Delhi Technological University

Swarm and Evolutionary Computing (CO423)

Comparative Study of Different Task Scheduling Algorithms in Cloud Computing Environment (Particle Swarm Optimization, Ant Colony Optimization & other traditional algorithms)

SUBMITTED BY:

Mohit Kumar (2K18/IT/072)

Mrigank Badola (2K18/IT/073)

SUBMITTED TO:

Ms. Pratima Sharma

Dept. of Computer Engineering

November 2021

Acknowledgment

We would first like to thank the almighty God for giving us the power, strength and perseverance to complete this work.

We would also like to express our deep gratitude towards Ms. Amrita Sisodia and Ms. Pratima Sharma, Department of Computer Engineering, Delhi Technological University for their constant support and guidance. This work would not have been possible without their support.

Nonetheless, we would also like to take the opportunity to thank our family members and friends for constantly motivating us to work harder.

Thanks & Regards

Mohit Kumar (2K18/IT/072)

Mrigank Badola (2K18/IT/073)

Certificate

This is to certify that the technical report entitled “Comparative Study of Different Task Scheduling Algorithms in Cloud Computing Environment (Particle Swarm Optimization, Ant Colony Optimization & other traditional algorithms)” is a record of the bona fide work carried out by Mohit Kumar (2K18/IT/072) and Mrigank Badola (2K18/IT/073), Delhi Technological University during the academic year 2021-2022.

MS. PRATIMA SHARMA

PROFESSOR

DEPT. OF COMPUTER ENGINEERING

Abstract

Whenever we talk about cloud computing, we refer to the various services available to us via the Internet. One such aspect of service is task scheduling. It refers to the process of allocating tasks to the virtual machines present in the datacenters. Now, usually the number of tasks at hand exceeds the number of virtual machines and hence one has to apply some sort of algorithm in order to ensure that: (i) All the tasks are finished at the earliest and/or (ii) Load is balanced. The task scheduling algorithms in the cloud could be compared to the CPU scheduling algorithms in the OS.

In IT207, Modelling and Simulation and in IT304, Software Engineering, we had learnt that why it is better to have a simulation of the system before the actual build. Simulations normally involve no capital investment, are helpful in risk evaluation, are easy to use and scalable. Hence, we have used CloudSim framework to simulate task scheduling in a cloud computing environment. This framework is open source, free of cost and doesn't require a computer with high specifications.

Hence in this project, we have implemented both traditional (like FCFS) and evolutionary algorithms for CloudSim framework in order to test them for different tasks at hand. We would then compare these scheduling algorithms on the basis of their finishing time.

Table of Contents

Introduction

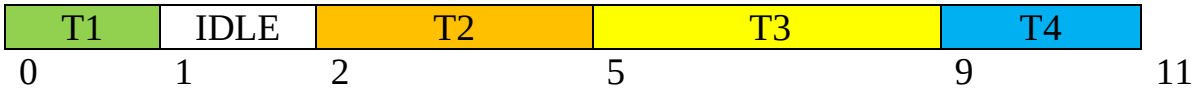
Traditional Scheduling Algorithms

One of the simplest algorithms for task scheduling is the First Come First Serve (FCFS) algorithm. It is based on real life application of a queue system. It simply schedules the tasks according to their arrival time. However, one can simply imagine why such an algorithm is not normally used in real life scenarios. It neither has the concept of *task prioritization* nor is a *pre-emptive* algorithm. This means, once a task schedule has begun, its execution is completed in one sitting. Hence, it is not an effective algorithm and several other algorithms have since developed to increase the performance of task schedulers. These include like Shortest Job First (SJF), Round Robin, Priority Based etc,

One of the ways to represent task scheduling is Generalized Activity Normalization Time Table (Gantt chart). It is a type of bar chart in which tasks completion is shown along with a timeline. For e.g., if FCFS algorithm is being followed and a list of tasks is given as:

TASKS	ARRIVAL TIME	BURST TIME
T1	0	1
T2	2	3
T3	2	4
T4	5	2

Then the Gantt Chart would be as follows:



What is an Evolutionary Algorithm?

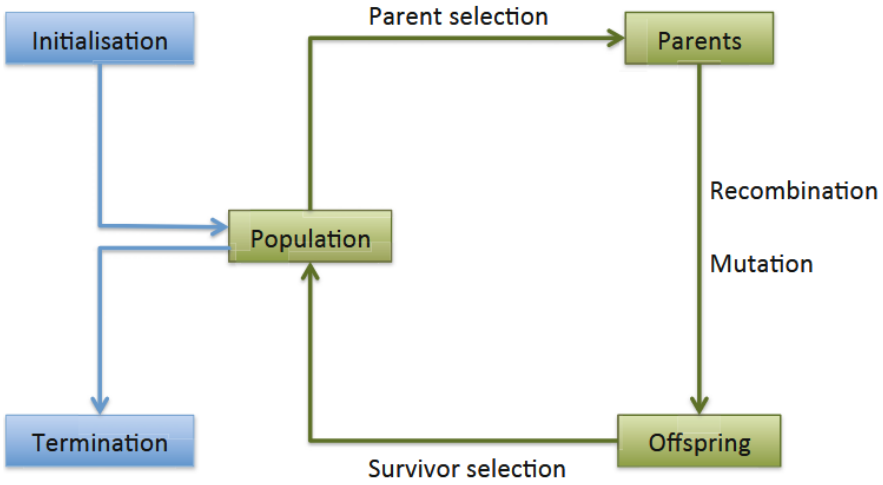
The concept of evolution has normally been taught and studied in the field of Biology. We say a species has been evolving if, over time and generations, they have shown some change(s) in their characteristics. Evolution relies upon the process of natural selection or "survival of the fittest".

With this real-life example at hand, evolutionary algorithms were introduced. EAs are generally used in optimization problems, and usually provide good approximate solutions. The steps followed in an EA are:

- Representation: Initializes population with random candidate solutions
- Evaluation: Evaluate each candidate with a fitness function
- Parent Selection: Select candidates for next step
- Apply variation operators: Mutation (applied to a single parent) and Recombination (applied to a pair of parents)
- Survivor selection: Offspring compete with current population and replace less-fit individuals with fitter offspring.

After initialization, steps 2-5 are repeated until the *termination condition* has been reached. Some of the conditions could be:

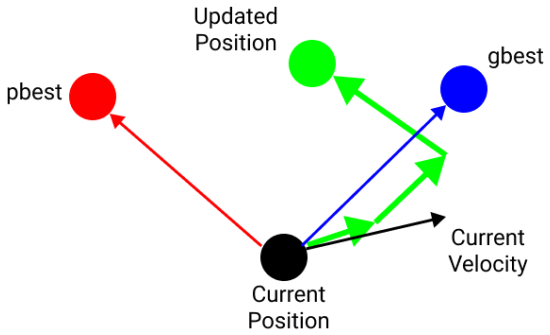
- Reaching a known optimal fitness level
- Reaching maximum limit of fitness evaluations
- Dropping of population diversity under a given threshold
- Reaching maximum time allowed by a system
- Fitness improvement remaining under a threshold value for a given period of time



Particle Swarm Optimization

Particle Swarm Optimization is an optimization technique which applies the concept of population-based search. Here, position of particle is the solution while swarm of particles act as searching agent. PSO find the minimum value for the function.

The PSO particle learns from other particles (social learning) as well as from its own experience (cognitive learning). Due to this, there are two solutions: *gbest* and *pbest*. *Velocity* V and *acceleration* play a role here as well since there are changes in position with respect to time (or iteration) plus there is an implementation of *random weighted acceleration*.



The *position* X at t+1 can be formulated as:

$$X(t+1) = X(t) + V(t+1)$$

And subsequently *velocity* V at t+1 can be formulated as:

$V(t+1) = wV(t) + c_1 r_1 \left(X_{pbest} - X(t)\right) + c_2 r_2 \left(X_{gbest} - X(t)\right)$ where,

w = Inertial weight; c_1, c_2 = Accelerating factor;

r_1, r_2 = Uniformly distributed random number $\in$ [0,1]

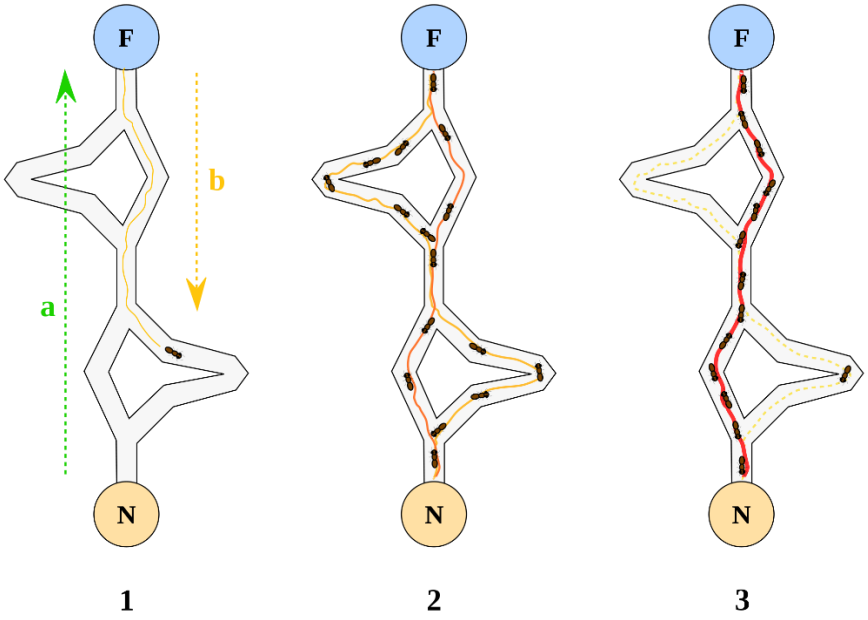
PSEUDOCODE

```
while termination condition not met {
  for every particle {
    fitness-function ()
    update-gbest ()
    update-pbest ()
    update-velocity ()
    update-position ()
  }
}
```

Ant Colony Optimization

Ant Colony Optimization is an optimization technique which applies the concept of population-based metaheuristic. It involves probabilistic techniques inspired from the behavior of real ant colonies. For this algorithm, problems must be considered in the form of path finding with a weighted graph. Hence, ACO finds the shortest path.

It starts with a group of ants leaving their nests and searching randomly for food. Whenever a particular ant finds food, they deposit *pheromones* on their way to the nest. The more the *pheromones* concentrated on the path, the more becomes the *probability* of it being followed. Since *pheromones* evaporate with time, longer paths are automatically discarded. Hence, different ants keep giving different path routes until we reach the *shortest path* where the *pheromone trails* have the strongest concentration. Almost all of the ants will then follow this path, reinforcing it further and hence all of them reach the food source.



Pseudocode

```
while termination condition not met {
  Construct-Ant-Solutions ()
  Apply-Local-Search () //optional
  Update-Pheromones ()
}
```

CloudSim

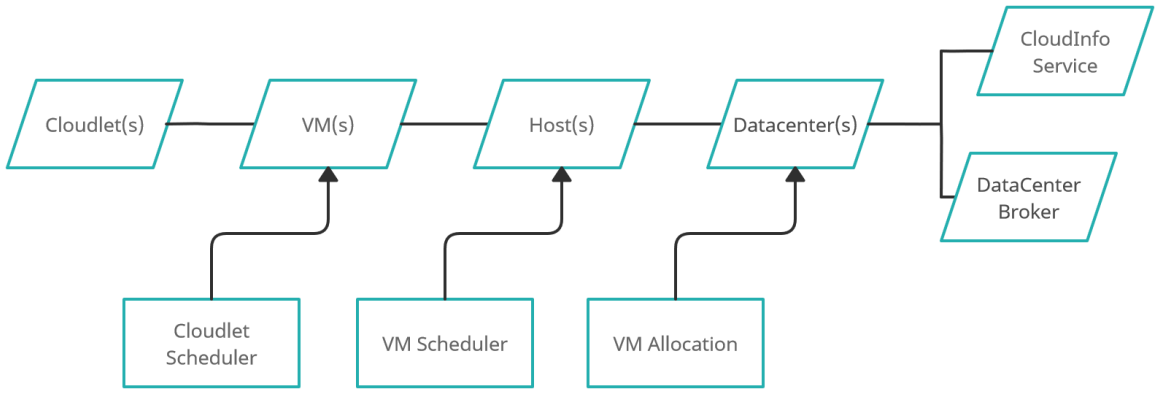
CloudSim is an open-source framework used for modelling and simulation of cloud computing infrastructures and services. It is developed by the CLOUDS Lab organization and is written entirely in Java.

It can successfully model and simulate infrastructures such as:

- Large-scale cloud computing datacenters
- Virtualized server hosts
- Application containers
- Energy-aware computational resources
- Datacenter network topologies

The major modal classes used in CloudSim simulation are:

- **Cloudlet:** Used to define a task to be scheduled in an environment.
- **Vm:** Used to define a virtual machine which will run the tasks in the simulation.
- **Host:** Used to define management of virtual machines.
- **Datacenter:** Used to define datacenter in the simulation.
- **DatacenterBroker:** Used to define as an entity which performs user actions.



Main parts involved in CloudSim

Methodology

Constants for Cloud Simulation

PARAMETERS	DESCRIPTION
POPULATION_SIZE	Number of particles for PSO Scheduler
NO_OF_ANTS	Number of ants for ACO Scheduler
NO_OF_GENERATIONS	Number of generations for ACO Scheduler
DATACENTER PARAMETERS	
NO_OF_DATACENTERS	Number of Datacenters
ARCHITECTURE	System Architecture
OS	Operating System
TIME_ZONE	Time zone of the Datacenter
COST_PROCESS	Cost of using processing in the Datacenter
COST_MEMORY	Cost of using memory in the Datacenter
COST_STORAGE	Cost of using storage in the Datacenter
COST_BANDWIDTH	Cost of using bandwidth in the Datacenter
HOST PARAMETERS	
STORAGE	Storage size (in MB)
HOST_MIPS	MIPS score for Host's Processing Element
HOST_RAM	RAM size (in MB)
HOST_BANDWIDTH	Bandwidth of Host (in Gbps)
VIRTUAL MACHINE PARAMETERS	
NO_OF_VMS	Number of VMs
VM_IMAGE_SIZE	VM Image size (in MB)
VM_RAM	RAM size (in MB)
VM_MIPS	MIPS capacity for VM's Processing Element
VM_BANDWIDTH	Bandwidth of VM (in Gbps)
VM_PES	Number of Processing Elements
VMM_NAME	Name of Virtual Machine Manager
CLOUDLET PARAMETERS	
NO_OF_TASKS	Number of Cloudlets
FILE_SIZE	Input file size (in Bytes) before execution
OUTPUT_SIZE	Output file size (in Bytes) after execution
TASK_PES	Number of PEs required to execute

Generating and Assigning Cloudlet length

- Length here implies number of instructions (in millions) to be executed.
- We create a matrix of size NO_OF_TASKS × NO_OF_DATACENTERS
- Every element is then assigned a value in the range [30, 730)
- This matrix is then saved into a text file for later use.
- Upon cloudlet creation, for every cloudlet, an element in the row is randomly selected if the scheduling algorithm is not PSO. If it is, then a custom mapping matrix is followed.

```
private void initCostMatrix() throws IOException {
    System.out.println("Initializing new Length Matrix...");
    BufferedWriter lengthBufferedWriter = new BufferedWriter(new FileWriter(lengthFile));

    for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
        for (int j = 0; j < Constants.NO_OF_DATACENTERS; j++) {
            lengthMatrix[i][j] = Math.random() * 700 + 30;
            lengthBufferedWriter.write(String.format("%.2f ", lengthMatrix[i][j]));
        }
        lengthBufferedWriter.write( c: '\n');
    }
    lengthBufferedWriter.close();
}
```

Procedure for Scheduling and Comparison

In every scheduler these steps are followed:

- Initialize CloudSim simulation with 3 parameters: number of cloud users (*num_user*), calendar instance with current date and time (*calendar*) and flag for tracing events (*trace_flag*)

```
public static void set_cloudsim_parameters(){
    num_user = 1;
    calendar = Calendar.getInstance();
    trace_flag = false;
}
```

```
Commons.set_cloudsim_parameters();
CloudSim.init(Commons.num_user, Commons.calendar, Commons.trace_flag);
```

- Create Datacenter(s) with the given parameters. Note that we have assigned only one Host and PE per Datacenter.

```
public static Datacenter createDatacenter(String name) {
    List<Host> hostList = new ArrayList<>();
    List<Pe> peList = new ArrayList<>();
    int mips = Constants.HOST_MIPS;
    peList.add(new Pe( id: 0, new PeProvisionerSimple(mips)));
    int hostId = 0;
    int ram = Constants.HOST_RAM;
    long storage = Constants.STORAGE;
    int bw = Constants.HOST_BANDWIDTH;
    hostList.add(new Host(hostId, new RamProvisionerSimple(ram),
            new BwProvisionerSimple(bw), storage,
            peList, new VmSchedulerTimeShared(peList)));
    String arch = Constants.ARCHITECTURE; String os = Constants.OS;
    String vmm = Constants.VMM_NAME; double time_zone = Constants.TIME_ZONE;
    double cost = Constants.COST_PROCESSING; double costPerMem = Constants.COST_MEMORY;
    double costPerStorage = Constants.COST_STORAGE; double costPerBw = Constants.COST_BANDWIDTH;
    LinkedList<Storage> storageList = new LinkedList<>();
    DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
            arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);
    Datacenter datacenter = null;
    try {
        datacenter = new Datacenter(name, characteristics,
                new VmAllocationPolicySimple(hostList), storageList,  schedulingInterval: 0);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return datacenter;
}
```

- Create a Datacenter Broker according to the scheduling algorithm.
- Create Virtual Machine(s) with the given parameters.

```
private static List<Vm> createVMs(int userId, int vms) {
    LinkedList<Vm> list = new LinkedList<>();

    Vm[] vm = new Vm[Constants.NO_OF_VMS];

    for (int i = 0; i < vms; i++) {
        vm[i] = new Vm(i, userId, Constants.VM_MIPS, Constants.VM_PES, Constants.VM_RAM,
                Constants.VM_BANDWIDTH, Constants.VM_IMAGE_SIZE,
                Constants.VMM_NAME, new CloudletSchedulerSpaceShared());
        list.add(vm[i]);
    }

    return list;
}
```

- Create Cloudlet(s) with the given parameters and *length*.

```
private static List<Cloudlet> createCloudlets(int userId, int cloudlets, int choice) {
    LinkedList<Cloudlet> list = new LinkedList<>();

    UtilizationModel umf = new UtilizationModelFull();

    Cloudlet[] cloudlet = new Cloudlet[cloudlets];

    for (int i = 0; i < cloudlets; i++) {
        int dcId;
        if(choice == 4)
            dcId = (int) (mapping[i]);
        else
            dcId = (int) (Math.random() * Constants.NO_OF_DATACENTERS);
        long length = (long) (1e3 * lengthMatrix[i][dcId]);

        cloudlet[i] = new Cloudlet(i, length, Constants.TASK_PES, Constants.FILE_SIZE,
                Constants.OUTPUT_SIZE, umf, umf, umf);
        cloudlet[i].setUserId(userId);
        cloudlet[i].setVmId(dcId + 2);
        list.add(cloudlet[i]);
    }
    return list;
}
```

- Submit the VM and Cloudlet list to the Datacenter Broker.
- Start the CloudSim simulation.
- Once there are no more events to execute, stop the CloudSim simulation.
- From the broker, fetch the Cloudlet list after execution.
- This list contains details related to the execution of Cloudlets like Execution Start Time and Finish Time.
- The total finish time is calculated.

Now for comparing the given scheduling algorithms, a Java file is present for executing them one-by-one and recording their finishing time in a Sorted Map for the same parameters.

```
public static void main(String[] args) throws InterruptedException {
    SortedMap<Double, String> map = new TreeMap<>();
    new GenerateLengthMatrix();
    Commons.lengthMatrix = GenerateLengthMatrix.getlengthMatrix();
    map.put(FCFS_Scheduler.main(args),  v: "First Come-First Serve");
    System.out.println("==========================================");
    TimeUnit.SECONDS.sleep( timeout: 1);
    map.put(SJF_Scheduler.main(args),  v: "Shortest Job First");
    System.out.println("==========================================");
    TimeUnit.SECONDS.sleep( timeout: 1);
    map.put(RoundRobin_Scheduler.main(args),  v: "Round Robin");
    System.out.println("==========================================");
    TimeUnit.SECONDS.sleep( timeout: 1);
    map.put(PSO_Scheduler.main(args),  v: "Particle Swarm Optimisation");
    System.out.println("==========================================");
    TimeUnit.SECONDS.sleep( timeout: 1);
    map.put(ACO_Scheduler.main(args),  v: "Ant Colony Optimisation");
    System.out.println("==========================================");
    TimeUnit.SECONDS.sleep( timeout: 1);
    System.out.println("Sorted list of algorithms (criteria: earliest finish time)");
    for(double i: map.keySet()){
        System.out.printf("%s: %.2f%n", map.get(i), i);
    }
    System.out.println("==========================================");
}
```

Tools and Frameworks Used

Language: Java

Java Libraries:

- cloudsim-3.0.3.jar – Contains all the model classes required for the cloud simulation.
- jswarm-pso_2_08.jar – Contains all the interfaces required (Particle, Fitness Function etc.) for simulating Particle Swarm Optimization. These interfaces are then extended to define our own classes.
- commons-math3-3.6.1.jar – Contains mathematics and statistics components, required when running a CloudSim project.

Results

For the given parameters:

PARAMETERS	VALUES
POPULATION_SIZE	25
NO_OF_ANTS	4
NO_OF_GENERATIONS	50
DATACENTER PARAMETERS	
NO_OF_DATACENTERS	5
ARCHITECTURE	“x86”
OS	“Linux”
TIME_ZONE	5.5
COST_PROCESS	3.0
COST_MEMORY	0.05
COST_STORAGE	0.001
COST_BANDWIDTH	0.1
HOST PARAMETERS	
STORAGE	1000000 MB
HOST_MIPS	1000
HOST_RAM	2048 MB
HOST_BANDWIDTH	10000 Gbps
VIRTUAL MACHINE PARAMETERS	
NO_OF_VMS	5
VM_IMAGE_SIZE	10000 MB
VM_RAM	512 MB
VM_MIPS	250
VM_BANDWIDTH	1000 Gbps
VM_PES	1
VMM_NAME	“Topa”
CLOUDLET PARAMETERS	
FILE_SIZE	3000 B
OUTPUT_SIZE	3000 B
TASK_PES	1

We run 5 iterations for NO_OF_TASKS = 10, 15, 20, 25, 30.

NO_OF_TASKS = 30

lengthMatrix:

```
 87.90 236.28 476.10 441.24 411.73
200.84 681.48 545.73  82.04 148.27
219.37 312.76 377.22 184.39 454.45
268.01 111.70 676.62 382.65 483.11
491.33 623.34 158.47 408.96 520.40
371.79 104.82 365.95 399.28 636.88
588.64  89.34 164.15 607.76 194.36
135.42 181.09 384.77  73.69 333.89
613.09  67.84  92.68 623.31 632.83
117.01  37.16 129.84 254.41 327.62
 73.78 144.48 505.15 106.43 297.97
221.54 152.44 400.34 129.92 459.61
 42.67 416.87 147.35 337.37 296.75
284.86 467.83 659.28  42.31 334.80
414.25 220.63  44.22 131.57 315.19
628.22 524.19 266.96 506.47 310.77
635.73 716.27 311.85 223.93 483.23
681.93 689.44 196.48 143.60 587.52
635.50  78.39 364.99 548.21 439.92
137.22  80.38 127.48 486.38 307.16
683.99 696.68 686.61 624.46 375.06
541.55 530.60 584.53 177.93 209.99
424.82 729.34 197.38 667.08 276.70
318.95 566.19 716.88  67.39 358.09
586.27 266.35  35.12 637.63 655.80
 43.86 268.75 511.77 113.16 444.05
485.06 264.25 513.55 252.27 534.57
721.75 186.01 404.43  48.46 648.85
197.30 663.24 703.73 707.48 375.08
438.58 245.67 183.83 114.99 548.38
```

```
==========================================
Sorted list of algorithms (criteria: earliest finish time)
Particle Swarm Optimisation: 6740.22
Ant Colony Optimisation: 11255.66
Round Robin: 12616.95
First Come-First Serve: 14581.90
Shortest Job First: 17657.92
==========================================
```

NO_OF_TASKS = 25

lengthMatrix:

```
276.95 451.56 331.83  80.96 392.00
 45.14 372.10  94.57 289.37 495.29
150.28 624.16 406.36 711.81 269.19
631.69 194.91 362.08 727.68 683.85
613.61 136.39 504.88  54.27 550.62
335.73 104.98 288.93 654.16 223.78
308.22 565.43 496.78 492.73 633.59
632.42 145.25 271.96 626.05 194.06
551.22 360.19 234.51 103.82 316.93
185.11 447.66 654.15 227.59 489.49
302.41 391.74 374.90 310.13  73.97
 54.02 127.03 654.80 397.83 609.74
427.28 393.12 462.43 295.70 128.01
 60.35 445.99 273.07 587.32 327.18
553.14 379.61 173.77 215.50 535.62
628.92  39.32 638.21 257.32 204.92
532.17 111.50 485.98 318.08 729.75
129.96 163.87 430.68 677.40 109.05
125.63 420.69 220.68 185.05 394.89
238.09 318.19 320.57 701.21 513.76
417.79 193.18 515.99 608.95 672.26
446.10 309.53 443.24 259.77 253.35
482.38 610.81  61.42 497.01 221.53
288.39 131.57 171.19 676.44 253.74
477.03 616.30 641.23 271.13 438.30
```

```
==========================================
Sorted list of algorithms (criteria: earliest finish time)
Particle Swarm Optimisation: 5258.80
Ant Colony Optimisation: 9241.99
Round Robin: 12340.82
Shortest Job First: 12669.66
First Come-First Serve: 13737.56
==========================================
```

NO_OF_TASKS = 20

lengthMatrix:

```
338.94 188.43  97.68 589.41 149.55
394.61 399.65 526.36 670.88 144.61
609.97  69.36  63.16 187.82 546.73
105.65 503.02 166.97 216.07 673.37
113.36 249.65 187.26 179.24 430.02
157.88 672.40 708.34 293.48  76.45
306.89 478.85 319.65 532.95  79.87
230.49 717.46  76.97 112.25 543.07
180.53 441.77 726.22 261.06 387.61
550.04 493.24 551.77 500.00 325.11
386.74 547.80 344.82 690.72 602.84
619.96 720.78 586.28 445.64 293.57
157.33 689.98 704.17 260.31 299.47
229.33 675.04 148.63 678.69 368.42
109.45 598.08 665.48 191.89 383.67
438.10 280.91 558.95 210.34 197.67
273.55 175.44 622.57 571.03 643.54
328.95 374.37  34.62 318.44 350.53
209.65 552.59 554.70 106.91 557.04
422.14 354.97 589.97 210.47  95.87
```

```
==========================================
Sorted list of algorithms (criteria: earliest finish time)
Particle Swarm Optimisation: 5573.03
Ant Colony Optimisation: 6320.04
Round Robin: 7080.73
Shortest Job First: 11845.37
First Come-First Serve: 12331.75
==========================================
```

NO_OF_TASKS = 15

lengthMatrix:

```
396.05 241.43 285.25 238.31 106.73
131.28 508.87 114.00 259.28 685.03
543.08 479.60  84.45 374.13 163.62
557.53 502.51 317.60 477.02 339.90
726.81 709.61 227.27 346.21 103.43
607.93 502.12 500.33 133.85 706.53
396.77 468.59 435.55 157.73 361.77
241.09 301.47 423.44 278.68 400.77
620.90 320.59 569.93 109.26 323.92
283.87  59.57 644.43 707.22 198.23
358.24 652.74 692.35 655.24 449.98
129.35 356.62 660.60 508.70 110.77
164.23 565.48 569.96 558.27 507.99
111.25 728.06 426.26 452.98  66.32
319.81 546.68 250.18 594.54 294.44
```

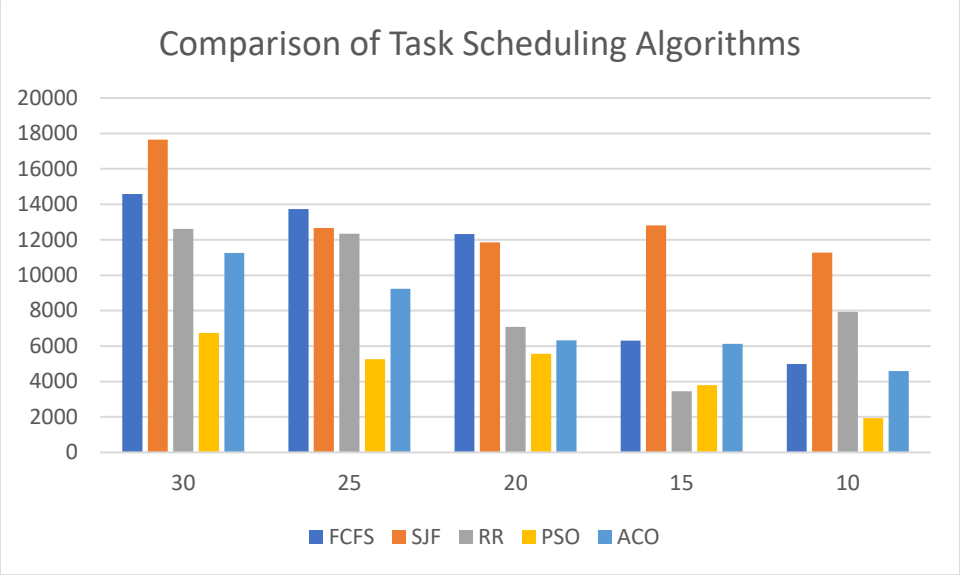
```
==========================================
Sorted list of algorithms (criteria: earliest finish time)
Round Robin: 3452.41
Particle Swarm Optimisation: 3798.34
Ant Colony Optimisation: 6125.02
First Come-First Serve: 6298.03
Shortest Job First: 12809.45
==========================================
```

NO_OF_TASKS = 10

lengthMatrix:

```
458.40 194.22 566.15 207.60 336.00
671.54  37.41 436.99 605.35  71.09
452.74 477.61 560.85 549.17 225.53
522.58 643.40 114.38 125.09 185.41
543.04 412.46 612.97 295.03  38.74
408.63 470.97 368.65 354.60 525.27
218.02 176.75 246.70 323.84 634.03
269.22 416.72 672.98 326.69 658.12
 77.42 219.72 291.90 150.67 470.66
377.38 330.32 396.69  61.04  62.50
```

```
==========================================
Sorted list of algorithms (criteria: earliest finish time)
Particle Swarm Optimisation: 1941.68
Ant Colony Optimisation: 4596.38
First Come-First Serve: 4993.32
Round Robin: 7940.08
Shortest Job First: 11274.74
==========================================
```



Therefore, the ranking of task scheduling algorithms from best to worst is:

#1 Particle Swarm Optimization Scheduling
#2 Ant Colony Optimization Scheduling
#3 Round Robin Scheduling
#4 First Come First Serve Scheduling
#5 Shortest Job First Scheduling

Conclusion

We have successfully analyzed and compared different task scheduling algorithms in a cloud computing simulation. We have found out that whatever may be the number of tasks at hand, evolutionary algorithms work exceptionally well as compared to their traditional counterparts, with Particle Swarm Optimization and Ant Colony Optimization holding #1 and #2 rank respectively.

References

1. S. Bilgaiyan, S. Sagnika and M. Das, "Workflow scheduling in cloud computing environment using Cat Swarm Optimization," 2014 IEEE International Advance Computing Conference (IACC), 2014, pp. 680-685, doi: 10.1109/IAdCC.2014.6779406.
2. http://www.cloudbus.org/cloudsim/
3. http://jswarm-pso.sourceforge.net/
4. https://commons.apache.org/proper/commons-math/