National College of Ireland

Improving Kubernetes Container Scheduling using Ant Colony Optimization

MSc Research Project
Cloud Computing

Shashwat Shekhar
Student ID: x17101506

School of Computing
National College of Ireland

Supervisor: Divyaa Manimaran Elango

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Shashwat Shekhar
Student ID:	x17101506
Programme:	Cloud Computing
Year:	2018
Module:	MSc Research Project
Supervisor:	Divyaa Manimaran Elango
Submission Due Date:	20/12/2018
Project Title:	Improving Kubernetes Container Scheduling using Ant Colony Optimization
Word Count:	6000
Page Count:	31

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:	
Date:	28th January 2019

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):	

Improving Kubernetes Container Scheduling using Ant Colony Optimization

Shashwat Shekhar
x17101506

28th January 2019

Abstract

In this paper we are looking at container scheduling algorithms which could be useful in improving the container and task scheduling for the popular container orchestration tools like Kubernetes. Container Orchestration is increasingly used at enterprise scale for automation in deployment and management of large container clusters. Orchestration typically includes activities like provisioning,communication between containers, instantiation and reconfiguration. Hence, container scheduling is an important aspect of orchestration. It has a direct impact on the application performance and an improved scheduling mechanism can enhance the application performance. We have chosen Kubernetes as the orchestration tool as it is one of the most popular orchestration tools and there have very limited studies in the field of container scheduling especially related to container scheduling. In this paper we have tried to use Ant colony Optimization, one of the popular meta-heuristic scheduling algorithms for container and task scheduling, measuring its impact on performance. The results of our experiments are highly encouraging and Ant Colony Optimization has shown an average improvement of 20 percent over traditional scheduling algorithms.

Contents

1 Introduction

The term "cloud" has been associated with computing since the computing infrastructure and applications became elastic, scalable, location independent and delivery of these services were on demand as per consumers need. The key enabling technology for cloud computing was virtualisation. Virtualisation enabled the infrastructure and applications resources to be used my multiple consumers at once enabling scalability, elasticity and on demand delivery. Virtualisation of hardware infrastructure up till now was mostly done by Virtual machines. However,the onset of containers has revolutionized the Infrastructure virtualisation. Containers are usually known as light weight virtual machines. Containers these days are widely adopted and are being used at enterprise scale for deployment of applications.

This Section presents the motivations for studying Containers and its orchestration, it briefly explains the goal of this thesis, the limitations, and targeted audiences. The last section will gives you an overview of this thesis structure

1.1 Inspiration

In a very short span containers have completely revolutionized the way applications are build,deployed and maintained. With the omnipresence of Docker the quite a few corporations are using Docker containers to deploy applications on container platforms. These container platforms help package applications so that a specific set of resources are easily accessed by these containers hosted either on a virtual or a physical hardware. The micro services architecture which break up the application further are best suited for containers. Now, Scalability and management are an operational challenge for containers. The management of enterprise wide deployment of thousands of containers having multiple services becomes a big problem. Hence, the container orchestration tools like Docker Swarm, Kubernetes and Apache Mesos Marathon are being used increasingly to manage the containers.

Container Orchestration is mainly managing the life cycle of containers in large dynamic environments. container orchestration tools like Kubernetes and Docker Swarm help in deployment and maintenance of large number of containers which in combination form an application. Kubernetes is an open source orchestration tool that has been developed by Google and is being used comprehensively for managing and maintenance of docker clusters. Hence, this way of deploying containers which could be easily managed by a single source was of great interest and we wanted to know that how this open source tool schedules and scales containers while its deployment.

1.2 Research Objective

Since Kubernetes is a widely used container orchestration tool we will be looking at the scheduling of this orchestration tool. Scheduling of containers is very important aspect of these orchestration tool as it determines that how effectively and efficiently the containers are scheduled on the machines which could have a huge impact on the application performance and the overall performance of the system. Container scheduling becomes more important in cases where the resource distribution is non uniform in the cluster. Some similar experiments done by (Kaewkasi and Chuenmuneewong; 2017) over Docker Swarm which is an in house docker container orchestration tool has shown promising results upon changing the scheduling for docker swarm. Hence its brings to our research question -

Can container scheduling throughput of Kubernetes be improved via meta heuristic algorithm like Ant Colony Optimization?

1.3 Target Audiences

This work is mainly targeted towards computer science students and cloud enthusiasts who have keen interest in cloud technologies specially related to containers and its orchestration. It may also interest the Kubernetes orchestration tool administrators and the people working on devops.

1.4 Structure and Outline

The rest of the thesis has been organized into five section which are described as below.

Section 2 of this paper describes the related work on this topic. It explains briefly Ant colony Optimization and Kubernetes. Section 2.2 Describes Kubernetes and its need in todays world. Section 2.3 is the literature review of other related work done on cloud, containers and Virtual machines using ant colony optimization. Section 2.4 compares Ant Colony optimization with other scheduling meta-heuristic algorithms and how it is best suited in this scenario. The last section in this compared Ant Colony optimization with existing scheduling mechanism used by Kubernetes.

Section 3 of this paper details about the setup and the lab environment used for accomplishing this work.

Section 4 of this paper provide overview on the implementation of this work and its intricate details.

Section 5 of this paper gives some results of the experiments that were carried out for this research work.

Section 6 is the final section which provides the conclusion and future work related to this research project.

2 Background And Related Work

In this section of report we will taking a background look at ant colony optimization, what it is and few examples that used ant colony optimization in its preliminary phase. We will also looking at kubernetes and its need and briefly discuss on its architecture and functioning. We will be reviewing the implementation of Ant colony optimization usage in cloud computing and how other researches have used Ant Colony Optimization in their research work. This will be followed by comparison of Ant colony optimization with other scheduling algorithms and finally with kubernetes scheduling mechanism.

2.1 Ant Colony Optimization Explained

As per (Dorigo and Sttzle; 2010) Ant colony optimization follows a meta-heuristic approach to solve hard optimization problems which are combinatorial in nature. The inspiration of this algorithm came from ant behavior in finding their food. The ants leave a pheromone trail which are used a communication medium for the following ants which pick up these trails to find the path to food followed by previous ants.

In ACO, a number of artificial ants build solutions to an optimization problem and exchange information on their quality via a communication scheme that is reminiscent of the one adopted by real ants(Dorigo and Sttzle; 2010).

2.1.1 History

The behavior of Ants leaving a trail of chemicals called pheromones was noticed by a French entomologist named *Pierre-Paul Grasse*. He observed that these trails or secretions by ants can act as a significant stimuli for the ants producing them as well as the ants that follow them. He coined the term *stigmergy* for this type of communication. There are a few main differentiators of *stigmergy* which help in identifying it from other form of communication. It is essentially an indirect form of communication done by modifying the environmental conditions around them. Secondly, its local in nature and can be accessed by only a type of insects or species that are nearby (Dorigo and Birattari; 2011).

The early examples of this algorithm is Ant Systems (Dorigo et al.; 1996) which was proposed and applied on the Travel Salesman Problem (Applegate et al.; 2006) with encouraging initial results, However the Ant System was not able to compete with other state of art algorithms related to Travel Salesman Problem.

However, it did lay foundation for further research on Ant colony Optimization and quite a few research work was carried out on this which resulted in better computational performance.As of now, a considerable application of Ant Colony Optimization exist where performance obtained is nothing short of word class(Dorigo and Sttzle; 2010).

Below is the list of Successful Ant Colony Optimization Implementations

Table 1: Successful Implementation of ACO

Algorithm	Authors	Year	References
Ant Systems	Dorigo ET AL	1991	(Dorigo et al.; 1996)
Elitist AS	Dorigo ET AL	1992	(Dorigo et al.; 1996)
ANT -Q	GAMBARDELLA & DORIGO	1995	(Gambardella and Dorigo; 1995)
Ant Colony System	GAMBARDELLA & DORIGO	1996	(Gambardella and Dorigo; 1996)
Max-Min AS	STUTZLE & HOOS	1996	(Stützle and Hoos; 1996)
Rank-Based AS	BULLNHEIMER ET AL.	1997	(Bullnheimer et al.; 1997)
ANTS	MANIEZZO	1999	(Maniezzo; 1999)
BWAS	CORDON ET AL.	2000	(Cordon et al.; 2000)
Hyper-Cube AS	BLUM ET AL.	2001	(Blum et al.; 2001)

2.1.2 Applications of Ant Colony Optimization

The advancements in Ant Colony Optimization has lead to a tremendous increase in interest for this algorithm by the scientific community. There has been variety of successful applications of Ant Colony optimization and some of the popular ones are discussed in the below sections.

Telecommunication Networks
Ant colony Optimization algorithms have been found effective in network routing problems and hence are used in telecommunications where there are large number of nodes and their availability overtime could be an issue . The initial applications of Ant Colony optimization for routing in circuit switched networks and packet switched networks were first studied by Schoonderwoerd et al. and Di Caro and Dorigo respectively. After the successful proof of concept performed by these gentlemen telecommunication routing algorithms were inspired by Ant Colony Optimization algorithm so much so that they were considered state of art for wired networks(Dorigo and Birattari; 2011). AntNet introduce by Di Caro and Dorigo is an algorithm which is well known as has outperformed other algorithms in routing applications. It has be tried and tested and the results have shown this to be highly adaptive and robust algorithm(Dorigo and Birattari; 2011).

Industrial Problems
The successful implementation of ACO in academic problems has prompted many of the leading organization to look at Ant Colony Optimization in solving the real world problems. Some of the major organizations which have used Ant Colony optimization to solve their problems are EuroBios which is using this for capacity constraints, maintenance calendars and resource compatibilities.Another major organization AntOptima (www.antoptima.com) uses this algorithm for optimization of vehicle routing. Some successful industrial products based on this algorithm is DYVOIL which is mainly used in management and optimization of fleet of trucks used for heating oil distribution. ANTROUTE another product based on this algorithm is used for routing of vehicles quite a few major corporations(Dorigo and Birattari; 2011). Gravel, Price and Gagne in their work explained in (Gravel et al.; 2002) have applied this algorithm in an aluminum casting center. Another real world industrial application was implemented by Bautista and Pereira to solve assembly line balancing problem explained in their work (Bautista and Pereira; 2002).

Dynamic Optimization Problems
The basic characteristic of dynamic problem is that the search criteria changes with the time duration. This may lead to continuous changes in search conditions problem definition and ultimately the quality of solutions. In these scenarios it is imperative that the algorithm used to solve this should be able to adjust to the continuous changes in the problem state(Dorigo and Birattari; 2011). Some of the pragmatic implementations of this algorithms in dynamic problem is in the telecommunications sector where the routing has to highly dynamic to efficient. Ant Colony system a variant of Ant Colony optimization is also used in vehicle routing solutions as these solutions are extremely random in nature (Montemanni et al.; 2005).

2.1.3 Recent Advancements

As per Dorigo and Birattari the research work on Ant Colony optimization will be much more focused on optimization problems which are stochastic, dynamic and will have multiple objectives. The recent trends in Ant Colony Optimization can be classified into three categories which are mentioned below -

- Applying ACO to non-standard problems.

- Development of hybridized Ant Colony Optimization.
- Parallel implementations of Ant Colony Optimization algorithms.

Application of Ant Colony Optimization to non-standard problems include *Multi-objective Optimization* in which evaluation of solutions are done on multiple and often conflicting objectives. The other examples of non-standard problems are *Dynamic NP-Hard Problems*, *Stochastic Optimization* and *Continuous optimization*. Examples of applications of ACO to NP Hard problem are in the area of network routing and vehicle routing solutions. Stochastic problems are characterized uncertainty in data due to noise, approximation and other external factors (Dorigo and Sttzle; 2010).

Researches initially focused on developing variants of ACO by modifying the pheromone variables that was intended on improving the algorithmic performance.Later the researched moved on exploring ACO with combinations of other algorithmic techniques. Some of the noteworthy developments include *Hybridization of ACO with other metaheuristics*,*Hybridizing ACO with branch-and-bound techniques* and *Combining ACO with constraint programming techniques*(Dorigo and Sttzle; 2010).

The basic nature of ACO is what leads to parallelism in this algorithm. As perDorigo and Sttzle parallelization strategies can be classified into *fine-grained* and *coarse-grained* strategies.Parallel version of Ant Systems have been used for Travel Salesman Problem are categorized as fine-grained parallelism scheme. Quite a few eminent researchers like Bullnheimer et al. and Manfrin et al. have recommended that coarse grained parallelism is extremely promising for Ant colony Optimization.

2.2 Kubernetes And Its Need

In this section will be introducing you to Kubernetes its origin, need and its industry usage.

2.2.1 History

Kubernetes was developed by Google, it has been well known that Google had been using containers much before its commercialization by Docker and Google has to create mechanism for orchestration and scheduling to handle system properties like isolation, load balancing and placement. Google had an in-house solution named *Borg* cluster management system which it has elaborated by Burns et al. in his work.When Docker had its initial release in March 2013, Google packaged and published its most useful pieces of *Borg Cluster Management System* using open source following which Kubernetes was born(Rensin; 2015).

2.2.2 Containers Explained

Containers are form of Lightweight visualization at the operating system level. Unlike the virtual machines which share the underlying hardware containers share the host operating system kernels. Studies conducted by researchers such as (Felter et al.; 2015), suggest that containers have better performance when compared to virtual machines.

Figure 1 shows the fundamental architectural difference between containers and Virtual machines.

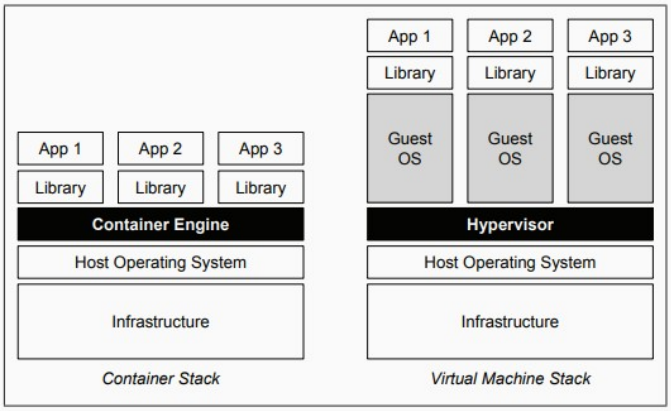

Figure 1: Virtual Machine And Container

Containers are isolated with each other with the help container engine or container management system which sits over the top host operating system. docker is a classic example of a container engine.

2.2.3 Need for Kubernetes

Containers being lightweight are very flexible in nature.However, this very property leads them to be fragile and short lived. Hence, to handle the such fragile and lightweight components we need a robust system that can handle system failures.Kubernetes brings this robustness to the container cluster by keeping an eye on the individual nodes of the cluster and replacing them with a healthy node even on hint of failure, it also helps to scale up the clusters as per the need. This helps in maintaining the system in healthy states and its Key performance Indexes such as Availability and Performance(Rensin; 2015).Below image shows a basic layout for Kubernetes Figure 2 -

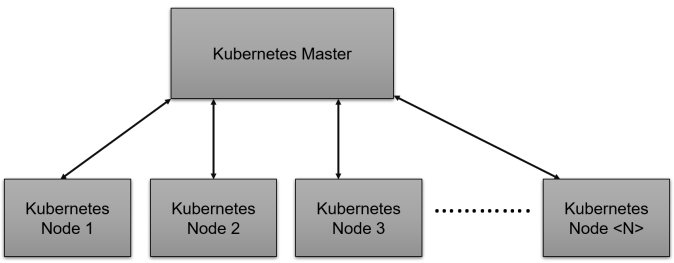

Figure 2: Basic Layout for Kubernetes

2.3 Literature Review

In this section we will taking a look at container scheduling and all the related work done on container scheduling. We will be also having a look at the meta heuristic algorithm Ant colony Optimization and its prominent uses in the scheduling problems.

We will be first having a look at the phenomenon of container scheduling.

2.3.1 Container Scheduling

Container orchestration systems have recently emerged as the way to manage the container clusters. Container orchestration tools are nothing but a cluster management system for clusters which enables the administrators to have a birds eye view of their container cluster and enables rapid deployment of the application in cluster. It helps administrators to manage the whole cluster from a single interface and speeds up activities deployment activities. There are quite a few tools in market for container orchestration, some of the leading tools for container orchestration are Docker Swarm. Google Kubernetes and Apache Mesos. Docker Swarm is the propriety toll for Docker used in scheduling dockers container farm. Google Kubernetes and Apache Mesos are open source and also widely used.

Scheduling algorithms used in these orchestration tools can have different purposes. Scheduling algorithms can be targeted at improving the resource utilization of the cluster and some other algorithms can be used to improve or maximize the application performance by scheduling the requests on the desired containers or virtual machines. The main architectures used by schedulers are explained in the paper published by -Google (Schwarzkopf et al.; 2013). This paper presented three main scheduler architectures for cluster schedulers to achieve their multiple goals like efficient usage of cluster, scheduling and management of cluster pods with user provided constraints and scheduling the requests as well as resources with degree of fairness being the prime goals. The three different architectures are Monolithic scheduling, Two level scheduling and Shared State Scheduling. A schematic overview of these scheduling algorithm are show in the Figure 3

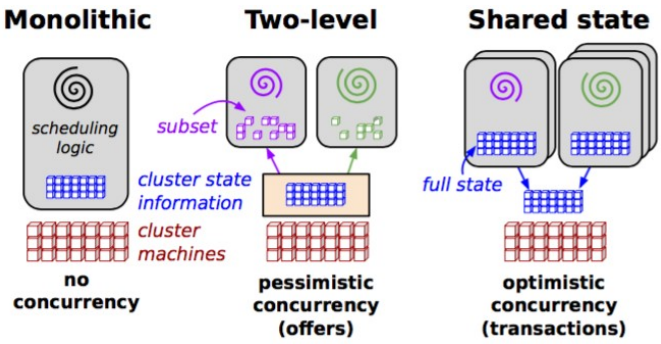

Figure 3: Scheduling Architectures prosed by Google in (Schwarzkopf et al.; 2013)

2.3.2 Comparing scheduling of Docker Swarm, Kubernetes and Apache Mesos

Grillet has compared the container scheduling strategies of the most container orchestration tools like Docker Swarm, Kubernetes and Apache Mesos. However, the author has done more of a qualitative comparison rather than a quantitative comparison. All these container orchestration tools have options to customize the scheduler logic. As per the author Docker Swarm is an extremely simple orchestration tool and its API can work in a similar fashion as it works on cluster and an individual machine. To perform his qualitative analysis on these orchestration tool author has used a few applications to test them. Firstly, he has used the sample project used in Docker tutorial for beginners which is nothing but a Food truck application and a voting application which was deployed on Amazon Web Services. The application were made to run inside containers which were placed different virtual machines and are scalable in nature. His experiments led to

conclusion that docker swarm has a simple and comprehensive scheduler which provides filters and strategies, however as per the author it lacks certain features to detect node failures. However, Since Docker Swarm being the in house orchestration tool it fully compliant with Docker APIs which is a huge advantage. As per the author the concept user by Kubernetes is pods which is completely different from Docker and Apache Mesos.

Generic Comparison of Docker Swarm and Kubernetes
Docker Swarm and Kubernetes are the most popular container orchestration tools and hence we do a generic comparison of the capabilities of these two tools in the below table -

Table 2: Kubernetes vs Docker Swarm

Features	Kubernetes	Docker Swarm
Scalability	Highly Scalable service that can scale with the requirements.	Very high scalability, up to 5 times more scalable than Kubernetes
Load Balancing	Manual load balancing often required	Capability to perform automatic between containers in same cluster
Rollbacks	Automatic rollback with ability to deploy rolling updates	Automatic rollback only available in Docker 17.04 and higher
Optimization Target	Optimized for one single large cluster	Optimized for multiple smaller clusters
Networking	Overlay network is used in communication across multiple pods	Docker Daemons is connected by overlay network and the overlay network driver is used
Ava liability	High availability and health checks performed directly on pods	High availabilty, containers are restarted on a new host if a host failure is encountered

2.3.3 Ant Colony Optimization and Scheduling Problems

There are quite a few work done on Ant colony to be used to solve scheduling problems. However, many of these are used for solving scheduling problems of virtual machines and task scheduling in cloud. We will taking a look at scheduling problems and how Ant colony has been used to solve this and provide a better solution.

The author Zhang et al. in his paper proposed an ant Colony optimization technique for optimizing Job Shop Scheduling Problem commonly known as JSP. The application of Ant colony optimization to Job Shop scheduling Problem showed positive results and the author concluded that this meta heuristic algorithm was an effective way to solve the scheduling problem related to Job Shop Scheduling.

The author Goyal and Singh implemented Ant colony Optimization in an adaptive load balancing algorithm for grid computing environment. In this experiment the Ant colony optimization was compared with other algorithms using the Grid Sim simulation tool. With this experiment they proved that using Ant colony optimization for

solving load balancing resulted in better resource utilization when compared with other algorithms and the standard deviation between them was found to be between 0.71-0.77. The work done by these authors focused on resource utilization and was performed using simulation.The experiments proposed by us focus on the Application performance and response time taken by tasks scheduled on containers. Also, the author has used virtual machines in his experiment.

In other experiment performed by the author Tawfeek et al. implemented Ant colony based task scheduling algorithm. This algorithm was then compared with FCFS which is First-Come-First-Serve and round Robin algorithm. This simulation was carried on cloud Sim and the results of Ant Colony Optimization was better compared to the other algorithms in the aspects such as degree of imbalance in cluster and the average response time of tasks on cluster. Their work was focused on improving the cluster resource utilization by improving resource imbalance and reducing make-span. This is in contrast to the experiments proposed by us which focuses on better distribution of tasks on containers by scheduling them on a the cluster. The distribution of tasks will be heuristic in nature and should help in improving the overall application performance.

In paper Gao et al. author developed a multi objective algorithm based on Ant colony for the efficient placement of Virtual Machines in the cluster. Their work was focused on solving the placement problem related to Virtual Machine in the cluster and the solution proposed by them reduced the time frame of Virtual machine placement to a few minutes in the cluster with having a large set of virtual machine. This helped in improving the overall Virtual Machine allocation time improving the scalability of cluster.

The experiments done by Kaewkasi and Chuenmuneewong uses a variant of Ant colony Optimization called ant systems for scheduling of containers on a cluster for improved utilization. The author carried out experiments on docker Swam and is successful in showing that Ant colony Optimization can improve the scheduling of containers on the cluster and its proper placement. It concluded that using Ant Colony Optimization there could be 15 improvement in container scheduling time when compared with the default Docker Swarm scheduling algorithm. However, this experiment was carried on docker Swarm and focused on container placement time, whereas our experiments are focused on Kubenetes and scheduling of tasks on its cluster with a focus on over application performance in the cluster.

Below table shows the summary of the related works done on scheduling using Ant Colony Optimization -

Table 3: Summary of Related Work

Author and Year	Related Work
(Zhang et al.; 2006)	Implementation of an Ant Colony Optimization technique for job shop scheduling problem
(Goyal and Singh; 2012)	Adaptive and Dynamic Load Balancing in Grid Using Ant Colony Optimization
(Tawfeek et al.; 2013)	Cloud task scheduling based on ant colony optimization
(Gao et al.; 2013)	A multi-objective ant colony system algorithm for virtual machine placement in cloud computing
(Kaewkasi and Chuenmuneewong; 2017)	Improvement of Container Scheduling for Docker using Ant Colony Optimization

3 Methodology

In this of the parer we will taking a look at the methodology followed for our experiments. The experiment is carried out in two parts. the first part consists of deployment of a application on Kubernetes cluster and testing its response time. The focus here is to check the response time for the application as its the single most important factor which ultimately leads to user satisfaction. This paper aims at finding that if Ant Colony Optimization is algorithm that is feasible to be used in container scheduling which we have tried to prove using simulation in cloud Sim. Secondly, we are also measuring the response time for a task completion once Ant colony Optimization is applied on container scheduling.

3.1 Kubernetes on Google Cloud Platform (GCP)

Google has a internal Kubernetes Engine called Google Kubernetes Engine (GKE) which it mainly uses for orchestration of Docker container or container clusters. Google Kubernetes Engine is mainly run on Google Compute Engine instances. The Kubernetes clusters are run on these instances. The Google Kubernetes Engine consists of a master node to manage th cluster, the interaction of cluster is done through API server which performs tasks such as service of API requests and scheduling of containers. Apart from these elements a kubernetes cluster also includes one or more nodes which may be running a Docker container and a kubelet agent to manage the cluster. Figure 4 shows the overall architecture of Kubernetes.

The Kubernetes engine organizes containers into pods which is nothing but a logical representation of a container group which are related to each other. Typically Kubernetes engine is used for activities like creation and resize of containers, replication controllers, up-gradation and updation of the cluster. Gcloud command line interface is used to interact with Google Kubernetes Engine*What is Google Kubernetes Engine (GKE)? - Definition from WhatIs.com*.

The processes followed to deploy the application in Google Kubernetes Engine is showed in Figure 5. The application deployed is small application with Go Programming

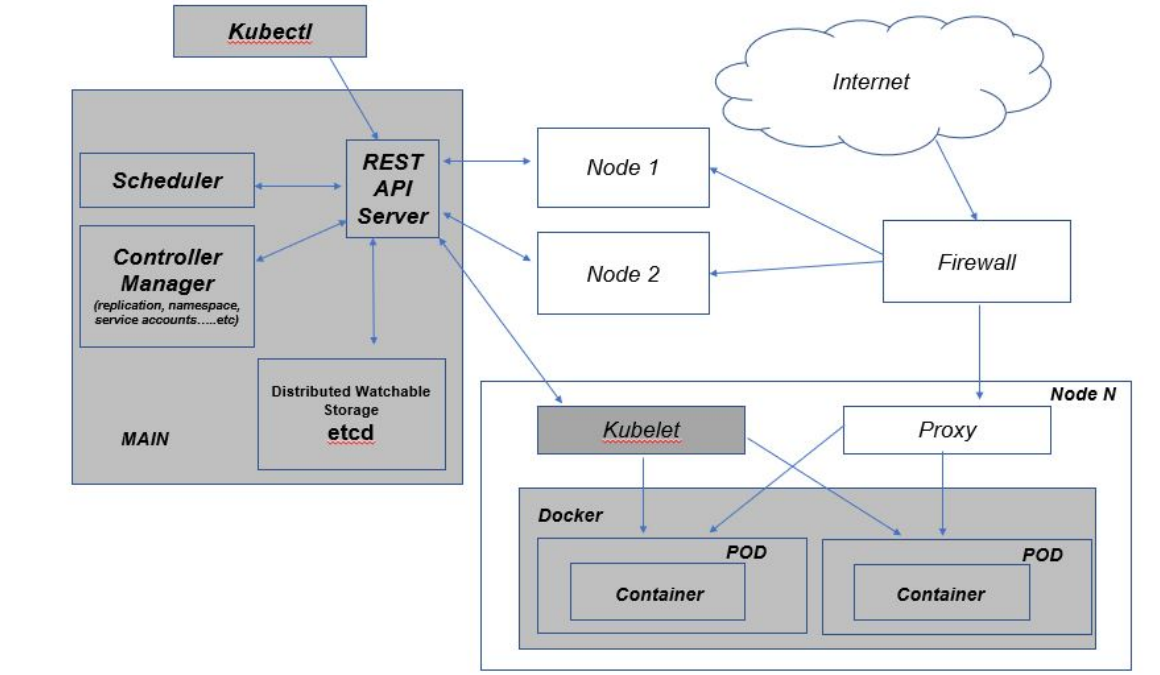

Figure 4: Kubernetes Architecture (Netto et al.; 2017)

over the Docker Container. the image of that container was ten placed on Google Container Registry and was download on the container cluster.The application is running a Go Web server and is accessible externally using external IP address of the cluster.

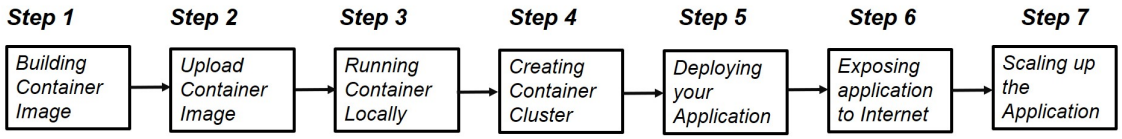

Figure 5: Kubernetes Application Deployment Process

The above process has been followed for deploying the application on the kubernetes cluster.

3.2 Cloud Sim Ant Colony Container Scheduling Process

Cloud Sim is a new generalized and extensible framework used fro simulating, modeling and experimenting on the cloud computing infrastructure and services. This framework was developed by Rajkumar Buyya and team to allow the researcher community to simulate the cloud infrastructure entities and extend the framework as per their need. His work is explained in the paper Buyya et al. in which he explains all the aspects of Cloud Sim. There are quite a few features provided by cloud Sim, some of the popular features are that it supports modeling and simulation of cloud Computing Infrastructure which cloud be on a large scale, it also provides self contained platform for data center modeling and testing of scheduling and allocation policies. Recently, Cloud Sim 4.0 was introduced which has features and extensions for simulations on containers. Below Figure 6 that shows cloud Sim setup for our experiment. The process followed to create

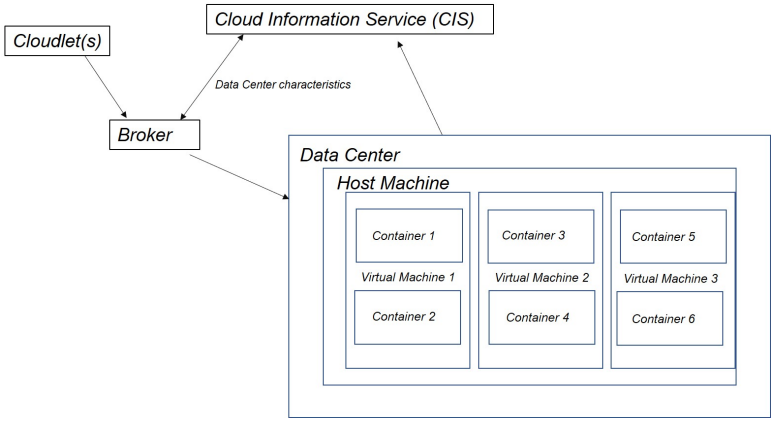

Figure 6: Cloud Sim Environment Setup

the containers and run the tasks or the user requests in form of cloudlets is shown in the below Figure 7.

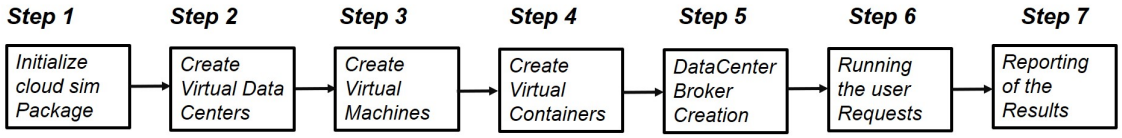

Figure 7: Cloud Sim Container Creation Process

4 Design Specification

Figure 8 depicts the flow followed by using Ant Colony Optimization for scheduling the containers in order to a improve the application performance. The start of Ant colony Optimization takes place by initializing the ACO parameters followed by construction the solution. Solution construction is followed by local Pheromones update. All the ants visit all the nodes to find the optimal node for scheduling.

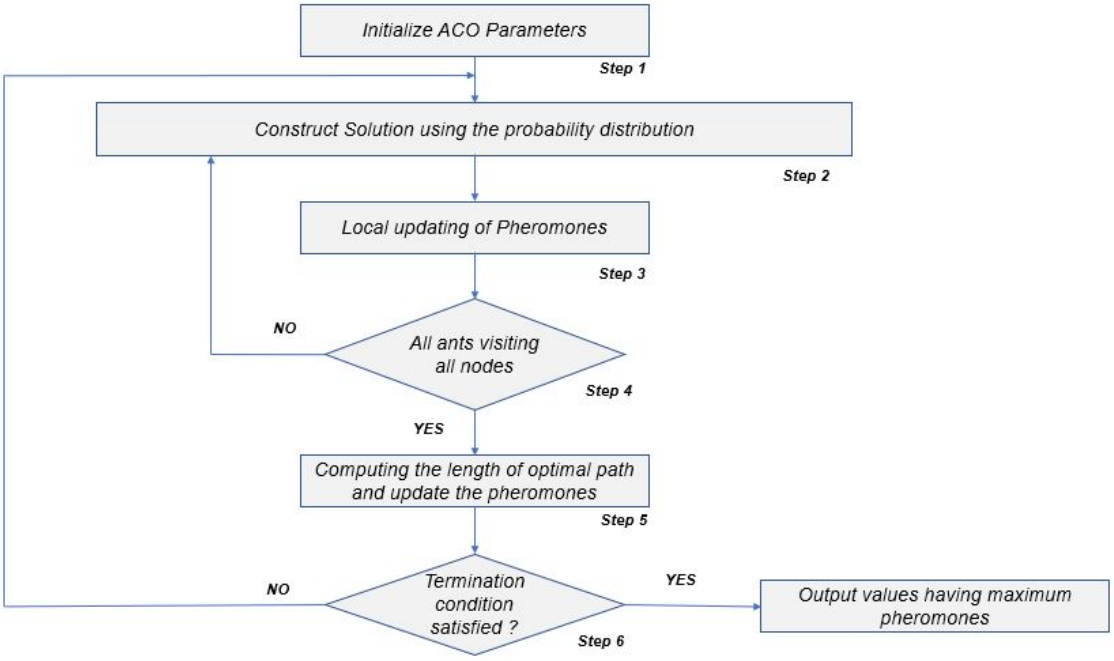

Figure 8: Flow Chart for Ant Colony Optimization

Below tables show the specifications of the Virtual machines on Cloud Sim as well as Google Kubernetes Engine.

Table 4: Configuration Table

	Number of Instances	CPU	Total Memory
Cloud SIM Data Center	3	1 Each	11.25 GB
Google Kubernetes Instances	3	1 Each	11 GB

5 Implementation

In this Section we will looking at the details of implementation of the project and its details.

5.1 Implementation on Google Cloud Platform (GCP)

To test the application and its response in real time we have deployed a sample application on the Kubernetes Engine on Google Cloud. Figure 9 shows the set of commands that have been followed for a successful deployment of application Kubernetes cluster. The application can be accessed using the URL *http://35.228.39.205:80/* where port 80 is listening port. The application is further scaled up to 3 nodes for testing the impact of response time after scaling up the solution(*Deploying a containerized web application | Kubernetes Engine Tutorials*; n.d.).

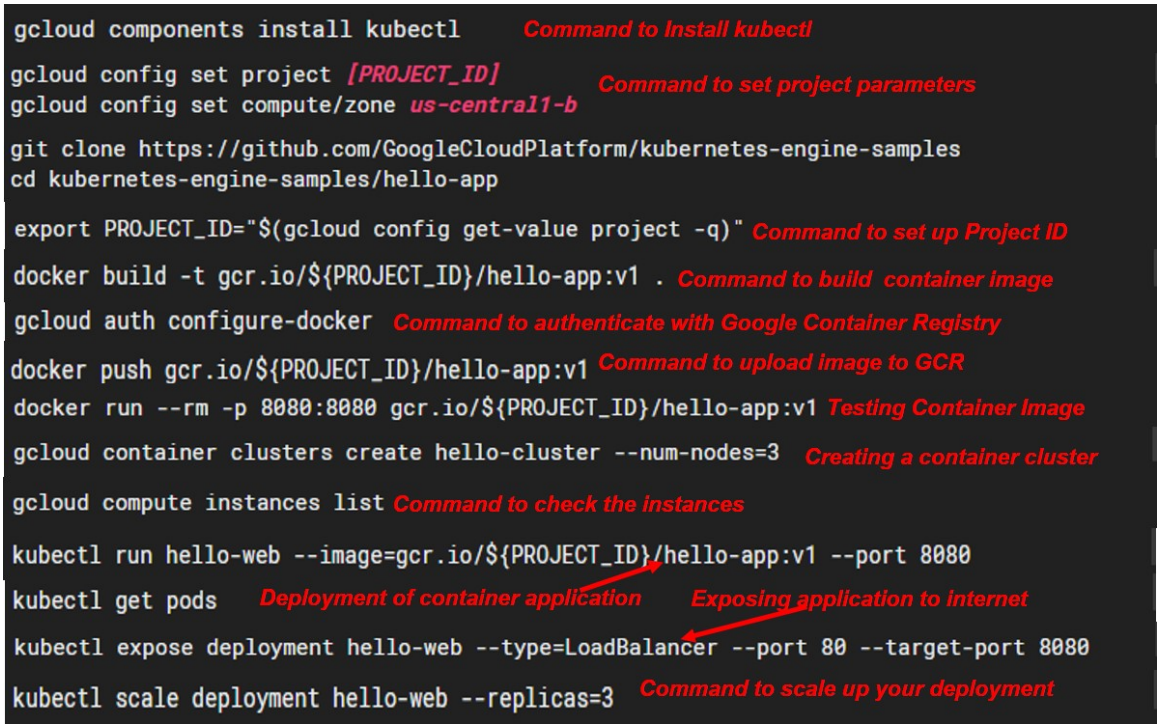

Figure 9: Commands to setup a Kubernetes Cluster

5.2 Implementation on Cloud Sim

In this section we will be taking a look at the actual implementation of the Ant colony Optimization in Cloud Sim. We will looking at the classes and the initialization of ant colony parameters that was used to run the simulation.

5.2.1 Class Diagram

The Figure 10 shows the class diagram for the solution that is implemented in cloud Sim for Ant Colony Optimization. The major class to look out for is the *Ant colony Optimization.java* class which has the code to run implement the Ant colony Optimization. It has functions like *initlizePheromone* and *getExecutionTime* which helps to initialize the pheromones and calculates the total execution time. The *AWTChart.java* class is used to create the out results of in form of a chart. The *AntColonyMain.java* class has ll the functions to create the datacenters, virtual machines and containers. It also invokes the constructor of *Ant Colony Optimization.java* class.

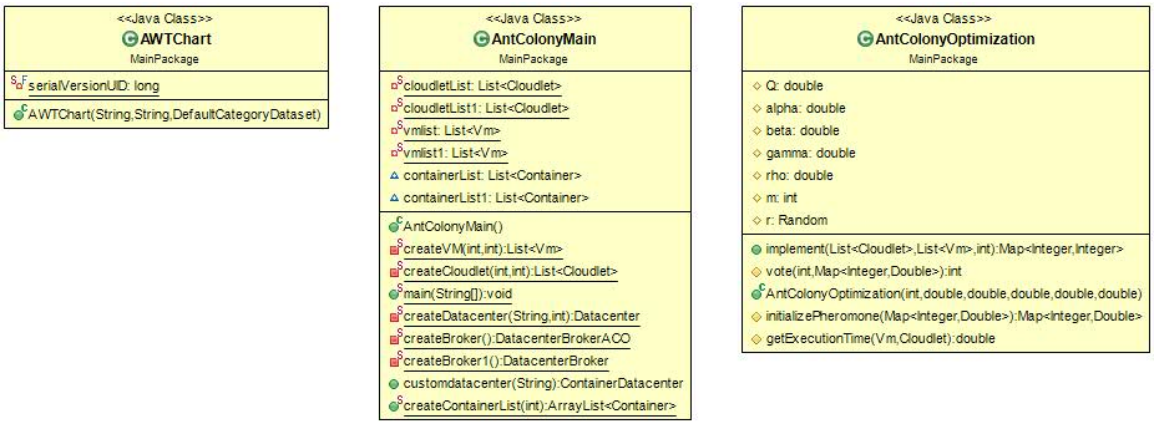

Figure 10: Class Diagram

5.2.2 Initialization of Ant Colony Optimization

The below table shows the parameters that are used in ant Colony optimization. The parameter m represents the number of ants that are required for the simulation process.We have considered 30 ants as the starting point for this simulation. The parameter *alpha* represents the importance of the pheromone trail and the initial value for this has been taken as 2. The *beta* parameter represents the ants distance priority and as per convention *beta* parameter should have a greater value compared to *alpha*. This helps to generate best results during simulation. The value *Rho* represents the evaporation variable of pheromones for every iteration, whereas Q denotes the amount of pheromone trail left by the ants.

Finally, there are some random factors which affect the simulation and helps in bringing randomness in the results.

Table 5: Ant Colony Optimization Parameters

ACO Parameter Settings for experimental Runs			
Parameter	Interpretation in Code	Notation	Values
Number of Ants	Original Frequency of Requests	m	30
Pheromone Importance	Used in calculating probability of requests	*alpha*	2
Ants Distance Priority	Used in calculating probability of requests	*beta*	1
Original Number Of Trails	Used in calculating probability of requests	*gamma*	4
Pheromone Evaporation Factor	Change in Frequency of Requests	*Rho*	0.5
Total Amount of Pheromone left on Trail	Minimum number of requests	Q	1

The Ant Colony Optimization code written by us in the Java class *ant colony Optimization* roughly follows the below algorithm while find the optimal path to schedule the containers on the Virtual machines. In this algorithm the artificial ants after the

initialization of parameters and Pheromone trails sets out to find the optimal path to place the containers and tasks on those containers.

Table 6: Ant Colony Optimization Pseudo Code

High Level Pseudo code for Ant Colony Optimization	
Step 1	Set Parameters and Initialize Pheromone Trials
Step 2	While the termination condition is not met
	Construct Ant Solutions
	Apply Local Search
	Update Pheromones
End	

This ends the implementation section of the report where we have given brief details on how to implement the solution.

6 Evaluation

In this section we will be evaluating results of the our experiments. We have done a comparison between the response time of the tasks scheduled on the cluster between the default algorithm which is round robin and our proposed algorithm which is Ant colony Optimization. Furthermore, to test the validity we compared the response time with a real time application deployment on kubernetes cluster. The kubernetes cluster deployed on Google cloud was scaled up manually to 3 nodes for testing purposes. The auto scale feature of the cluster was disabled on purpose to keep the spawning of number of nodes in check. We carried evaluation in three different experiments which were conducted in a controlled fashion. We also used ping plotter to keep an eye on the latency from the client machine to endpoint deployed on Google Cloud. Figure 11 shows the latency between the client and the end point and was obtained using the tool ping plotter. The latency between client and the endpoint was around 50 msec. The endpoint is located in Europe North region of Google Cloud.

Below are the scenarios that were considered for evaluation -

- Scalability test with 20 Nodes and 30 Ants to check the performance
- Single Node ; 3 Instances ; 30 Ants in Ant colony Parameters.
- Three Nodes ; 9 Instances ; 30 Ants in Ant colony Parameters.

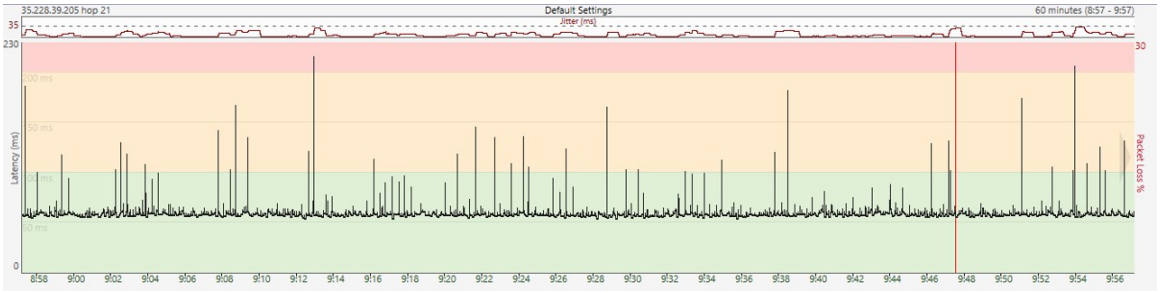

Figure 11: Latency graph to end point 35.228.39.205

6.1 Experiment / Scalability test

The below test was carried out to check the performance delivery of the Ant Colony Optimization algorithms for the clusters that are scaled up or have larger number of nodes. The results have shown that the ACO is able to perform as per the expectations. We can clearly see from the below graph that the response time for Ant Colony Optimization is below 2 seconds which is less than the Round Robin algorithm.

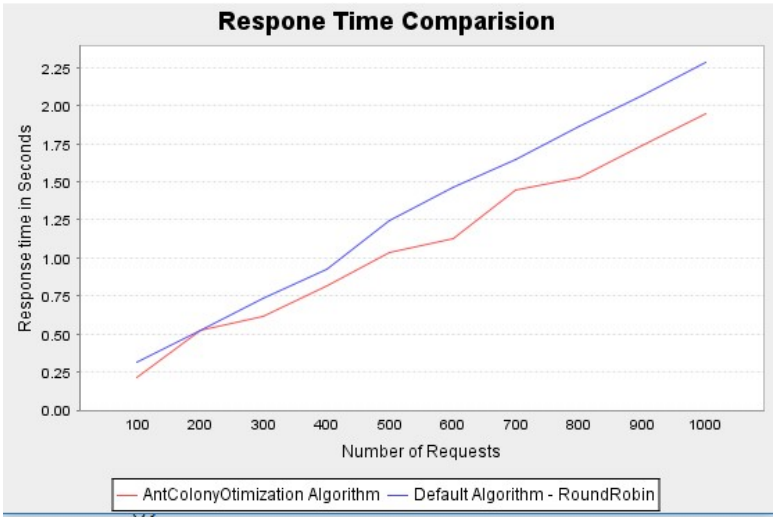

Figure 12: Response Time Under Load for Round Robin And ACO

6.2 Experiment / Single Node Experiment

The below response time graph is for the endpoint hosted on Google Cloud platform. The below graph has been generated using the performance test tool JMeter and shows the average response time for the endpoint from the user machine located in Dublin is 20 seconds.

The below response time graph is from the simulation of Cloud Sim. The below graphs represents the comparative study of the results of Round Robin and Ant Colony Optimization. It can be clearly seen that Ant colony outperforms the default algorithm.

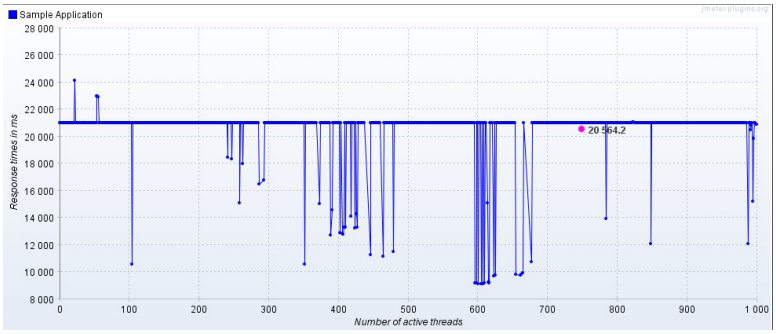

Figure 13: Response time under load graph for endpoint

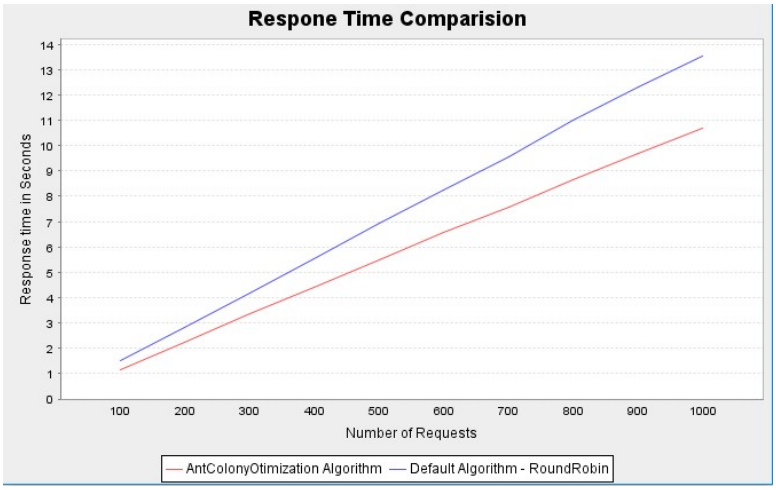

Figure 14: Response Time Under Load for Round Robin And ACO

6.3 Experiment / Three Node Experiment

Figure 15 response time graph is for the endpoint hosted on Google Cloud platform.The below graph has been generated using the performance test tool JMeter and shows the average response time for the endpoint from the user machine located in Dublin is 20 seconds.

Figure 16 response time graph is from the simulation of Cloud Sim.The below graphs

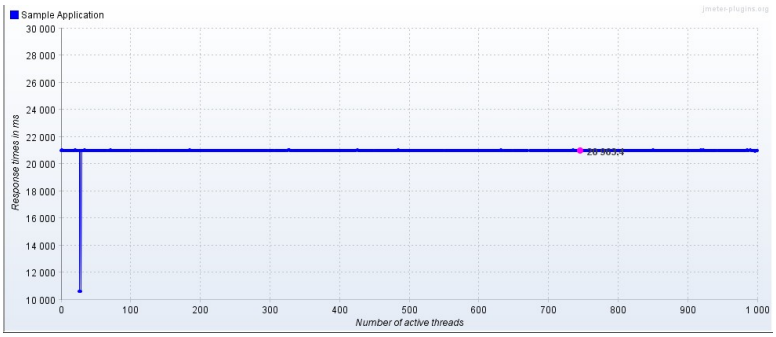

Figure 15: Response time under load graph for endpoint

represents the comparative study of the results of Round Robin and Ant Colony Optimization. It can be clearly seen that Ant colony outperforms the default algorithm.

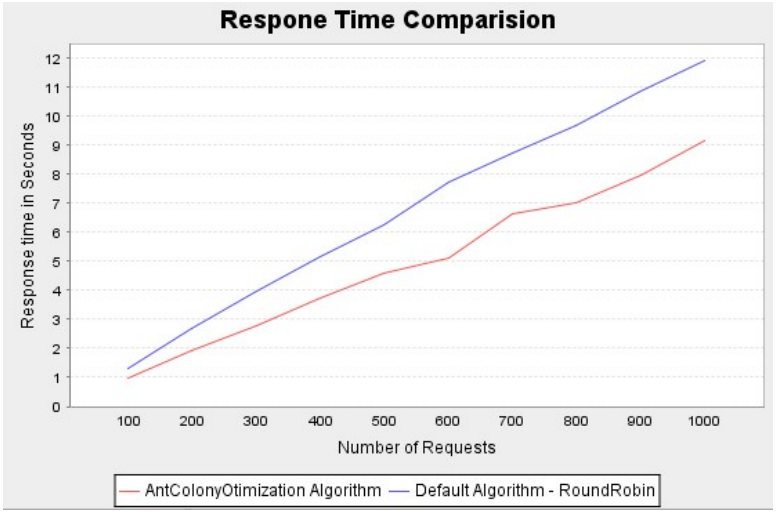

Figure 16: Response Time Under Load for Round Robin And ACO

6.4 Discussion

Table 7: Results Table

	Scalability Test (Response Time in Seconds)			Single Node Test (Response Time in Seconds)			Three Node Test (Response Time in Seconds)		
	Default algorithm	ACO	% Improvement	Default algorithm	ACO	% Improvement	Default algorithm	ACO	% Improvement
1000 Threads	2.25	1.8	20.00%	14	11	21.43%	12	9	25.00%

In the above table we have considered the peak load scenario to compare the response time of the tests. The peak load in this case is 1000 threads user load. We can clearly see that in each case Ant Colony Optimization results were better than that of the Default algorithm. The percentage improvement over the actual algorithm is 20 percent for scalability test having 20 nodes and for the other two tests it were 21 and 25 percent respectively.

In our simulation and experiments, it has been observed that the overall Response time of Ant colony based scheduling is much better compared to the traditional scheduling mechanism followed by Kubernetes. In all of our experiments the Ant colony Optimization showed better results when compared with Round Robin mechanism. The experiments also proved that Ant colony Optimization is a highly scalable solution and as the scale of the cluster increases we can expect better outputs for the requests as the

ants provide a pheromone trail of shortest path to schedule the task on Virtual Machines and containers.

7 Conclusion and Future Work

To conclude we can say that the ant colony Optimization as per the simulation results look promising and have outperformed the traditional scheduling mechanism and can be looked as a viable option to replace the default algorithm. This research successfully demonstrates the usefulness of Ant colony Optimization in case of scheduling related problems and it can also be successfully applied to containers as well.

Furthermore, the future work related to this may the implementation of this Algorithm on the actual Kubernetes source code and observing the results of the implementation. The simulation results having being compared with the actual outputs have given us some good pointers on how this algorithm may behave in real time.

Acknowledgements
I would specially like to thank my supervisor Divyaa Manimarn Elango for helping me and guiding me through the whole research and implementation process. Her consistent guidance has helped me achieve the desired results and complete my dissertation work. Finally, I would like to thank my family an friends for keeping me motivated and helping me through this duration.

References

Applegate, D. L., Bixby, R. E., Chvatal, V. and Cook, W. J. (2006). *The traveling salesman problem: a computational study*, Princeton university press.

Bautista, J. and Pereira, J. (2002). Ant algorithms for assembly line balancing, *International Workshop on Ant Algorithms*, Springer, pp. 65–75.

Blum, C., Roli, A. and Dorigo, M. (2001). Hc–aco: The hyper-cube framework for ant colony optimization, *Proceedings of MIC*, Vol. 2, pp. 399–403.

Bullnheimer, B., Hartl, R. F. and Strauss, C. (1997). A new rank based version of the ant system. a computational study.

Bullnheimer, B., Kotsis, G. and Strauß, C. (1998). Parallelization strategies for the ant system, *High Performance Algorithms and Software in Nonlinear Optimization*, Springer, pp. 87–100.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J. (2016). Borg, Omega, and Kubernetes, *Communications of the ACM* **59**(5): 50–57.
URL: *http://dl.acm.org/citation.cfm?doid=2930840.2890784*

Buyya, R., Ranjan, R. and Calheiros, R. N. (2009). Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities, *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*, IEEE, pp. 1–11.

Cordon, O., de Viana, I. F., Herrera, F. and Moreno, L. (2000). A new aco model integrating evolutionary computation concepts: The best-worst ant system.

Deploying a containerized web application | Kubernetes Engine Tutorials (n.d.).
URL: *https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app*

Di Caro, G. and Dorigo, M. (1998). Antnet: Distributed stigmergetic control for communications networks, *Journal of Artificial Intelligence Research* **9**: 317–365.

Dorigo, M. and Birattari, M. (2011). Ant colony optimization, *Encyclopedia of machine learning*, Springer, pp. 36–39.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **26**(1): 29–41.

Dorigo, M. and Sttzle, T. (2010). Ant Colony Optimization: Overview and Recent Advances, *in* M. Gendreau and J.-Y. Potvin (eds), *Handbook of Metaheuristics*, Vol. 146, Springer US, Boston, MA, pp. 227–263.
URL: *http://link.springer.com/10.1007/978-1-4419-1665-5_8*

Felter, W., Ferreira, A., Rajamony, R. and Rubio, J. (2015). An updated performance comparison of virtual machines and Linux containers, *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172.

Gambardella, L. M. and Dorigo, M. (1995). Ant-q: A reinforcement learning approach to the traveling salesman problem, *Machine Learning Proceedings 1995*, Elsevier, pp. 252–260.

Gambardella, L. M. and Dorigo, M. (1996). Solving symmetric and asymmetric tsps by ant colonies, *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, IEEE, pp. 622–627.

Gao, Y., Guan, H., Qi, Z., Hou, Y. and Liu, L. (2013). A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, *Journal of Computer and System Sciences* **79**(8): 1230–1242.

Goyal, S. K. and Singh, M. (2012). Adaptive and dynamic load balancing in grid using ant colony optimization, *International Journal of Engineering and Technology* **4**(4): 167–74.

Gravel, M., Price, W. L. and Gagné, C. (2002). Scheduling continuous casting of aluminum using a multiple objective ant colony optimization metaheuristic, *European Journal of Operational Research* **143**(1): 218–229.

Grillet, A. (2016). Comparison of Container Schedulers.
URL: *https://medium.com/@ArmandGrillet/comparison-of-container-schedulers-c427f4f7421*

Kaewkasi, C. and Chuenmuneewong, K. (2017). Improvement of container scheduling for docker using ant colony optimization, *Knowledge and Smart Technology (KST), 2017 9th International Conference on*, IEEE, pp. 254–259.

Manfrin, M., Birattari, M., Stützle, T. and Dorigo, M. (2006). Parallel ant colony optimization for the traveling salesman problem, *International Workshop on Ant Colony Optimization and Swarm Intelligence*, Springer, pp. 224–234.

Maniezzo, V. (1999). Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem, *INFORMS journal on computing* **11**(4): 358–369.

Montemanni, R., Gambardella, L. M., Rizzoli, A. E. and Donati, A. V. (2005). Ant colony system for a dynamic vehicle routing problem, *Journal of Combinatorial Optimization* **10**(4): 327–343.

Netto, H. V., Lung, L. C., Correia, M., Luiz, A. F. and S de Souza, L. M. (2017). State machine replication in containers managed by Kubernetes, *Journal of Systems Architecture* **73**: 53–59.
URL: *http://linkinghub.elsevier.com/retrieve/pii/S1383762116302752*

Rensin, D. K. (2015). *Kubernetes - Scheduling the Future at Cloud Scale*, 1005 Gravenstein Highway North Sebastopol, CA 95472.
URL: *http://www.oreilly.com/webops-perf/free/kubernetes.csp*

Schoonderwoerd, R., Holland, O. E., Bruten, J. L. and Rothkrantz, L. J. (1997). Ant-based load balancing in telecommunications networks, *Adaptive behavior* **5**(2): 169–207.

Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M. and Wilkes, J. (2013). Omega: flexible, scalable schedulers for large compute clusters, *Proceedings of the 8th ACM European Conference on Computer Systems*, ACM, pp. 351–364.

Stützle, T. and Hoos, H. H. (1996). Improving the ant system: A detailed report on the max–min ant system, *FG Intellektik, FB Informatik, TU Darmstadt, Germany, Tech. Rep. AIDA–96–12* .

Tawfeek, M. A., El-Sisi, A., Keshk, A. E. and Torkey, F. A. (2013). Cloud task scheduling based on ant colony optimization, *Computer Engineering & Systems (ICCES), 2013 8th International Conference on*, IEEE, pp. 64–69.

What is Google Kubernetes Engine (GKE)? - Definition from WhatIs.com (n.d.).
URL: *https://searchitoperations.techtarget.com/definition/Google-Container-Engine-GKE*

Zhang, J., Hu, X., Tan, X., Zhong, J. H. and Huang, Q. (2006). Implementation of an ant colony optimization technique for job shop scheduling problem, *Transactions of the Institute of Measurement and Control* **28**(1): 93–108.

8 Appendix - Configuration Manual

8.1 Setting up Google Cloud Platform

Kubernetes Cluster setup and application was deployed Google Cloud Platform using the following Steps.

Step 1 Login into Google Cloud console (URL - https://console.cloud.google.com) you will probably need a gmail ID to login into Google console

Step 2 Create new project on google cloud console as shown in the below snapshot

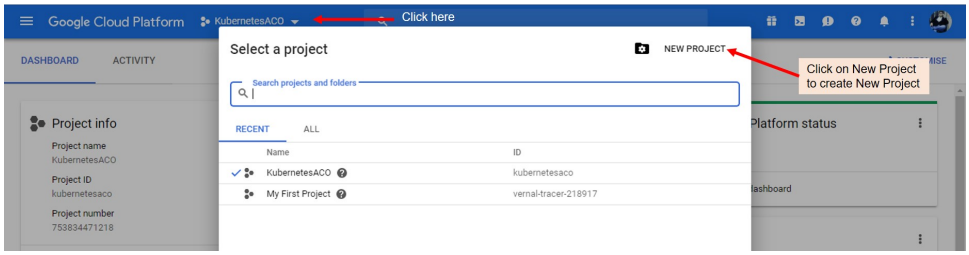

Figure 17: Google Cloud console

Step 3-Enable Billing for the selected project by going on the billing dashboard of Google Cloud Platform

Step 4-Open Google cloud shell as shown in the below snapshot

Figure 18: Google Cloud console Cloud Shell

Step 5-Install Kubectl using the gcloud command line tool.

Step 6 Create a Kubernetes cluster using Google Kubernetes Engine.

Step 7 Configure your Project ID and Compute Zone for the cluster.

Step 8 Download the source code of the application to be deployed.

Step 9 Setup the Project ID environment variable in the shell. This will be used to tag the container image when you are pushing it on Google container Registry

Step 10 Build and tag the container image for upload in container registry

Step 11- Docker Command Line tool to be configured to authenticate Container Registry.

Step 12- Upload the docker image to container registry using docker command line tool

Step 13 - Uploaded image can be locally tested using docker engine

Step 14 Run the command to create a cluster

Step 15 Check the total compute instances allocated using compute instance list command

Step 16 Deploy your application and make port 8080 as the listener port

Step 17 Check the pods created after the deployment

Step 18 Run commands to expose your application on Internet

Step 19 Check the external IP of your endpoint or the load balancer

Step 20 Finally scale up your application by running replicas.

Manifest of the Docker Container image on Google Cloud

”schemaVersion”: 2, ”mediaType”: ”application/vnd.docker.distribution.manifest.v2+json”, ”config”: ”mediaType”: ”application/vnd.docker.container.image.v1+json”, ”size”: 2007, ”digest”: ”sha256:873c1eae237c0f5e7500704ab3ea473f838ea71ae591a768603b3bcbcd27eb2a” , ”layers”: [”mediaType”: ”application/vnd.docker.image.rootfs.diff.tar.gzip”, ”size”: 2206931, ”digest”: ”sha256:4fe2ade4980c2dda4fc95858ebb981489baec8c1e4bd282ab1c3560be8ff9bde” , ”mediaType”: ”application/vnd.docker.image.rootfs.diff.tar.gzip”, ”size”: 2071432, ”digest”: ”sha256:a993e4111f10dbb0d583e0d28cdfbab86ad3064fc331abc7ca4faba91c3aedce”]

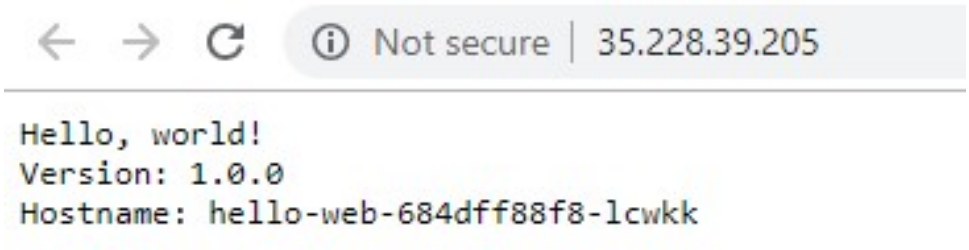

Figure 19: Output of Deployed Application

8.2 Setting up Cloud Sim

Below are the steps to setup Cloud Sim on your personal machine.

Step 1 Download the latest version of Java from the link *https://www.oracle.com/technetwork/java/downloads-2133151.html* as shown in the below figure.

Java SE Development Kit 8u191

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement | Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.97 MB	jdk-8u191-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.92 MB	jdk-8u191-linux-arm64-vfp-hflt.tar.gz
Linux x86	170.89 MB	jdk-8u191-linux-i586.rpm
Linux x86	185.69 MB	jdk-8u191-linux-i586.tar.gz
Linux x64	167.99 MB	jdk-8u191-linux-x64.rpm
Linux x64	182.87 MB	jdk-8u191-linux-x64.tar.gz
Mac OS X x64	245.92 MB	jdk-8u191-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	133.04 MB	jdk-8u191-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	94.28 MB	jdk-8u191-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	134.04 MB	jdk-8u191-solaris-x64.tar.Z
Solaris x64	92.13 MB	jdk-8u191-solaris-x64.tar.gz
Windows x86	197.34 MB	jdk-8u191-windows-i586.exe
Windows x64	207.22 MB	jdk-8u191-windows-x64.exe

Figure 20: Java Link Download

Step 2 Install latest version of Java.

Step 3- Setup Java environment variables.

Step 4 Download Eclipse Integrated Development environment from *https://www.eclipse.org/downl* Download the latest version of eclipse from this link.

Step 5 Install the latest version of eclipse on your work desktop.

Figure 21: Cloud Sim download Link

Step 6- Download the latest cloud Sim version from *https://github.com/Cloudslab/cloudsim/releases* as shown in he below figure. Cloud Sim 4.0 is downloaded as it has the latest frameworks for containers.

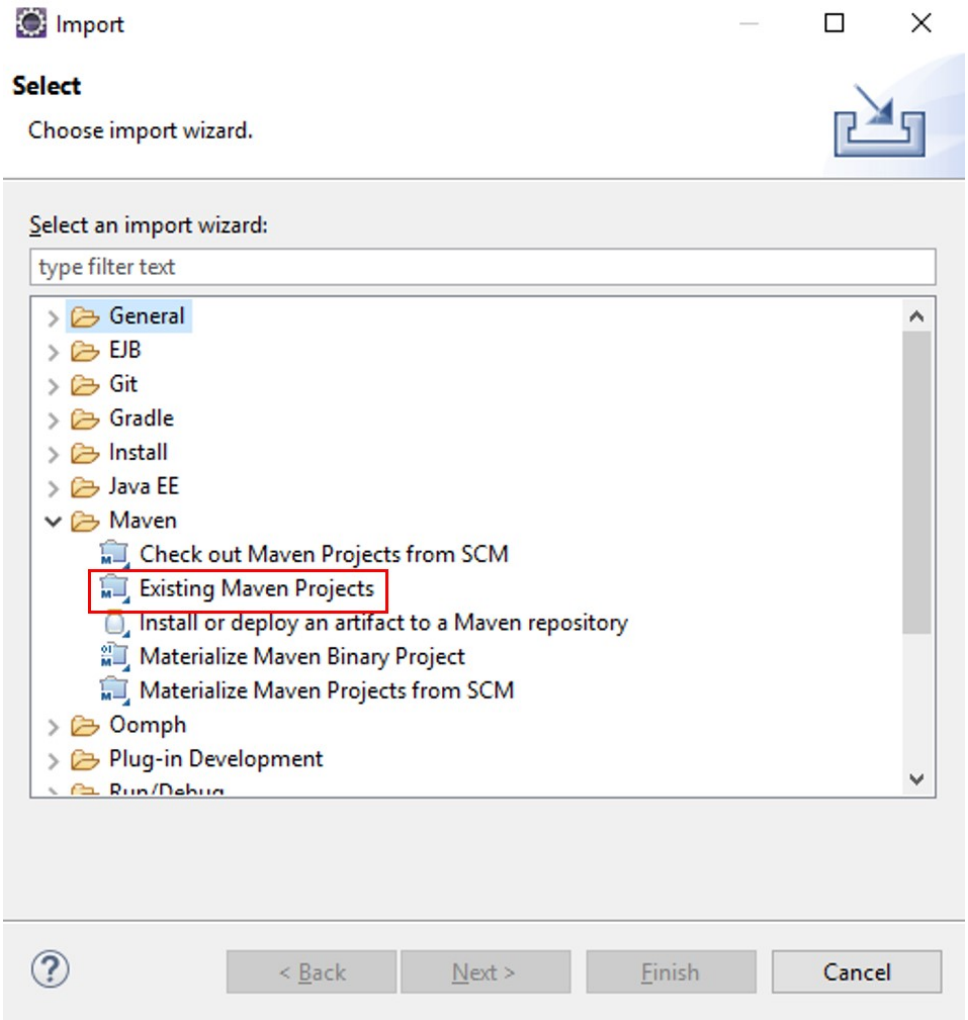

Figure 22: Importing Cloud Sim 4.0 Files using Maven

Step 7 Unzip and unpack Cloud Sim 4.0 and Import in eclipse using the Maven for dependencies as shown in the below figure.

Step 8 You will see the below tree structure if the import is successful.

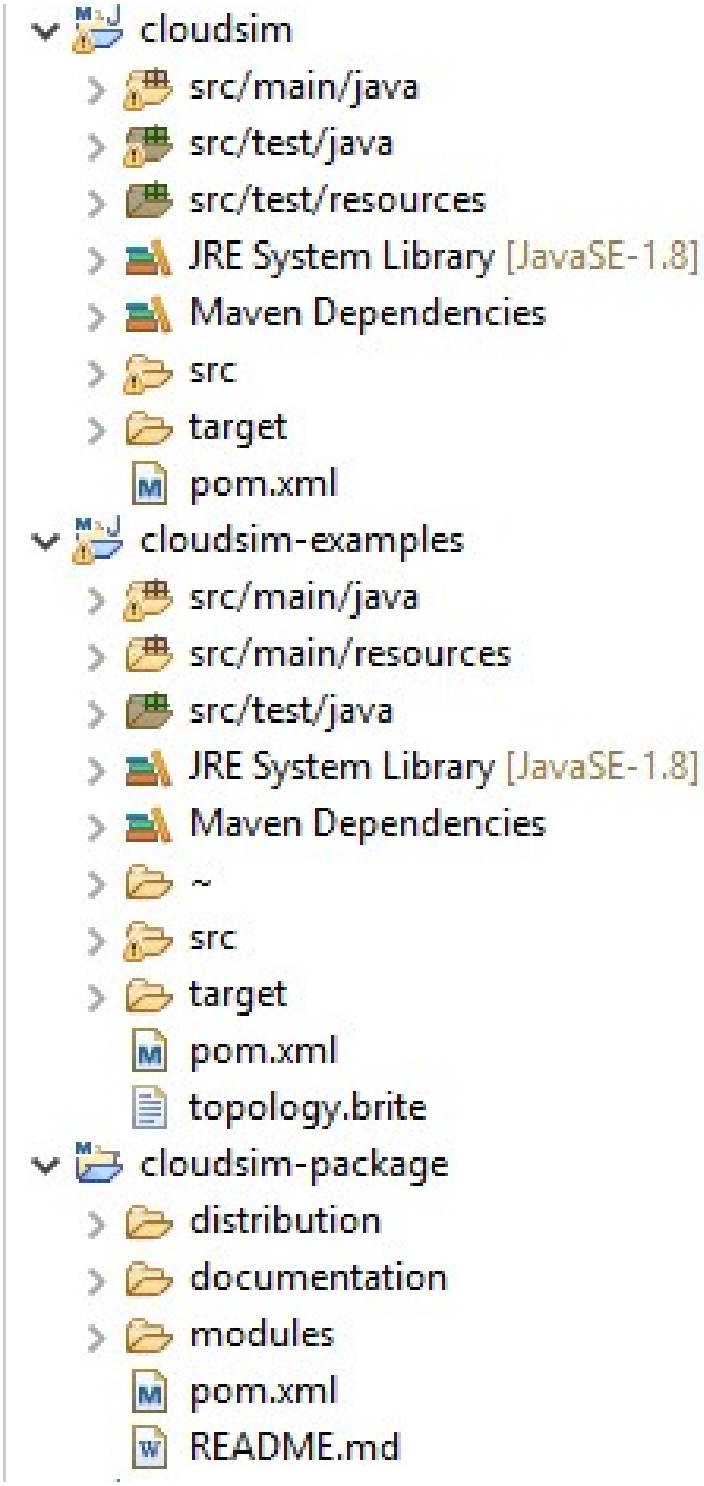

Figure 23: Cloud Sim 4.0 Tree Structure

8.3 Code And Output

The Ant Colony Optimization code is Written in Java File Ant colony Optimization.java. It has 4 functions and a Constructor. The Functions are Implement, Initialize Pheromones, get Execution Time and Vote. Snapshots of these functions are given below.

```
public Map<Integer,Integer> implement(List<ContainerCloudlet> taskList,List<ContainerVm> vmList,int tmax) throws FileNotFoundException{
    int tasks = taskList.size();
    int vms = vmList.size();
    Map<Integer,Integer> allocatedtasks = new HashMap<>();
    Map<Integer, Map<Integer,Double> > execTimes;
    Map<Integer,Double> cc, pheromones;

    execTimes = new HashMap<>();
    cc = new HashMap<>();

    for(int i=0;i<tasks;i++){
        Map<Integer,Double> x = new HashMap<>();
        for (int j=0; j<vms ; j++) {
            //double t = getExecutionTime(vmList.get(j),taskList.get(i));
            double t = getExecutionTime(vmList.get(j),taskList.get(i));
            x.put(j,t);
        }
        execTimes.put(i,x);
    }

    for(int i=0;i<vms;i++){
        ContainerVm vm = vmList.get(i);
        double Cc = vm.getNumberOfPes()*vm.getMips() + vm.getBw();
        cc.put(i,Cc);
    }
```

Figure 24: Ant colony Optimization Implement function

```
protected int vote(int vms, Map<Integer,Double> probab){
    int []freq = new int[vms];
    int sum = 0;

    for(int i=0;i<vms;i++){
        freq[i] = (int)(probab.get(i)*100000.0);
        sum += freq[i];
    }

    int n = 1 + r.nextInt(sum);
    if(n <= freq[0]){
        return 0;
    }

    for(int i=0;i<vms-1;i++){
        freq[i+1] += freq[i];
        if(n>freq[i] && n<= freq[i+1]){
            return i+1;
        }
    }
    return 0;
}
```

Figure 25: Ant Colony Optimization Vote function

```
public AntColonyOptimization(int m, double Q, double alpha, double beta, double gamma, double rho){
    this.m = m;
    this.Q = Q;
    this.alpha = alpha;
    this.beta = beta;
    this.gamma = gamma;
    this.rho = rho;
    r = new Random();
}
```

Figure 26: Ant colony Optimization Class Constructor

The Output of this is mainly in form of the graphs which are presented in Research Report, but below are some of the snaps from the Output Logs.

```
0.1: Broker: VM #0 has been created in Datacenter #2, Host #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
0.1: Broker: VM #7 has been created in Datacenter #2, Host #0
0.1: Broker: VM #8 has been created in Datacenter #2, Host #0
```

Figure 27: VM Creation in simulations

```
4.110152284263959: Broker: Cloudlet 105 received
4.110152284263959: Broker: Cloudlet 468 received
4.110152284263959: Broker: Cloudlet 678 received
4.110152284263959: Broker: Cloudlet 506 received
4.110152284263959: Broker: Cloudlet 941 received
4.110152284263959: Broker: Cloudlet 466 received
4.110152284263959: Broker: Cloudlet 914 received
4.110152284263959: Broker: Cloudlet 936 received
4.173982869379014: Broker1: Cloudlet 279 received
4.173982869379014: Broker1: Cloudlet 454 received
4.173982869379014: Broker1: Cloudlet 670 received
4.173982869379014: Broker1: Cloudlet 733 received
4.173982869379014: Broker1: Cloudlet 850 received
4.173982869379014: Broker1: Cloudlet 671 received
4.173982869379014: Broker1: Cloudlet 941 received
4.173982869379014: Broker1: Cloudlet 69 received
4.173982869379014: Broker1: Cloudlet 141 received
4.173982869379014: Broker1: Cloudlet 177 received
4.173982869379014: Broker1: Cloudlet 186 received
4.173982869379014: Broker1: Cloudlet 312 received
4.173982869379014: Broker1: Cloudlet 447 received
4.173982869379014: Broker1: Cloudlet 735 received
4.173982869379014: Broker1: Cloudlet 933 received
4.173982869379014: Broker1: Cloudlet 942 received
4.173982869379014: Broker1: Cloudlet 997 received
4.173982869379014: Broker1: Cloudlet 287 received
4.173982869379014: Broker1: Cloudlet 404 received
4.173982869379014: Broker1: Cloudlet 728 received
4.173982869379014: Broker1: Cloudlet 836 received
4.218362831858407: Broker: Cloudlet 621 received
4.218362831858407: Broker: Cloudlet 683 received
```

Figure 28: Cloudlets Received Output

```
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Datacenter_1 is shutting down...
Broker is shutting down...
Broker1 is shutting down...
Simulation completed.
```

Figure 29: Cloudlets Received Output

8.4 Miscellaneous Tools

We have also used some of the other tools for evaluation purposes. The tools used were JMeter and Ping Plotter.

JMeter is a popular open source performance testing tool developed and managed by Apache Foundation. Its been widely used for performance testing in industry. We have used this tool to evaluate response time of our endpoint deployed on Google Cloud. JMeter can be downloaded from URL *https://jmeter.apache.org/download_jmeter.cgi* as shown in the below image.

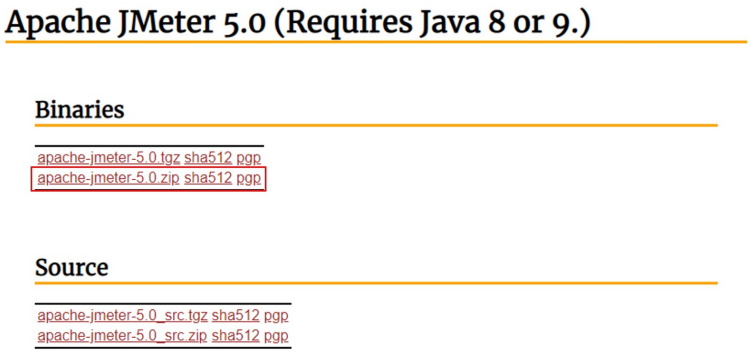

Figure 30: Apache JMeter Download Link

It can be used by extracting the zip folder and running the batch file under bin folder in JMeter folder structure. Upon opening the batch file you will see the JMeter screen as shown in below image

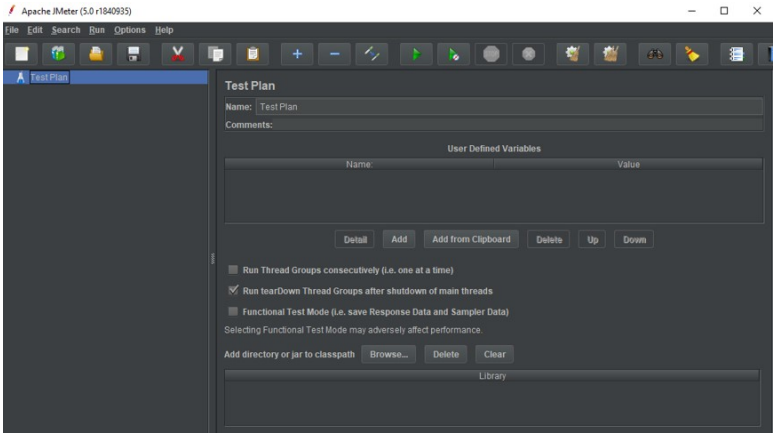

Figure 31: Apache JMeter Home Screen

The other tool which we have used in Ping Plotter. This can be downloaded from the link https://www.pingplotter.com/ and we are using the 14-day trial version of this tool for our experimental evaluation. Ping Plotter is used to track the latency and trace route over the network. It also helps in identifying any packet loss from source to destination. The installation of Ping plotter is very straightforward and once the installation is complete you can see the below screen where you can configure the IP address you want to ping.

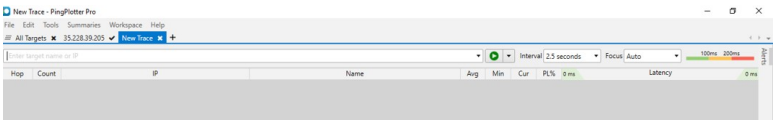

Figure 32: Ping Plotter Screen