INDIAN INSTITUTE OF TECHNOLOGY (BHU), VARANASI

An Exploration into the World of Cloud and Fog Computing

Authors

Bhanu Verma - 22075021
Devashish Raj - 22075028

Under the Guidance of **Dr. Ravi Shankar Singh**

Contents

1 Introduction

1.1 Cloud Computing

Cloud computing has emerged as a transformative technology that provides on-demand access to a wide range of computing resources, including processing power (CPU), memory, and storage, with minimal overhead and operational complexity. This innovative approach to computing has revolutionized the way organizations and individuals access and utilize computing resources, enabling them to scale their infrastructure up or down based on their immediate needs.

Cloud computing offers three primary service models:

- **Infrastructure as a Service (IaaS)**: IaaS offers users virtual infrastructure, including computing power, storage, and networking resources, on-demand. Users can easily provision and manage virtual machines, storage volumes, and network configurations, without the need to invest in and maintain physical hardware.

- **Platform as a Service (PaaS)**: PaaS provides users with a platform for developing, testing, and deploying applications without the need to manage the underlying infrastructure. This service model abstracts away the complexities of server management, operating system maintenance, and software updates, allowing developers to focus on building and deploying their applications.

- **Software as a Service (SaaS)**: SaaS provides users with access to software applications hosted on the cloud. These applications are accessible through web browsers or dedicated client software, eliminating the need for users to install and maintain the software on their local devices. SaaS offerings often include features such as automatic updates, scalability, and multi-tenancy, making them attractive for businesses and individuals alike.

One of the key advantages of cloud computing is the ease with which users can deploy applications without the burden of managing the underlying hardware and software infrastructure. Cloud providers handle the provisioning, maintenance, and scaling of the necessary resources, allowing users to focus on their core business objectives. Additionally, cloud computing enables a pay-as-you-go pricing model, where users only pay for the resources they consume, reducing upfront costs and providing flexibility in resource allocation.

A recent trend in cloud computing is the emergence of serverless computing, which takes the concept of abstraction one step further. In a serverless architecture, cloud providers manage the server infrastructure entirely, including provisioning, scaling, and maintaining the necessary resources. Users simply deploy their code, and the cloud provider automatically manages the execution of the code, scaling resources up or down based on demand. This approach allows developers to focus solely on writing and deploying their application logic, without worrying about server management or capacity planning.

Cloud computing has transformed the way organizations and individuals access and utilize computing resources, offering on-demand scalability, reduced operational complexity, and cost-effective pricing models. As the adoption of cloud computing continues to grow, it is expected to drive further innovation and enable new use cases across various industries and domains.

1.2 Fog Computing

Fog computing extends cloud computing to the edge of networks, bringing low-latency processing closer to users. By leveraging this approach, real-time applications can be facilitated, and local resources like user devices can be utilized more effectively, potentially enhancing IoT capabilities and enabling faster processing. This paradigm shift towards edge computing not only reduces latency but also enhances the overall efficiency of processing tasks by utilizing resources in proximity to end-users.

In the context of fog computing, the integration of local resources and edge devices plays a crucial role in optimizing processing efficiency and enhancing the performance of real-time applications. By harnessing the computing power of devices at the edge of the network, fog computing enables faster data processing and response times, leading to improved user experiences and enhanced system performance. This utilization of local resources not only reduces the reliance on distant cloud platforms but also opens up new possibilities for innovative applications that require rapid data processing and real-time decision-making capabilities.

Fog computing represents a significant advancement in distributed systems by extending cloud computing to the edge of networks. By bringing processing capabilities closer to users, this approach enables real-time applications, leverages local resources effectively, and enhances IoT capabilities by facilitating faster processing and reducing latency. The integration of local resources and edge devices in fog computing not only optimizes processing efficiency but also paves the way for innovative applications that require rapid data processing and real-time decision-making capabilities.

1.3 Edge Computing

Edge computing involves the processing of data in close proximity to its source, typically at devices and sensors, to achieve ultra-low latency. This approach facilitates the execution of real-time applications and diminishes the dependence on centralized cloud resources. Devices such as routers, switches, and IoT devices assume the role of miniature computing centers at the periphery of the network, enhancing the efficiency and responsiveness of data processing tasks.

1.4 Workflow

A workflow can be formally defined as a Directed Acyclic Graph (DAG) that represents a set of tasks or activities that need to be executed in a specific order, adhering to a predefined set of dependencies and constraints. This formal representation of a workflow can be denoted as G(V, E), where V represents the set of all the tasks or activities, and E represents the set of dependencies between them.

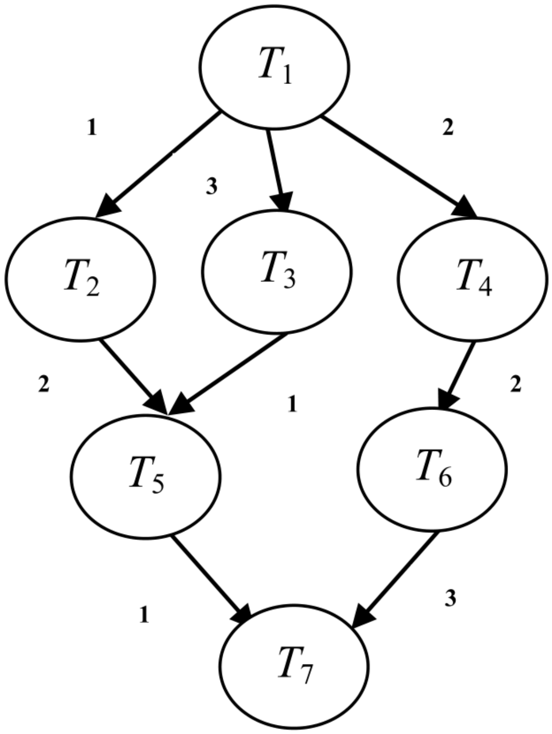


Figure 1: An Example Workflow

In this formal representation, each vertex in the graph corresponds to a specific task or activity that needs to be performed as part of the workflow. The edges in the graph represent the dependencies between these tasks, indicating the order in which they must be executed. If (T_g, T_h) is an element of E, it implies that task T_g is the predecessor (pred) of task T_h, and task T_h is the successor (succ) of T_g. In such a case, task T_g needs to be executed first in order to enable the execution of task Th.

Furthermore, the weight associated with each edge in the graph represents the communication time required between the two connected tasks. This weight factor is crucial in determining the overall execution time and efficiency of the workflow, as it accounts for the time needed to transfer data or information between dependent tasks.

By formally representing a workflow as a directed acyclic graph, it becomes possible to analyze and optimize the execution of tasks within the workflow. This formal representation allows for the identification of critical paths, the detection of potential bottlenecks, and the implementation of scheduling algorithms to ensure efficient and timely execution of the workflow.

2 Workflow Scheduling

2.1 Scheduling Problem

The Workflow Scheduling Problem is a fundamental challenge in the field of distributed computing, particularly in the context of cloud computing and parallel processing environments. This problem revolves around the efficient allocation of tasks within a workflow to available computing resources while considering various constraints such as task dependencies, resource availability, communication overhead, and user-defined Quality of Service (QoS) requirements. It is an NP-hard problem. The primary objective of solving the Workflow Scheduling Problem is to optimize the overall workflow execution time, resource utilization, and cost-effectiveness, ensuring that the workflow is executed in a timely and efficient manner while meeting the specified QoS criteria. This complex optimization task involves determining the best scheduling strategy to minimize makespan, maximize resource utilization, and meet performance objectives such as deadline constraints and budget limitations. Various algorithms, heuristics, and optimization techniques, including meta-heuristics, approximation algorithms, and machine learning approaches, are employed to address the Workflow Scheduling Problem. These methods aim to achieve efficient task allocation, load balancing, and improved system performance in distributed computing environments, ultimately enhancing the overall productivity and cost-effectiveness of workflow execution in cloud and parallel processing systems.

2.2 Resource Provisioning

Resource provisioning is a crucial component in the efficient management of workflows in cloud environments. It involves selecting and allocating resources based on Quality of Service (QoS) parameters to ensure optimal performance and resource utilization. This phase aims to balance loads among servers and efficiently utilize computing resources. The function of resource provisioning can be defined as

$$\text{Resource provisioning(r prov)} : \mathrm{I}_{\mathrm{i}} \rightarrow S_{\mathrm{j}}.$$
$$\text{where, } \mathrm{I}_{\mathrm{i}} = \text{A VM instance, } \mathrm{S}_{\mathrm{j}} = \text{A Server instance}$$

The selection of resources is influenced by various factors such as the capacity of the server, the execution time of tasks on VM instances, and the required energy to execute tasks on VM instances. The capacity of each VM instance is a critical factor in determining the suitability of a resource for a task.

2.3 Task Scheduling

Task scheduling is a critical phase in the management of workflows in cloud environments, focusing on determining the optimal order of tasks to meet scheduling objectives efficiently. Task schedulers employ heuristic or dynamic strategies to minimize the makespan and total execution cost of workflows while adhering to various Quality of Service (QoS) constraints. The main goal of task scheduling is to assign tasks in a manner that optimizes resource utilization and system performance. It plays a vital role in balancing loads among servers and ensuring tasks are executed in a timely and cost-effective manner. By considering factors such as task dependencies, resource capacities, and communication overhead, task schedulers aim to create schedules that maximize efficiency and meet performance metrics. The function of task scheduling can be represented as

$$\text{Task Scheduling (t\_ sch)} : \mathrm{T_k} \rightarrow I_{\mathrm{i}}.$$

where, $\mathrm{T_k}$ = Tasks of workflow, $\mathrm{I_i}$ = A VM instance

2.4 Scheduling Objectives

The Scheduling Objectives refer to the key goals and metrics that guide workflow scheduling strategies in cloud and fog environments. These objectives include:

- **Total Execution Time (Makespan)**: The total execution time or makespan of a workflow is the time required to complete all its tasks on the selected VM instances. It is the completion time of the last task of the workflow. The objective is to minimize the total execution time of a workflow by efficiently executing tasks on selected VM instances. This involves considering the size of tasks and the resource capacity of the VM instances to optimize workflow performance.

- **Total Execution Cost:** The total execution cost of a workflow depends on three factors: (a) required number of VM instances for running the tasks of a workflow; (b) resource capacity of each VM instance; and (c) task-VM mapping strategy. An optimal schedule needs to assign the tasks on the minimum number of VM instances with feasible resource capacity using a suitable task-VM mapping strategy. This may minimize the overall execution cost of a work-flow.

- **Energy Consumption**: Energy consumption refers to the amount of energy consumed by computing resources while executing the tasks of a workflow. Minimizing the energy consumption of a workflow is a critical challenge in cloud workflow scheduling, as it can significantly reduce the carbon footprint of the cloud data centers and the operating costs of running the infrastructure. Industries, organizations, and governments are increasingly focused on reducing energy consumption and carbon emissions.

3 FogWorkflowSim

3.1 Precursors to FogWorkflowSim

3.1.1 CloudSim

CloudSim is an open-source framework that allows researchers and developers to simulate cloud computing environments and evaluate the performance of various resource provisioning and scheduling algorithms. It provides a comprehensive modeling framework that enables seamless modeling, simulation, and experimentation of emerging cloud computing infrastructures and application services. CloudSim works by simulating a list of jobs, given in the form of cloudlets, which represent the tasks or applications that need to be executed in the cloud environment. These cloudlets are submitted to a broker, which is responsible for managing the execution of the cloudlets on the available virtual machines (VMs) in the simulated cloud infrastructure. However, there is no support for Fog environments or Workflow scheduling in CloudSim.

3.1.2 WorkflowSim

WorkflowSim is a toolkit designed for simulating and analyzing scientific workflows in distributed cloud environments. It offers a powerful platform for researchers and developers to model, simulate, and evaluate various workflow scheduling strategies, resource provisioning algorithms, and task execution models. By leveraging WorkflowSim, users can explore the intricate dynamics of scientific workflows, experiment with different scheduling algorithms, and analyze their impact on workflow execution time, cost, and resource utilization. This toolkit provides valuable insights into optimizing workflow management strategies, enhancing efficiency and effectiveness in cloud environments. However, there is no support for Fog computing environments in WorkFlowSim.

3.1.3 iFogSim

iFogSim is an open-source toolkit tailored for simulating and assessing resource management strategies within edge and fog computing environments. This specialized toolkit provides researchers and developers with a robust platform to delve into the intricacies of resource management in edge and fog computing settings. By offering a detailed simulation environment for individual tasks, iFogSim enables users to explore and optimize resource allocation strategies, enhancing the efficiency and performance of edge and fog computing systems. However, there is no support for Workflow scheduling in iFogSim.

3.2 FogWorkflowSim Simulator

FogWorkflowSim is an open-source toolkit for the simulation of workflows on Fog Computing environments, written in Java. It extends the previous tools, and is built upon CloudSim, WorkflowSim and iFogSim packages. It also provides a Graphical User Interface (GUI) for ease of simulation.

FogWorkflowSim defines three main device classes: Cloud, Fog, and Mobile. These classes represent the various components of a Fog Computing architecture, allowing users to model and configure the different types of resources available in the system.

The Fog Device class plays a crucial role in FogWorkflowSim, as it encapsulates the representation of all resources within the Fog layer. This class enables users to simulate a wide range of Fog devices by adjusting their specifications, such as MIPS (Million Instructions Per Second), storage capacity, and bandwidth. By providing this flexibility, FogWorkflowSim allows for the creation of realistic Fog Computing environments that closely resemble real-world scenarios.

The simulation process in FogWorkflowSim is initiated by the Planner module, which coordinates the execution of workflows on the simulated Fog and Cloud resources. The Parser module, on the other hand, is responsible for parsing the input workflow XML file and converting it into Task class objects. These Task class objects represent individual tasks within the simulated workflow, enabling the toolkit to accurately model the dependencies and execution requirements of each task.

The resource management layer comprises the Resource module, Offloading module, Scheduling module, and Controller module. The Resource module functions as a virtualized pool housing computation and storage resources of diverse types. The Offloading and Scheduling modules serve as adaptable libraries offering a range of computation offloading strategies and task scheduling algorithms within the system. Meanwhile, the Controller module serves as a foundation for the models of different performance metrics utilized to assess the efficiency of active workflow applications.

3.2.1 Limitations/Drawbacks of FogWorkflowSim

There are several Problems in the FogWorkflowSim Simulator that persist even in the latest release of the simulator. Some of these are:

- No offloading strategy, other than the simple offloading strategy, has been implemented in the Simulator.
- Many features and elements have been hard-coded into the source code of the simulator, no loose coupling.
- No Support for removing edge nodes, forced to include atleast one mobile device.
- No option to use the Static local scheduling algorithm.

4 Scheduling Algorithms

4.1 Genetic Algorithm (GA)

Genetic Algorithm Scheduling, particularly in the context of Fog Computing, offers a powerful approach to address the complex optimization challenges inherent in dynamic and heterogeneous environments. Leveraging principles inspired by biological evolution, Genetic Algorithms iteratively generate and refine potential scheduling solutions based on the principles of natural selection, crossover, and mutation. In Fog Computing, where computing resources are distributed across a network of heterogeneous devices, Genetic Algorithms excel in optimizing task allocation, resource utilization, and Quality of Service (QoS) parameters such as latency and energy efficiency. By iteratively evolving schedules over multiple generations, Genetic Algorithms adapt to changing environmental conditions, workload dynamics, and resource constraints, ultimately producing robust and efficient scheduling solutions tailored to the specific requirements of Fog Computing applications. This approach not only enhances the performance and scalability of Fog Computing systems but also facilitates the realization of real-time, resource-efficient, and responsive services in dynamic and unpredictable environments.

4.1.1 Pseudocode

Algorithm 1 Genetic Algorithm

```
Initialize population $P$ with random individuals
Evaluate fitness of each individual in $P$
while termination condition not met do
    Select parents from $P$ based on fitness
    Apply genetic operators (e.g., crossover, mutation) to create offspring
    Evaluate fitness of offspring
    Select individuals for the next generation (e.g., elitism, tournament selection)
    Replace the current population $P$ with the selected individuals
end while
return Best individual found in $P$
```

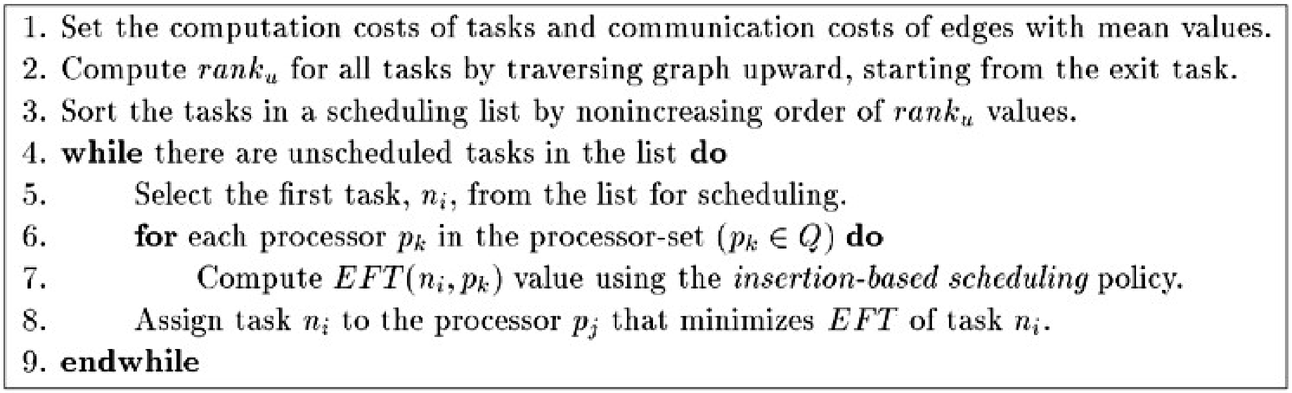
4.2 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) Scheduling Algorithm offers a dynamic and efficient approach to address scheduling challenges in Fog Computing environments. Inspired by the collective behavior of swarms, PSO optimizes scheduling solutions by simulating the movement of particles through a search space. In the context of Fog Computing, where resources are distributed across heterogeneous nodes, PSO excels in optimizing task allocation, resource utilization, and Quality of Service (QoS) parameters such as latency and energy efficiency. By iteratively updating the velocity and position of particles based on their own experience and the collective knowledge of the swarm, PSO adapts to changing environmental conditions and evolving workload demands. This adaptability makes PSO well-suited for dynamic and unpredictable Fog Computing environments, where traditional optimization approaches may struggle. PSO Scheduling Algorithm enhances the scalability and performance of Fog Computing systems, facilitating the efficient allocation of resources and the delivery of real-time, responsive services to end-users.

4.2.1 Pseudocode

Algorithm 2 Particle Swarm Optimization

```
Initialize particles with random positions and velocities
Evaluate fitness of each particle
Initialize personal best positions for each particle
Initialize global best position based on the particle with the best fitness
while termination condition not met do
    for each particle do
        Update velocity using personal and global best positions
        Update position based on velocity
        Evaluate fitness of the new position
        if current position is better than personal best position then
            Update personal best position
        end if
        if current position is better than global best position then
            Update global best position
        end if
    end for
end while
return Global best position found
```

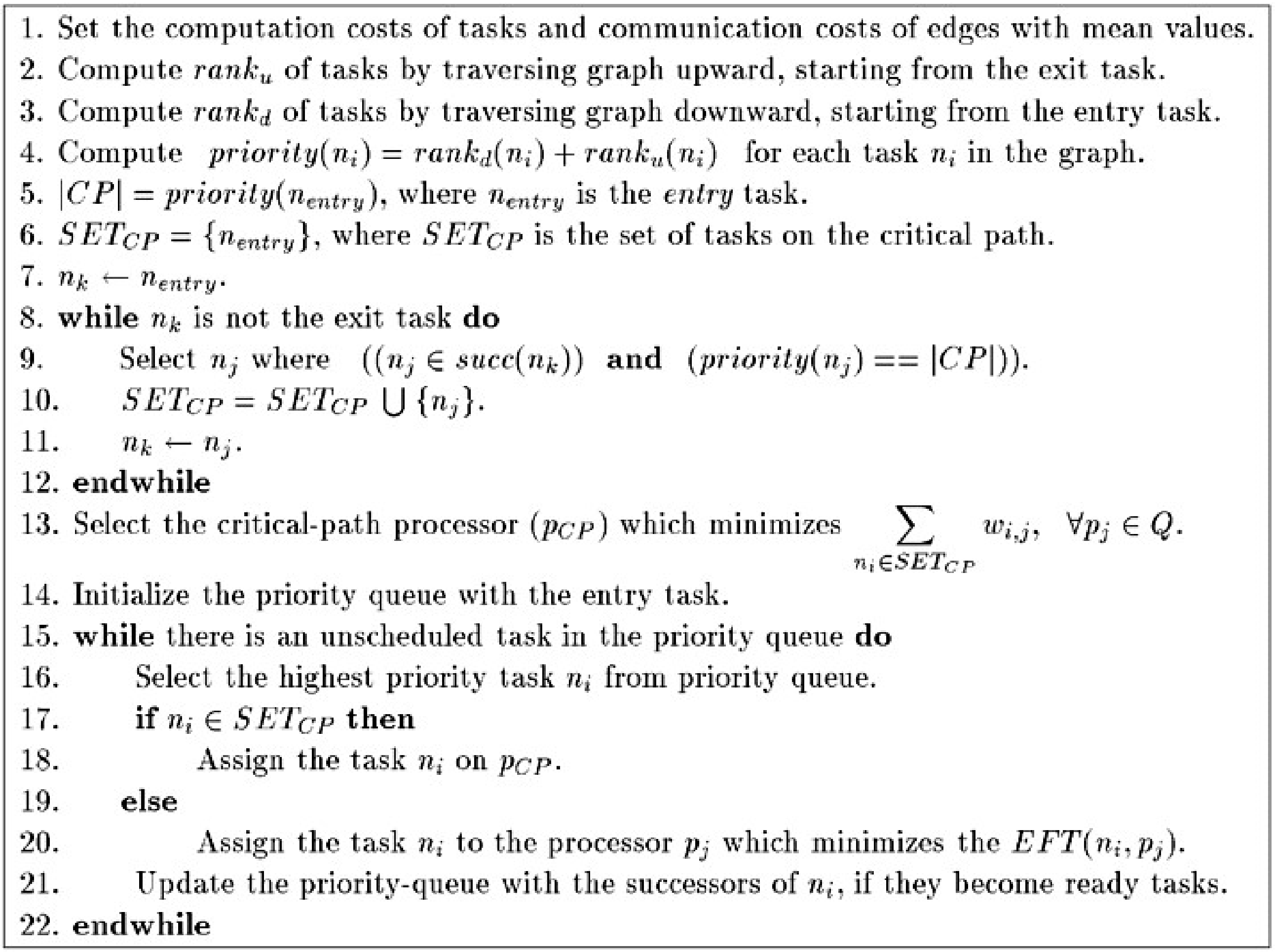


```
Set the computation costs of tasks and communication costs of edges with mean values.
Compute $rank_u$ for all tasks by traversing graph upward, starting from the exit task.
Sort the tasks in a scheduling list by nonincreasing order of $rank_u$ values.
while there are unscheduled tasks in the list do
    Select the first task, $n_i$, from the list for scheduling.
    for each processor $p_k$ in the processor-set ($p_k \in Q$) do
        Compute $EFT(n_i, p_k)$ value using the insertion-based scheduling policy.
    Assign task $n_i$ to the processor $p_j$ that minimizes $EFT$ of task $n_i$.
endwhile
```

Figure 2: HEFT (Heterogeneous Earliest Finish Time)

4.3 Heterogeneous Earliest Finish Time (HEFT)

Heterogeneous Earliest Finish Time (HEFT) is a widely utilized algorithm in task scheduling, particularly in the domain of Fog Computing. In Fog Computing, where resources are distributed across edge devices and cloud servers, efficient task allocation is critical for optimizing performance and minimizing latency. HEFT excels in this context by intelligently assigning tasks to the most suitable computing resources based on their processing capabilities and proximity to the data source. By prioritizing tasks with earliest finish times and considering both computation and communication costs, HEFT ensures optimal resource utilization and minimizes overall execution time in Fog Computing environments. Its adaptability to heterogeneous environments makes HEFT a valuable tool for enhancing the efficiency and responsiveness of fog-based applications.



```
Set the computation costs of tasks and communication costs of edges with mean values.
Compute $rank_u$ of tasks by traversing graph upward, starting from the exit task.
Compute $rank_d$ of tasks by traversing graph downward, starting from the entry task.
Compute $priority(n_i) = rank_d(n_i) + rank_u(n_i)$ for each task $n_i$ in the graph.
$|CP| = priority(n_{entry})$, where $n_{entry}$ is the entry task.
$SET_{CP} = \{n_{entry}\}$, where $SET_{CP}$ is the set of tasks on the critical path.
$n_k \leftarrow n_{entry}$.
while $n_k$ is not the exit task do
    Select $n_j$ where $((n_j \in succ(n_k))$ and $(priority(n_j) == |CP|))$.
    $SET_{CP} = SET_{CP} \bigcup \{n_j\}$.
    $n_k \leftarrow n_j$.
endwhile
Select the critical-path processor ($p_{CP}$) which minimizes $\sum_{n_i \in SET_{CP}} w_{i,j}, \ \forall p_j \in Q$.
Initialize the priority queue with the entry task.
while there is an unscheduled task in the priority queue do
    Select the highest priority task $n_i$ from priority queue.
    if $n_i \in SET_{CP}$ then
        Assign the task $n_i$ on $p_{CP}$.
    else
        Assign the task $n_i$ to the processor $p_j$ which minimizes the $EFT(n_i, p_j)$.
    Update the priority-queue with the successors of $n_i$, if they become ready tasks.
endwhile
```

Figure 3: CPOP (Critical-Path-on-a-Processor)

4.4 Critical-Path-on-a-Processor (CPOP)

Critical-Path-on-a-Processor (CPOP) Planning Algorithm offers a tailored solution for task scheduling in Fog Computing, where resources are decentralized across edge devices and cloud servers. CPOP optimizes task allocation by identifying critical paths within the workflow and allocating them to the most appropriate processing nodes. By prioritizing critical tasks, which significantly impact the overall completion time, CPOP ensures efficient resource utilization and minimizes latency in fog-based applications. Its ability to dynamically adapt to changing network conditions and resource availability makes CPOP well-suited for the dynamic and heterogeneous nature of fog environments. Moreover, CPOP's focus on critical paths allows it to effectively manage workload distribution, ensuring that resources are allocated optimally to meet application performance requirements.

4.5 Difference between HEFT and CPOP

	HEFT	**CPOP**
Approach	HEFT is a heuristic algorithm that aims to minimize the overall execution time of a set of tasks on a parallel system by assigning tasks to processors based on their estimated completion times.	CPOP, on the other hand, is a deterministic algorithm that schedules tasks on processors with the goal of minimizing the critical path length of the computation, hence the name "Critical-Path-On-a-Processor".
Objective	The primary objective of HEFT is to minimize the makespan, which is the total time taken to complete all tasks in a parallel application.	CPOP focuses on minimizing the length of the critical path, which represents the longest path through the task graph and is indicative of the minimum time required to execute the parallel application.
Complexity	HEFT is generally more computationally expensive compared to CPOP because it involves estimating task completion times and iteratively assigning tasks to processors based on these estimates.	CPOP is often simpler in terms of computation because it primarily focuses on identifying the critical path and scheduling tasks to minimize its length.
Determinism	HEFT is a heuristic algorithm, meaning it may not always produce an optimal solution but aims to provide a good solution in a reasonable amount of time.	CPOP is a deterministic algorithm, which means it will always produce the same solution given the same input and conditions. It is designed to find an optimal solution for minimizing the critical path length.
Performance	HEFT tends to perform well in a variety of scenarios and is often used in practice due to its effectiveness in reducing makespan.	CPOP may perform better in scenarios where minimizing the critical path length is crucial, such as applications with strict latency requirements or where resource constraints are tight.

5 Experimental Setup

5.1 Experimental Setup

The experiment has been performed and results were noted for the following setup:

- Number of Cloud Devices = 2
  - Host1: 2600 MIPS, 0.8 Cost
  - Host2: 2700 MIPS, 0.9 Cost
- Number of Fog Devices = 4
  - Host1: 1000 MIPS, 0.25 Cost
  - Host2: 1100 MIPS, 0.3 Cost
  - Host3: 1200 MIPS, 0.35 Cost
  - Host4: 1300 MIPS, 0.4 Cost
- Number of Mobile Devices = 1
  - Host1: 800 MIPS, 0 Cost

```
    // Env.xml

    <?xml version="1.0" encoding="UTF−8"?>
    <EnvironmentSetting>
        <Cloud name="cloud" hostnum="2" bandwidth="40">
            <host name="host−4" MIPS="2600" cost="0.8"></host>
            <host name="host−5" MIPS="2700" cost="0.9"></host></Cloud>
        <FogNode name="fog" hostnum="4" bandwidth="100">
            <host name="host−1" MIPS="1000" cost="0.25"></host>
            <host name="host−2" MIPS="1100" cost="0.3"></host>
            <host name="host−3" MIPS="1200" cost="0.35"></host>
            <host name="host−4" MIPS="1300" cost="0.4"></host></FogNode>
        <EndDevice name="cloud" hostnum="1" bandwidth="0">
            <host name="host−1" MIPS="800" cost="0"></host></EndDevice>
    </EnvironmentSetting>
```

- Offloading Strategy = Simple
- Primary Objective = Makespan (Total Execution Time)
- Secondary Objectives = Total Execution Cost, Energy Consumption

5.2 Changes in FogWorkflowSim Classes

The following changes were made in the FogWorkflowSim Simulator, in order to add the new algorithms:

1. In the parameters.java file, present at org.workflowsim.utils, add the name of the algorithm

```
    // Parameters.java

    public enum PlanningAlgorithm{
        INVALID, RANDOM, HEFT, DHEFT, CPOP
    }
```

2. In org.workflowsim.planning, make a new class for the algorithm, which inherits from the BasePlanningAlgorithm class.

```
    // CPOPPlanningAlgorithm.java

    public class CPOPPlanningAlgorithm extends BasePlanningAlgorithm {
        // Contents of the Algorithm
    }
```

3. Create a new case in the WorkflowPlanner.java file, present at org.workflowsim, and include the class for the new algorithm.

```
    // WorkflowPlanner.java

    private BasePlanningAlgorithm getPlanningAlgorithm(PlanningAlgorithm
       name) {
        BasePlanningAlgorithm planner;



        // choose which scheduler to use. Make sure you have add related
           enum in
        //Parameters.java
        switch (name) {
            //by default it is FCFS_SCH
            case INVALID:
                planner = null;
                break;
            case RANDOM:
                planner = new RandomPlanningAlgorithm();
                break;
            case HEFT:
                planner = new HEFTPlanningAlgorithm();
                break;
            case DHEFT:
                planner = new DHEFTPlanningAlgorithm();
                break;
            case CPOP:  // Added New Case
                planner = new CPOPPlanningAlgorithm();
                break;
            default:
                planner = null;
                break;
        }
        return planner;
    }
```

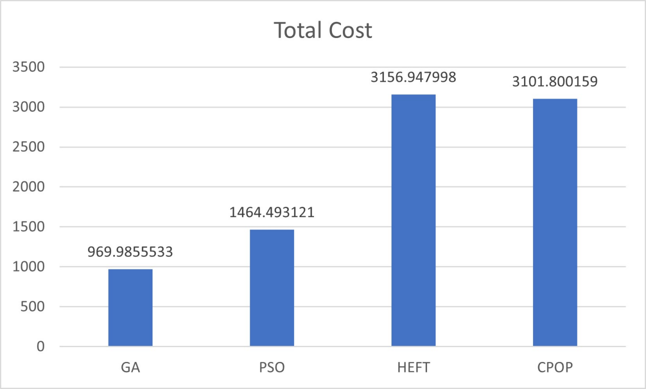
4. Change the planner in the main function of the program.

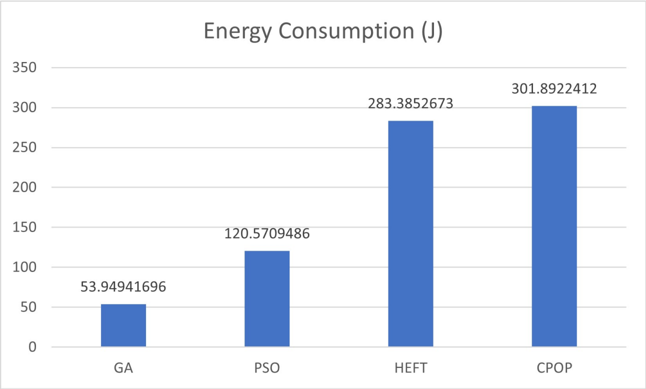
```
        // Example1.java

        Parameters.PlanningAlgorithm pln_method = Parameters.PlanningAlgorithm.CPOP;
```

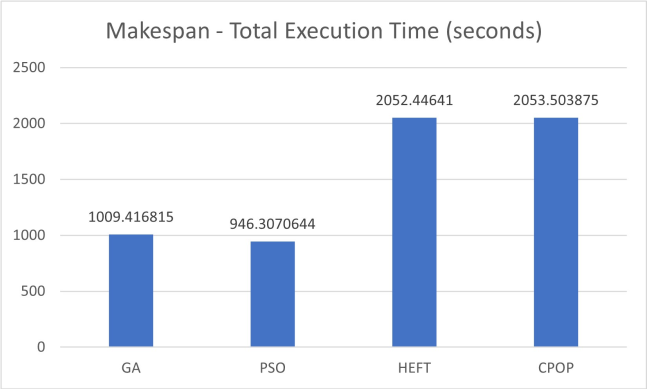
6 Results (Fogworkflowsim)

The following results are obtained upon running the Genetic Algorithm (GA), the Particle Swarm Optimization (PSO), the Heterogeneous Earliest Finish Time (HEFT) and the Critical-Path-on-a-Processor (CPOP) Algorithms on the FogWorkflowSim Simulator for multiple Scientific Workflows, namely: Montage-300, CyberShake-1000 and Inspiral-1000

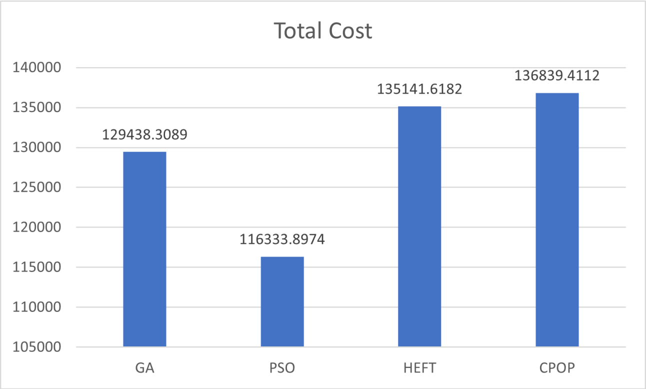
6.1 Montage-300

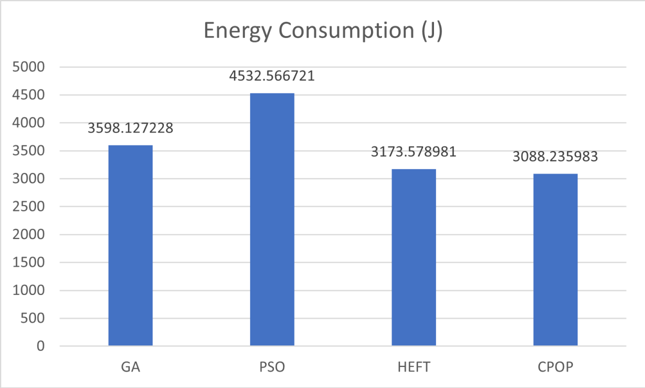


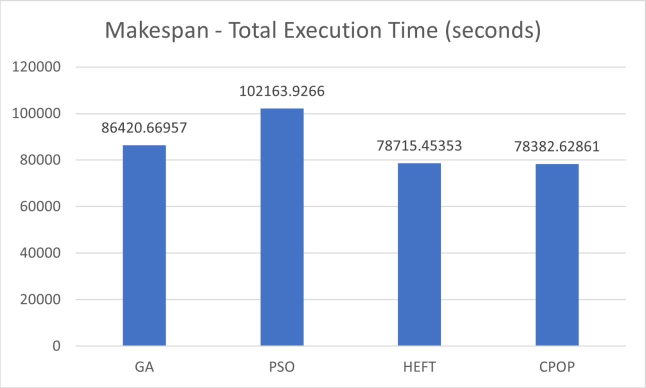




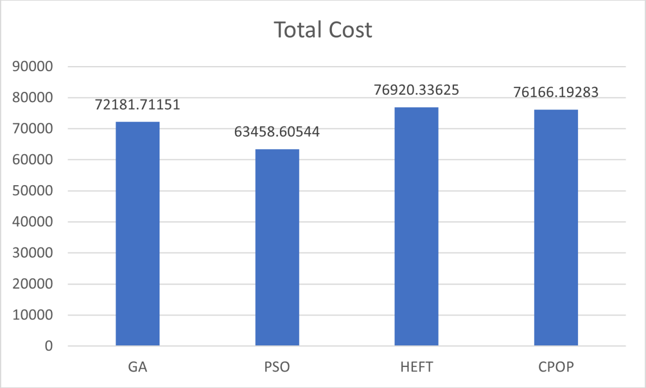
6.2 CyberShake-1000

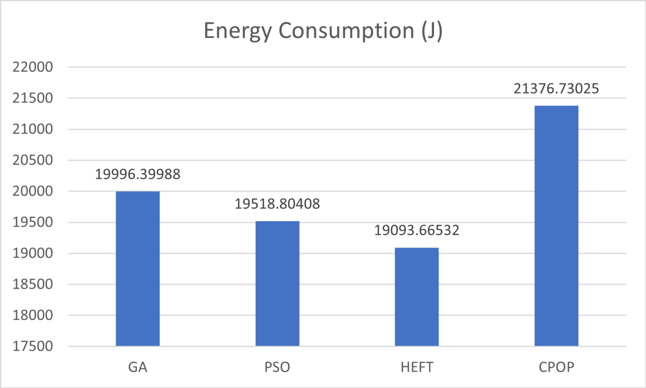


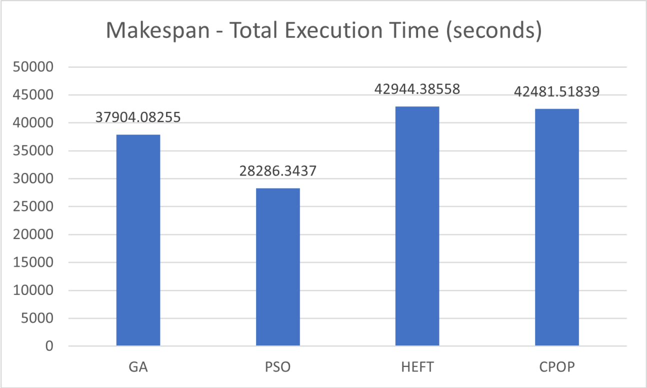




6.3 Inspiral-1000







6.4 Observations

- The PSO Algorithm works optimally for almost all datasets. The cost and makespan of PSO is consistently better than other algorithms, for all datasets. However, the energy consumption for PSO is higher.

- The HEFT and CPOP algorithms yield an almost similar result, with CPOP having better costs and HEFT having better makespan.

- GA performs exceptionally well on Montage-300. However, it seems to lag behind PSO for bigger datasets, ie, CyberShake-1000 and Inspiral-1000.

7 COSCO

7.1 Simulation Toolkit - COSCO

A basic design of the COSCO framework is shown in Fig. 1. The hosts in this architecture can be virtual hosts or real computing nodes, each having memory, disk, bandwidth, and instructions per second (IPS) capacity as well as power consumption models that show CPU usage. In simulation situations, workloads can be time-series models that show RAM, disk, bandwidth, and IPS needs; in practical tests, workloads can be actual application programs. Workloads are produced at each scheduling interval's beginning. Furthermore, the system keeps a waiting list for tasks that were not able to be assigned in the previous time frame.

The Simulator employs a Scheduler that utilizes utilization metrics and Quality of Service (QoS) parameters to make allocation decisions for new workloads or migration decisions for existing ones. At the beginning of each scheduling interval, completed workloads from the previous interval are terminated by the simulator. Instead of executing tasks on physical machines, we employ trace-driven discrete-event simulation.

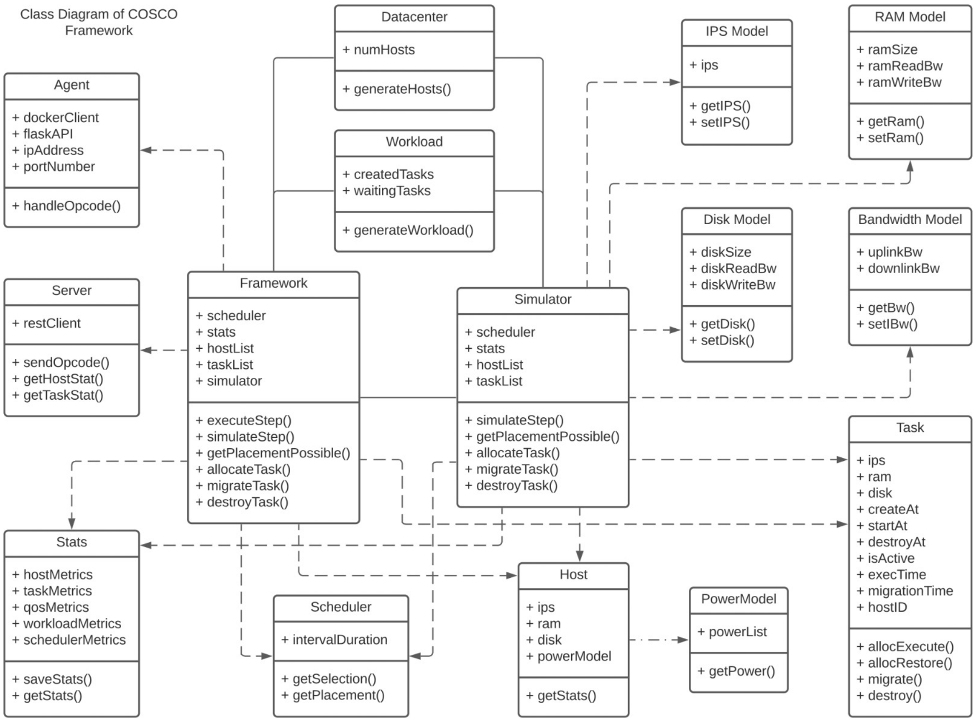


Figure 4: Class Diagram of COSCO

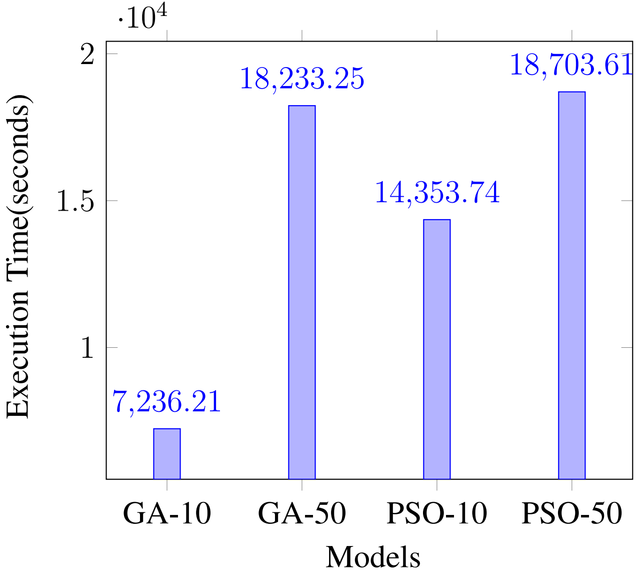
7.2 Installing COSCO

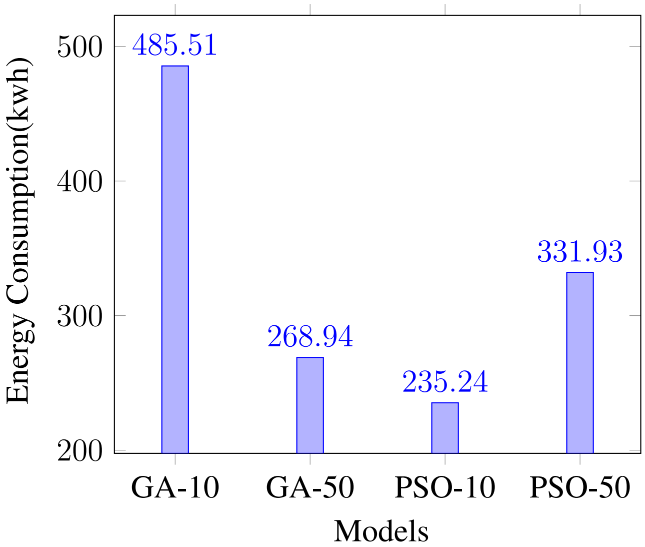
Run the following commands on the terminal to get COSCO installed on your computer-

```
sudo apt update
sudo add-apt-repository ppa:deadsnakes/ppa  get deadsnakes
sudo apt-get update
sudo apt install python3.7 python3.7-pip python3.7-distutils python3.7-dev python3.7-venv get the older version of python
mkdir Explo
cd Explo
python3.7 -m venv virtualenv
source ./virtualenv/bin/activate
sudo apt install git
git clone https://github.com/imperial-qore/COSCO
cd COSCO
python3 install.py
python3 main.py
```

8 Results (COSCO)

The following results are obtained upon running the Genetic Algorithm (GA) and the Particle Swarm Optimization (PSO) Algorithm on workflows in COSCO:





8.1 Observations

- PSO yields lower energy consumption than GA.
- GA yields lower makespan than PSO.