6.83          16.62
    12         SUCCESS          5          7.17        14.85         22.02
    17         SUCCESS          9          14.41       11.04         25.45
    4         SUCCESS          2          9.89        16.62         26.52
    13         SUCCESS          4          12.37       15.13         27.5
    18         SUCCESS          9          5.88        25.45         31.33
    5         SUCCESS          2          6.08        26.52         32.59
    19         SUCCESS          9          8.22        31.33         39.55
    16         SUCCESS          4          14.43       27.5          41.92
    6         SUCCESS          2          10.59       32.59         43.18
    8         SUCCESS          2          11.63       43.18         54.81
    11         SUCCESS          2          13.64       54.81         68.45
    14         SUCCESS          2          14.28       68.45         82.73
    15         SUCCESS          2          12.79       82.73         95.52
Make span : 217.505
Execution Cost: 16.5
Total simulation time: 241 ms
GA-PSO finished!
```

Figure 11: Execution output for proposed hybrid algorithm

```
Problems  Javadoc  Declaration  Console  Terminal
<terminated> PSO_Scheduler [Java Application] C:\Users\gshiv\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038
========== OUTPUT ==========
Cloudlet ID     STATUS          VM ID       Time     Start Time     Finish Time
    3          SUCCESS          3         52.15        0.1            52.25
    2          SUCCESS          2         56.55        0.1            56.65
    8          SUCCESS          6         56.6         0.1            56.7
    16          SUCCESS          8         57.4         0.1            57.5
    12          SUCCESS          7         60.75        0.1            60.85
    0          SUCCESS          0         61.6         0.1            61.7
    17          SUCCESS          9         66.55        0.1            66.64
    5          SUCCESS          5         66.6         0.1            66.7
    4          SUCCESS          4         67.25        0.1            67.35
    1          SUCCESS          1         70.85        0.1            70.95
    7          SUCCESS          2         57.4         56.65          114.04
    13          SUCCESS          6         57.45        56.7           114.15
    9          SUCCESS          5         52.3         66.7           119
    6          SUCCESS          4         52           67.35          119.35
    18          SUCCESS          7         60.75        60.85          121.6
    10          SUCCESS          0         66.7         61.7           128.4
    14          SUCCESS          1         66.65        70.95          137.6
    11          SUCCESS          2         56.55        114.04         170.59
    15          SUCCESS          5         52.2         119            171.2
    19          SUCCESS          7         57.55        121.6          179.15
Make span: 1195.8250000000003
Execution cost: 20.8
Total simulation time: 164 ms
PSO finished!
```

Figure 12: Execution output for traditional PSO algorithm