National College of Ireland

Multi-agent based Interconnected clouds for execution of workload with deadline

MSc Research Project
Cloud Computing

Anil Judhistira Gauda
Student ID: x18180663

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Anil Judhistira Gauda
Student ID:	x18180663
Programme:	Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	17/08/2020
Project Title:	Multi-agent based Interconnected clouds for execution of workload with deadline
Word Count:	6742
Page Count:	20

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.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	
Date:	15th August 2020

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):	

Multi-agent based Interconnected clouds for execution of workload with deadline

Anil Judhistira Gauda
x18180663

Abstract

Cloud computing is well known to provide computing resources on-demand some of these resources that are most prominently used are virtual machines.Using a single cloud can led to various issues. Issues like lack of availability and scalability. This issues can be handled by using multiple cloud vendor platforms instead of one this approach in cloud computing is known as the interconnected cloud. In this paper, an agent-based system is developed to manage workload execution and distribution of workloads with a deadline on a multi-cloud environment. The developed system is capable to prioritize and send workload to the multiple cloud environment as well as collect metrics for re-execution of jobs efficiently. Developed Multi-agent system use user-preference to allocate jobs to the cloud environment, user-preference is taken as an input from the user along with jobs. Resource provisioning algorithm taking into account User-Preference has proven to give 100% satisfaction for both deadline and non-deadline based jobs assuming that sufficient virtual machines are registered with developed scheduler.

1 Introduction

Cloud computing has revolutionized the method to acquire computing resources enabling users to get resource on-demand with lesser efforts and at a much cheaper price. Considering virtual machines (VM), VM instances are provided by service providers under different labels and pricing models, for example, AWS's EC2 Instance, Microsoft Azure's virtual machines and GCP's Cloud compute engine. These virtual machines are provided by cloud service provider comes under various pricing model i.e Reserved model or on-demand model along with different price models there are also availability and scalability associated with the cloud computing resources depending upon the model that is used by a cloud service user. Due to an increase in demand of cloud computing, service providers create services that are complex but cannot collaborate with different service providers. To combine multiple cloud service providers there has been various standards proposed by International Organization for Standardization (ISO), National Institute of Standards and Technology's (NIST), Open Cloud Computing Interface (OCCI), Open Grid Forum's (OGF) and many more. And some efforts have been made from cloud users perspective by creating middlewares and brokers [1].
Interconnected cloud is a paradigm that focuses on cloud users perspective and brokers implementation to combine multiple cloud environments. Broker based approach has the biggest challenge of communication between the two or cloud environment as the services

that are provided by the cloud service providers are not according to any predefined standards. To enable coordination between the cloud environments and enable brokers to merge computing resources and micro manage it for better efficiency and scalability agent-based technology has been preferred [2].
An agent is a computational system that can perform independent actions to achieve certain design objectives [2]. When the system of more than one agents collaborate to perform a certain complex task and achieve a difficult goal it is known as a multi-agent based system. In cloud computing, multi-agent based systems are designed to perform a wide range of tasks, from knowledge gathering of cloud services to negotiation between cloud environments. Some of the previous works focus on Multi-agent based systems for scaling and efficient utilization of computing resources [3; 4].
The existing systems are created to offload the load to the cloud or schedule workloads into the cloud considering factors such as cost, deadline. But to improve the quality of service (QoS) it is necessary to consider users inclination towards a cloud environment. User can have a customized billing model towards a cloud environment as AWS reserved instance provide or the user wants a system to transfer and execute workloads in an on-premise private cloud as it can be scaled without considering the impact of cloud. This dynamicity is lacking in existing works and the developed model considers this by taking input from the user before scheduling workload into the cloud known as "User-Preference".
Research Question:
"*Can execution of workloads with deadline be improved by implementing multi-agent based application using a user preference based data-aware resource provisioning algorithm in interconnected clouds?*"
The report is organized as follows. Section 2 contains the literature review of interconnected cloud and multi-agent based systems. Section 3 describes the technology and framework that is used to develop the Multi-agent system. Section 4 displays the design of the Multi-agent system as well as the architecture of the scheduler. Section 5 contains the implementation of the multi-agent system with detail description of the framework and technologies used. Section 6 describes the evaluation performed considering different test cases and scenarios. In Section 7, contains the conclusion and future work of this paper.

2 Related Work

Multi-Agent based systems are developed and being used to create adaptive software systems, the systems are flexible enough to change their functionality according to change in requirement. A Multi-agent system comprises of custom programmed agents, these agents can communicate with each other to complete the execution of the tasks. Using a Multi-Agent System (MAS) can make resource management in a cloud environment more flexible and self-reliant [5]. In Cloud Computing, resources are shared among the cloud service users. Multi-Agent System can enable the users to combine multiple cloud service providers, giving users the best services of multiple cloud platforms. Primary Goals for Multi-Agent based systems include:

1. Merging Computing resources provided by the different cloud service provider.

2. Efficient Organizing of the shared resources.

3. Contracts between Cloud service provider and user.

Agent-based systems for cloud computing has been explored to manage computing resources some of the significant work done by Hafez et al.[5] and Kwang et al.[2]. Their works focuses on creating agent-based systems for cloud service discovery, service negotiation and service composition. According to Hafez et al. [5] Multi-Agent (MAS) based model is stated to have following characteristics:

1. Each agent deployed on the system has knowledge about its own system rather than knowledge of the entire system.

2. Absence of a centralized control system.

3. Data is distributed.

4. Asynchronous computation of jobs

According to Hafez et al. and Prieta et al. [5; 6] communication and behaviour being the primary focus of the agents and taking into consideration of the above characteristics Multi-Agent Systems (MAS) is classified into the following categories :

1. Centralized Model:
   Agents that follow this model are homogeneous in nature and follow master-slave architecture to communicate with each other. Control System is with the master as the master is responsible for maintaining the life-cycle of agents.

2. Distributed Model:
   Agents are distributed throughout the system and controlled using a single control layer. Every agent present in the system has partial knowledge of the entire domain.

3. Hierarchical Model:
   In this model, some agents are empowered with rights to override the existing behaviour of the agent.

Following similar guidelines, many agent-based models have been developed to handle workloads in different methods. According to a survey and research carried out by Hafez et al. [5] and Kwang et al. [2], JADE is one such framework used to create Multi-Agent Systems. Elastic JADE by Umar et al. [4] a Multi-Agent system, to dynamically scale cloud resources on-demand was developed. In this article, the Multi-Agent System was used to offload the code from the local machine to the public cloud. As soon as the local machine is unable to process the data-intensive workload the similar workload which was configured in public cloud would be activated to process the data. Umar et. al. [4] tested the system using face detection application as a testing bed and it was done on AWS EC2 instance. Spawning or stoping EC2 instances when it was not utilized was implemented providing the feature of elasticity. But this system was "Vendor-Locked" and was focused on using AWS EC2 Instances only as well as there was the additional configuration that is to be done before initiation of the workload load execution.
Another approach towards Multi-Agent Systems is done by Anirban [7], a mobile agent platform was developed using JADE to control grid and cloud resources for execution of the workload. Agents were created dynamically according to the Jobs in Queue and were terminated accordingly if the jobs were less in the queue, resulting in efficient utilization of the resources. This model was developed to handle batch jobs as well as been tested to

handle Hadoop jobs. This agent system was not developed considering multiple clouds and jobs which has a deadline associated with it. Even though the elasticity and resilient was achieved but the overall system was less dynamic.
A Multi-Agent System (MAS) is developed by Fan et al. [3] to handle the workload with the deadline on multiple virtual machines present on a cloud environment. In this system, a JADE based Multi-Agent system is used to select virtual machine instance which can efficiently complete the execution of the workloads under the deadline associated with it. Fan et al. [3] have proposed a federated layer which has agents that would connect different cloud providers. Along with connecting different cloud providers, this system can scale up and down according to jobs in the queue, which makes this system efficient. But the proposed model was concerned about selecting Virtual Machine Instance rather than utilizing the Virtual machines efficiently.

2.1 Scheduling Frameworks

Table 1: Comparison of proposed with existing Scheduling Frameworks

Paper	**Deadline**	**Cost**	**Input File Transfer Time**	**Dynamic Provisioning**	**Config. of Public Machines**	**user preference**
[8]	yes	yes	no	no	no	no
[9]	yes	yes	no	no	no	no
[10]	yes	yes	no	no	no	no
[3]	yes	yes	no	no	no	no
[11]	yes	yes	no	no	no	no
[12]	yes	yes	yes	yes	yes	no
[13]	yes	yes	yes	yes	no	no
[14]	yes	yes	yes	yes	no	no
[15]	yes	yes	yes	yes	no	no
Proposed model	yes	yes	yes	yes	yes	yes

The primary focus of cloud computing has always been on providing a better quality of service (QoS) so many scheduling frameworks has been designed to satisfy QoS and make them compatible with the cloud environment. QoS in a cloud environment can be classified into task execution time, Cost, Reputation, Reliability and availability of the cloud environment. A framework proposed by Wang et al. [16] focuses on cost and task execution time while assuming the availability of the system constant. Wang et al. have proposed a new scheduling algorithm for hybrid cloud environment known as Adaptive-Scheduling-with-QoS-Satisfaction (AsQ) which focuses on increasing utilizing private cloud more for job completion as a result achieving optimal cost and task execution of deadline-based jobs. The proposed system lacks the ability of distributed computing as agent-based systems can provide as well as the proposed model has an issue of scalability i.e. it cannot reduce the excessive resources which makes the system less flexible.
Similar to Wang et al. [16] a scheduling framework is proposed by Singh et al. [17], which focuses on resource provisioning to achieve a better quality of service (QoS). According to Singh et al. the QoS service in a cloud environment is impacted by heterogeneity,

distributed resources which leads to difficulty in resource management. To resolve the issue of resource management the proposed framework to re-cluster the workload as per deadline, cost and energy efficiency then assign it to respective resources on the cloud environment. The proposed framework focused more on managing the existing resources and failed to consider scalability in the consumption of cloud resources.
As per Wang et al. [16] and Singh et al. [17] QoS can be classified into various categories Table 1 will show the comparison of different scheduling frameworks that were developed to execute workloads with deadline in the cloud environment. As shown in Table 1 the scheduling framework is categorized into Deadline 2.1.1, cost 2.1.2, Input File Transfer Time 2.1.3, Dynamic Provisioning 2.1.4, Configuration of Public Machines 2.1.5, User preference 2.1.6.

2.1.1 Deadline

Deadline for any workload can be stated as time constraint in which the job has to be completed. It has become the epicentre of many proposed scheduling frameworks as the deadline is the most important factor to achieve to provide better Quality of Service (QoS). As shown in Table 1 every framework is focused on satisfying deadline, but the approach followed by each of them is different as some are focused on rejecting deadline-based job's if there are not sufficient resources while some will spawn resources on-demand to complete the job execution on time. One such method of selecting similar job for execution which was previously executed is proposed by Nayak et al. [10]. Nayak et al. used Multi-criteria decision making algorithm VIKOR as a back-filling mechanism, which is responsible for ranking and scheduling jobs similar jobs to be deployed on Open Nebula. Another mechanism to handle deadline constraint jobs using lease and lease manager. Scheduling framework proposed by Nayak et al. [8] creates Deadline sensitive lease which used along with lease manager heiza. Heiza uses the Analytical Hierarchy Process (AHP) as a backfilling algorithm to schedule jobs for execution on the cloud. Another work proposed by Nayak et al. [9] uses backfilling algorithm along with the current time (CT) and Gap Time(GT) to identify and schedule jobs with the deadline on priority.
A Multi-agent based system (MAS) for handling workload with the deadline was introduced by Fan et al. [3]. In this system proposed by Fen et al. is based on 4 agents deployed on Open Stack cloud platform, these agents were responsible for System monitoring, Job dispatch, job execution and instance group management. The proposed agent-based system used Rough set theory to schedule jobs with a deadline to the Open cloud environment but this method was deployed and tested on the private cloud platforms created using Openstack and other cloud systems were only assumed.

2.1.2 Cost

After deadline every scheduling framework focuses on optimizing cost and aims to achieve efficent job execution in lowest cost possible. Nayak et al. [10] has proposed a theory that uses a VIKOR back-filling technique to avoid adding new instances to the resource pool and save cost. A similar approach by Fan et al. [3] considers cost per job to allocate resource. Nayak et al. [8] work with back-filling and AHP with Hieza also considers cost management with open nebula by reallocating the jobs to efficiently use computing resources, while Nayak et al. [9] proposed another model that also works on Job rejection on similarity which prevents loss of jobs witch further makes it cost-effective, But Bossche et al. [13] proposed a framework that makes an efficient way factoring Cheapest to the

public cloud which will line up the cheapest job first. Malawski et al.[14] proposed a scheduling technique that works better by using additional vendors which thereby manages cost per value effectively.
For Providing minimal cost per performance Total Moschakis et al.[11] used Trace Cost(TTC) and Cost performance efficiency(CPE)to determine the impact of performance by cost, it also compares parameters like TSA, SA and FPLT to measure average performance by percent, whereas Abdi et al. [12] compares different types of vendor instances for taking the most optimal cost for performing tasks. Work ow-aware DPDS (WA-DPDS) used by Malawski et al. [15] aims to utilize the Virtual Machines by keeping them busy with smaller jobs as other jobs are prepared thereby preventing wastage of resources over the cloud. Many theories attempt to provide cost-effective solutions to the work associated, but all theory lacks in some aspect or other, in terms of provisioning resource or managing critical resources hence we can conclude that the cost may differ from method to CSP we use.

2.1.3 Input File Transfer Time

Input file transfer time refers to the job transfer time to the cloud environment. This is also a vital parameter while scheduling job because it contributes to the overall effectiveness of job execution. As shown in table 1 techniques proposed by Nayak et al. [9; 8] uses lease to schedule and provision workloads with a deadline on cloud environment but they do not consider the time required to complete the execution of the job as a result obtained result is not accurate. Some techniques have focused on file transfer time, Bossche et al.[13] proposed a technique that would consider input file size and transfer time to compute the feasibility of job for certain cloud environment before transferring it for execution.
Sharing of the input file among the jobs is also a mechanism introduced by Malawski et al. [14] to reduce the overhead of transferring same jobs to the cloud environment the proposed model uses Object Storage services provided by cloud service providers like Rackspace cloud files and Amazon S3. Other work proposed by Malawski et al.[15] uses workflow technique to schedule jobs in virtual machines and files sharing is done between virtual machines to complete the job execution.

2.1.4 Dynamic Provisioning

Scalability and Elasticity have been the key characteristics of cloud computing and efficient scheduling of jobs in the cloud. The system proposed by Nayak et al. [10] aims at utilizing existing computing resources efficiently, on the contrary, the system proposed by Fan et al. [3] considers the computing resources while scheduling jobs in the cloud using Multi-agent based system and aims towards being cost-efficient. Partial consideration of scalability is done by Nayak et al. [9; 8] as they consider using resource pool to schedule and reschedule jobs.
Scheduling framework proposed by Nayak et al. and Malawski et al. [13; 14] both focus on utilizing existing computing resource pool similar to Nayak et al. [10] but it focuses on multiple cloud environment hence it also fails to provide complete elasticity. A Cloud provider-level restriction (CPLR) model is proposed by Abdi et al. [12] that is a mathematical model with no limitation on resource provisioning hence there is no limitation on virtual machine instance creation on fixed resource pool and aims at re-utilizing the

instances of the virtual machines. In the developed model dynamic provisioning is considered on CPU core utilization level as result spawning new agents and terminating unused agent is implemented.

2.1.5 Different Configuration of Public Machines

Consideration towards the multi-cloud environment has the additional overhead of configuring the public or private cloud along with scalability. Some scheduling frameworks shown in Table 1 are been developed and tested on a single cloud environment even though they have been planned for multiple clouds but was never tested on one. Frameworks proposed by Nayak et al. [10; 9; 8] is developed on considering only Open nebula as the sole cloud service provider and is not designed for multi-cloud. Partial consideration of multi-cloud is done by Fan et al.[3], in the proposed agent-based multi-cloud environment but the proposed system was only tested and implemented on Open stack.
Abdi et al. [12] has implemented scheduling and tested it out using CPLEX optimization software for simulation similar simulation was also done by Bossche et al. and Malawski et al. [13; 14; 15]. Simulation can provide results but it fails to consider the real scenarios of latency while transferring file or poor network performance. In the developed framework configuration towards any public cloud service provider is handled using scripts that can be executed while registering the virtual machines.

2.1.6 User preference

As shown in Table 1 none of the proposed models considers user preference while scheduling jobs in a multi-cloud environment. Most of the model considers cost and deadline to be the most important factors but the cost can vary according to the type of services purchased by cloud user. AWS reserved instance provides a custom pricing model to different cloud service user depending upon the time and type of computing resources required by the user. In the developed framework user-preference along with deadline is the most important factor which is considered while provisioning jobs to the respective cloud. This framework not only provides an additional layer of flexibility to the existing work but also improves Quality of Service (QoS), which is the epicentre of cloud computing.

3 Methodology

JADE(Java Agent Development Framework) is one of the most preferred frameworks to create agent-based systems [5; 2; 3]. A survey carried out by Hafez et al. [5] have analyzed different agent-based frameworks and concluded that JADE provides better flexibility in terms of configuration and development. Similarly, Kwang et al. [2] has proposed various criteria that can be considered while developing agents and have identified that JADE can be used to develop a Multi-agent based systems. A Multi-agent based framework designed by Fan et al. [3] is based on JADE and the decision to utilize JADE was because of it is developed on JAVA and has better-established community and flexibility than other programming languages. The Multi-agent based system is developed using Java and JADE (Java Agent Development Framework). As research focuses on user-preferences based scheduling of the deadline based workloads in multiple cloud, User interface (UI) is necessary to take user-preference from the user while scheduling jobs, So UI is developed on JavaFX. To store the metrics such as Job name, Job Transfer time, Job Completion

time and user preference as well as VM details are stored in Postgres database.
Scheduling algorithm are been developed for the multi-cloud environment one of those algorithms is Resource provisioning algorithm [18], this algorithm focuses on the scheduling of jobs in hybrid cloud environment important criteria that the scheduler focuses on is data transfer time. Some work has been done to enhance the performance and QoS of this algorithm such as [19] which focuses on sharing of data between the jobs to increase the efficiency of the scheduler. Calheiros et al. [20] proposed a work by modifying Resource provisioning algorithm to consider the AWS spot instances to reduce the cost of the job execution. Vecchiola [21] proposed to use resource pools with resource provisioning algorithm to handle scientific workloads on hybrid cloud. In developed work for Scheduling jobs in multiple cloud environment, a customized approach of resource provisioning algorithm [18] was implemented to consider user preference for selecting virtual machines.
Job execution details are fetched from respective cloud environments (AWS and Openstack) details such as Job transfer time and Job Completion time is retrieved from agents deployed on the cloud platform and parameters such as Virtual machine and user preference is collected while provisioning the jobs. Based on the job completion time and Job Transfer time the scheduling of the jobs is impacted, virtual machines with lower transfer time and completion time is given priority while scheduling the jobs.
The approach towards the methodology for the developed project can be summarized as follows:

- **Language:** Java
- **Framework:** JADE (Java Agent Development Framework)
- **Database:** Postgres
- **Cloud Environments:** AWS, Openstack
- **Database Parameters:** Job Completion Time, Job transfer time and User-Preference

4 Design Specification

Developing a Multi-agent based system to schedule and handle workloads in a multi-cloud environment is the primary goal of the research. For scheduling the workload in multi-cloud resource provisioning algorithm with consideration to user-preference is developed. Application design and Master-Slave Architecture are the respective designs and architecture followed while developing the application.

4.1 Application Design

As shown in component diagram 1 the overall application has two major components cloud providers and scheduler. The scheduler will be the java based application running on any machine with proper UI support and java installed and configured. The scheduler allows the user to upload jobs and based on user preference, jobs are configured on respective virtual machines on the cloud based on preference. Cloud layer have public and private cloud service providers with agents configured on their respective virtual machines. These agents will follow master-slave architecture to manage job execution

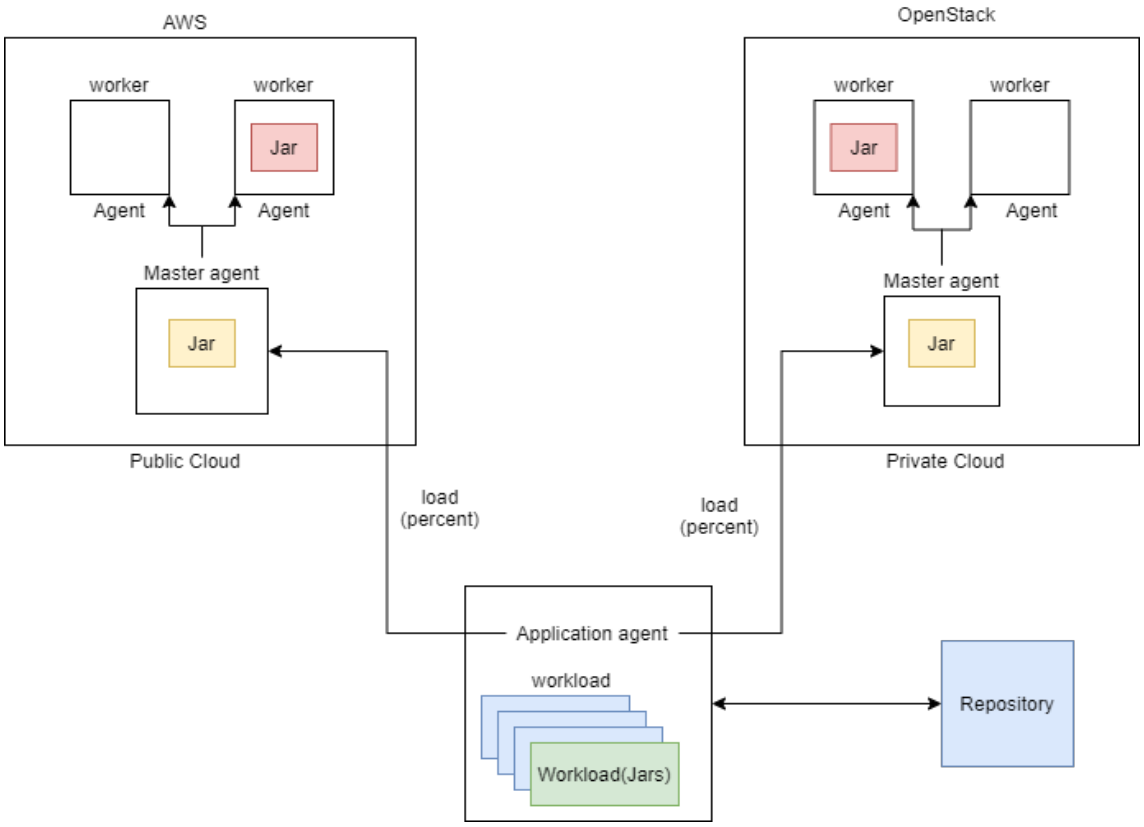

Figure 1: Component Diagram

and generating logs with job execution details such as Job name, Job transfer time and job completion time. This detail about the jobs will be used by agents to schedule jobs for better performance and efficiency. Jobs are an independent unit of tasks that will be analysed and uploaded to the cloud. Along with generating logs master agent will also be responsible to spawn new agents when required and delete unused agents according to the need.

4.2 Master-slave architecture

Master-slave architecture is prominently used in different scenarios for accomplishing different tasks such as database replication, load balancing and making system resilient. A similar approach has been followed to develop the Multi-agent based model, here master node is present on each virtual machine that has been registered with the scheduler. The master node is developed using the JADE framework and it has the functionality to scale up and down (i.e. to spawn and terminate agents) according to the workload that has been sent by the scheduler. Any slave nodes that have been inactive (i.e. not executing workload) for more than 2 minutes. Master nodes are also responsible to get the workload from the scheduler and generate logs with parameters such as Job name, Job completion time and Job transfer time this logs will be further processed by the scheduler.
Slave nodes are also developed using JADE and master node is responsible for managing the life-cycle of the slave nodes. Slave nodes fetch the workloads assigned by master node and execute the workload, during execution each slave node will calculate the total time taken by them to complete the job execution and report back to master node when the

job completed. As soon as the job execution is finished the agents go into dormant stage waiting for new jobs to be assigned to them by the master node.

5 Implementation

The developed Multi-agent scheduler has multiple components such as Scheduler and Agents. Agents are developed using JADE framework [22] and scheduler is developed using core java, JavaFX is used to develop a user interface that can be accessed by the user to input jobs and preference which would be considered by the scheduler. Communication between application and agents deployed on any cloud environment is established by using Secure Shell (SSH), SSH functionality is provided by java based library known as JSCH [23].

5.1 JADE Framework

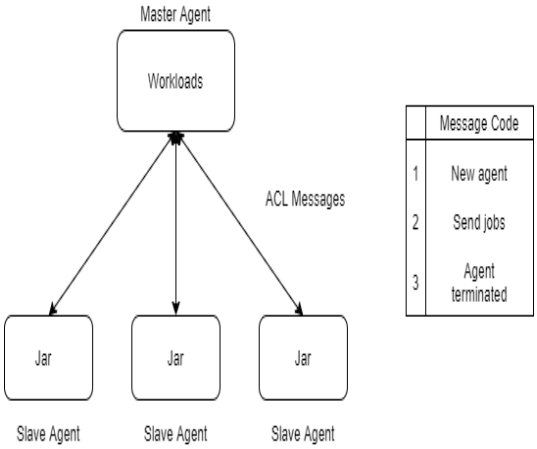

Figure 2: Agents Component Diagram

As shown in the above Figure 2 the developed multi-agent system follows master-slave architecture. Master and slave agent uses ACLMessages to communicate with each other and allocate the job. As soon as a master agent is created service description of type "intercloud" and ontology of "master-ontology" is created and in case of slave agents type is "intercloud" and ontology is "slave-ontology" this type and ontology classifies the agents based on the respective responsibilities. As shown in the sequence diagram of agent figure 3, as the workload is received by the master agent, slave agent is created and registered with "intercloud" type message with message code "1" is sent to master agent indicating that slave agent is ready to execute the new workload. The message is parsed by master agent and if there is a new job in a queue new job is assigned to the slave agent with message code "2". Then slave agent will start the execution of the job and will return the total time required to complete the execution of the job. After job execution is finished master node will check for a new job if there is no new job then the slave will go into a dormant state for 2 minutes before termination. As soon as slave agent is terminated after the 2 minutes the agent will send a new message with message code "3" indicating that the agent is terminated and the master agent will remove agents from its queue so that no new jobs are allocated to it.

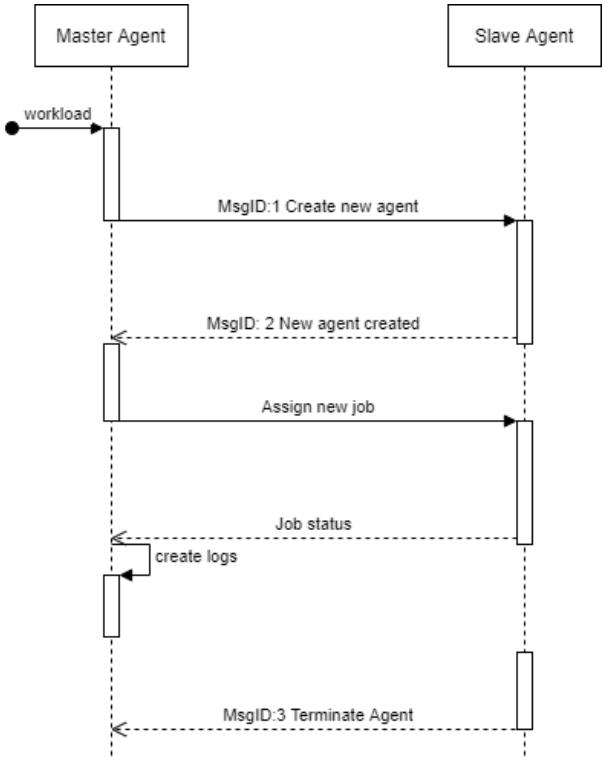

Figure 3: Agents workflow diagram

5.2 Resource Provisioning Algorithm

Algorithm 1 Resource Provisioning with user preference

Input: jobs, userPref, privateCore, publicCore
Output: jobExecutionDetails

```
1  publicPreference= jobs*(userPref/100.0)
2  privatePreference=jobs - publicPreference
3  for jobs do
4      jobsWithDeadline(jobs)
5      jobsWithoutDeadlines(jobs)
   /* Sort jobs with deadline in ascending order to execute lowest deadline jobs first */
6  sortJobsWithDeadline
   /* Jobs with deadline on private cloud virtual machines                          */
7  while jobsWithDeadline > 0 && privateCore > 0 && privatePreference != 0 do
8      getPrivateVMdetails
9      transferWorkload
10     privateCore−1
11     jobsWithDeadline−1
   /* Jobs with deadline on public cloud virtual machines                           */
12 while jobsWithDeadline > 0 && publicCore > 0 && publicPreference != 0 do
13     getPublicVMdetails
14     transferWorkload
15     publicCore−1
16     jobsWithDeadline−1
```

Algorithm 2 Resource Provisioning with user preference continued..

```
/* Jobs without deadline on private cloud virtual machines */
while privatePreference != 0 && jobsWithoutDeadlines > 0 && privateCore > 0 do
    getPrivateVMdetails
    transferWorkload
    privateCore−1
    jobsWithoutDeadlines−1
/* Jobs without deadline on public cloud virtual machines */
while publicPreference != 0 && jobsWithoutDeadlines > 0 && publicCore > 0 do
    getPublicVMdetails
    transferWorkload
    publicCore−1
    jobsWithoutDeadlines−1
```

As shown in above user preference based resource provisioning algorithm is used by the scheduler. List of jobs, User preference, Private core and Public core available for the jobs to be scheduled. Based on user preference number of jobs that is to be allocated to the virtual machines is calculated and jobs are classified into two categories i.e. jobs with deadline and jobs without a deadline. Jobs with a deadline are given priority over jobs without a deadline and the list of jobs which have a deadline is sorted in ascending order to complete execution of those priority jobs. After jobs are sorted they are scheduled to the private and public virtual machines if the number of cores in respective virtual machines are available and preference towards the respective type of cloud i.e. (private, public) is not 0.

5.3 Jobs/Workloads

Workloads are independent Jars that can be executed in an environment that has Java installed and configured. In the developed application workloads are required to be placed in "workload" folder in the current directory, the application has the functionality to create the folder if such folder is not found. The naming of workload is mandatory to be in following format "<deadline in minutes> − <Jobname>.jar" e.g. 05-piEstimation.jar.

5.4 Sequence diagram

As shown in Figure 4 the workload is submitted by a user on Scheduler with a proper naming convention and in workload folder present in the current directory. As soon as the workload is checked it will be supplied to resource provisioning algorithm for virtual machine selection and allocation. As per the resource provisioning algorithm, the jobs are then transferred to the respective virtual machines. These virtual machines are configured with master agents that can accept workloads and spawn or reuse old slave agents as per requirement. The jobs are transferred to the respective slave nodes for execution and soon as the execution is completed job status is returned in terms of messages to the master agent. Master Agent will generate logs for the scheduler that is deployed on local machines, logs will be parsed by scheduler collecting and computing parameters such as Job name, Job Transfer time, Job Execution time and user preference. These parameters will be stored in the database/repository which will be fetched and used in scheduling the jobs.

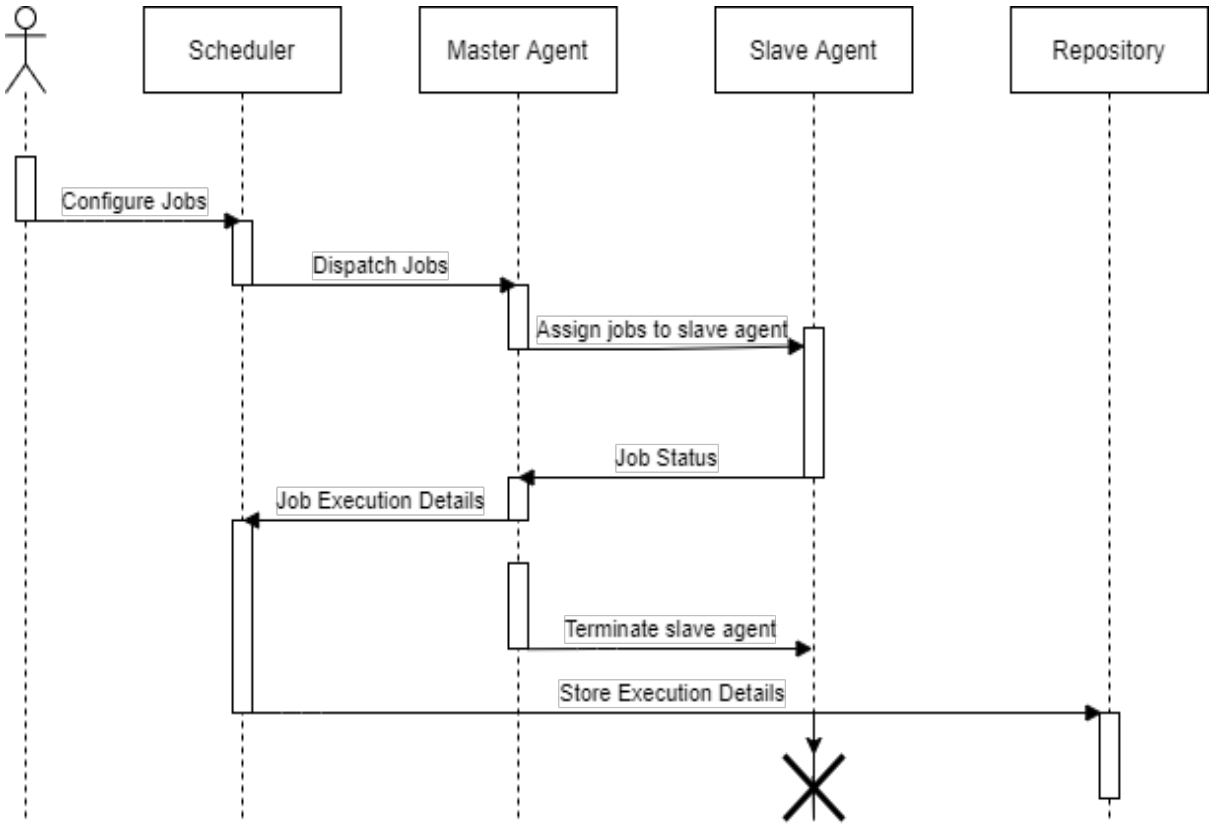

Figure 4: Sequence Diagram

5.5 Database Diagram

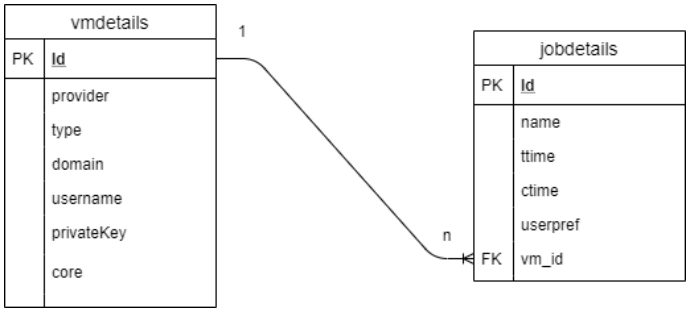

Figure 5: Class Diagram

As shown in class diagram Figure 5 the overall database/repository layer consists of two tables vmdetails and jobdetails. vmdetails and jobdetails form a one to many relationships i.e. many jobs can be deployed on one virtual machine. vmdetails have columns which include Id for the primary key, provider denotes the cloud service provider, type shows the type of cloud service provider (i.e. public or private), domain gives domain name of the virtual machine that will be used while accessing virtual machines, username and private key are the credentials that will be utilized by JSCH while accessing virtual machines and core indicates the number of cores that a virtual machine has. Job details store the job executions details in it and have columns such as id as a primary key, name gives name of the job, ttime is the transfer time taken by the respective job to be

transferred, ctime is the total time taken by the job for completion of execution. userpref is the user preference that is considered while the job was scheduled.

6 Evaluation

In this research, a multi-cloud based scheduler to execute deadline-based jobs is developed. This Scheduler uses a modified version of resource provisioning algorithm to consider user preference while scheduling jobs on the cloud along with that scheduler has an additional feature to select the best performing virtual machine every time the job is to be scheduled based on previous job execution data. The scheduler is developed on an agent development framework known as JADE which provides better utilization of computing resources on the cloud by managing agents on thread level [22]. As this scheduler is based on multi-agent for better efficiency and control, JADE framework is used for the development of this scheduler. Master-Slave architecture is followed for agent development, this architecture is perfectly compatible with JADE as it works on FIPA communication model and provides API for micro-control of the fellow agents.
Satisfying the deadline and completing the execution of jobs efficiently are the primary goal of the developed scheduler and these parameters can be measured using Job transfer time, Job Completion time, Deadline of the Job and User preference. These parameters can be described as follows:

- **Job Transfer Time:**
  It is the time taken by each job to be transferred to the respective virtual machines present and configured in the cloud environment. It is represented in seconds and stored in the database as a numerical value.

- **Job Completion Time:**
  It is the time taken by job to complete its execution. Job completion time is calculated in seconds and stored in the database as a numerical value.

- **Deadline:**
  It denotes the deadline assigned or allocated to the respective Job/Workload by the user. Deadline is fetched from the Job names as per prerequisite these will have a deadline on the job name. It is calculated in minutes and stored in the database as a numerical value.

- **User Preference:**
  It is a parameter provided by the user and implemented by the resource provisioning algorithm to schedule the jobs. As per application design, the user preference is taken as an input from the user as a numerical value, this numerical value is stored in the database along with other job details.

- **Total Job execution time:**
  This factor is the total time required by the job to complete its execution, that will be compared with the deadline to know if the deadline is violated or not. Total Job execution is calculated during the analysis of the different test cases and it is calculated as follows:
  Total execution time= Job Transfer Time + Job Completion Time

Scheduler performance analysis:
As the developed scheduler is designed to be resilient in terms of job execution, failure to complete the execution of a job can occur in certain scenarios. But the deadline provided by the user can be violated as virtual machines might have existing jobs/processes running or virtual machines system configuration is low than required. A job to estimate the value of pi using Monte Carlo's method is used to evaluate the performance of the scheduler. To evaluate the performance and a similar approach was taken by Fan et al. [3]. This job with different deadlines is scheduled and executed on a multi-cloud environment (i.e AWS and Openstack).

Configuration of the Virtual machines used for testing:
AWS(Public cloud): 4 Single core machines with Linux AMI2 operating system running on the virtual machines.
Openstack(Private Cloud): 1 four core machine with Linux centos operating system running on the virtual machine.

Jobs:
For the evaluation, approximately 24 jobs and 8 jobs at a time is scheduled using the scheduler for each test case. Jobs are configured with the longest deadline and some jobs with the lowest deadline to verify the performance of the scheduler with the deadline. Figure 6 shows the snapshot of the dataset used for evaluation, complete dataset will be shared with the project artefacts.

id	name	ttime	ctime	vm_id	userpref	deadline	deadline_inSeconds	total_execution_time	deadline_breach	cloud_type
1	10-PiEstimation-6.jar	3	649	5	50	10	600	652	yes	public
2	02-PiEstimation-5.jar	29	138	2	50	2	120	167	yes	private
3	03-PiEstimation-2.jar	29	139	2	50	3	180	168	no	private
4	04-PiEstimation-1.jar	29	137	2	50	4	240	166	no	private
5	05-PiEstimation-3.jar	20	450	2	50	3	180	470	yes	private
6	10-PiEstimation-6.jar	3	648	5	50	10	600	651	yes	public
7	09-PiEstimation-7.jar	3	711	4	50	9	540	714	yes	public
8	06-PiEstimation-4.jar	3	826	3	50	6	360	829	yes	public
9	08-PiEstimation-8.jar	3	861	1	50	8	480	864	yes	public
10	02-PiEstimation-5.jar	32	136	2	50	2	120	168	yes	private
11	03-PiEstimation-2.jar	22	139	2	50	3	180	161	no	private
12	04-PiEstimation-1.jar	13	137	2	50	4	240	150	no	private
13	05-PiEstimation-3.jar	3	135	2	50	5	300	138	no	private
14	06-PiEstimation-4.jar	3	598	3	50	6	360	601	yes	public
15	08-PiEstimation-8.jar	3	617	1	50	8	480	620	yes	public
16	10-PiEstimation-6.jar	3	649	5	50	10	600	652	yes	public
17	09-PiEstimation-7.jar	3	711	4	50	9	540	714	yes	public
18	06-PiEstimation-4.jar	3	630	3	50	6	360	633	yes	public
19	08-PiEstimation-8.jar	3	643	1	50	8	480	646	yes	public
20	10-PiEstimation-6.jar	3	649	5	50	10	600	652	yes	public
21	04-PiEstimation-1.jar	13	137	2	50	4	240	150	no	private
22	02-PiEstimation-5.jar	3	115	2	50	2	120	118	no	private
23	03-PiEstimation-2.jar	3	138	2	50	3	180	141	no	private
24	05-PiEstimation-3.jar	3	134	2	50	5	300	137	no	private

Figure 6: Dataset Snapshot

6.1 Discussion

Using the above mention testing bed the evaluation of the scheduler is completed and can be broken into 5 criteria mentioned below:

1. Jobs only with deadline user preference is 50 percent.
   In this test case as shown in Figure 7, 24 jobs were scheduled. All of the 24 jobs were with the deadline and user preference was 50% (i.e. 50% on public and 50% private) as per the user preference 12 jobs were scheduled on private and 12 jobs were scheduled on the public cloud. From the analysis shown in Figure 7 deadline

		Deadline Breached		
		yes	no	total
Cloud Type	Private	3	9	12
	Public	12	0	12
	Total	15	9	

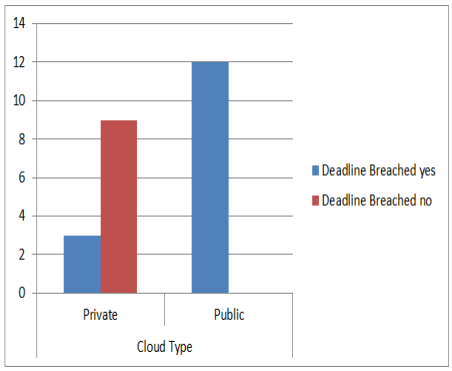

Figure 7: Evaluation 1

was breached by all of the 12 jobs scheduled on public cloud indicating the AWS has maximum deadline breach and Openstack has 3 deadline breach out of 12 jobs giving a better performance as compared to the public cloud.

2. Jobs without deadline user preference is 50 percent.
   As shown in figure 8, 24 jobs without deadline were scheduled. User preference

		Deadline Breached		
		yes	no	total
Cloud Type	Private	0	12	12
	Public	0	12	12
	Total	0	24	

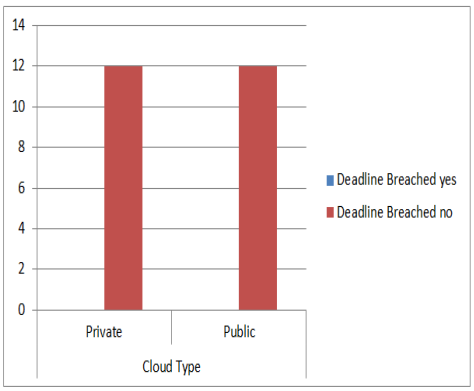

Figure 8: Evaluation 2

was 50% (i.e. 50% on public and 50% on private) and according to user preference, 12 jobs were scheduled on each cloud type. And based on resource provisioning algorithm as there were no deadline-based jobs 12 jobs were scheduled on private and 12 jobs were scheduled on public cloud and there was no deadline breach.

3. Mixed jobs with user preference is 50 percent.
   In this test case jobs with deadline and jobs without a deadline are scheduled for execution. As per the results obtained shown in figure 9, both types of jobs that were scheduled using the resource provisioning algorithm, there execution completed successfully without any deadline breach. Using resource provisioning algorithm jobs with the deadline was scheduled on private cloud (AWS) and non deadline-based jobs were scheduled on the public cloud. User preference was 50% so jobs were equally distributed among the public and private clouds.

	Deadline Breached		
	yes	no	total
Cloud Type Private	0	12	12
Cloud Type Public	0	12	12
Cloud Type Total	0	24	

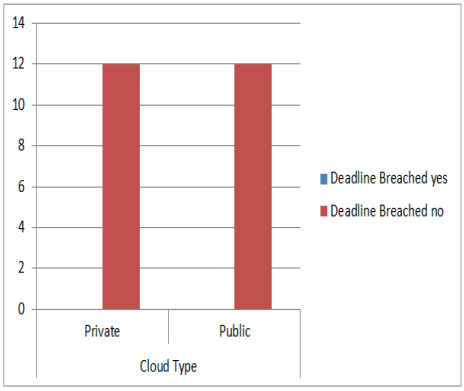

Figure 9: Evaluation 3

4. Mixed jobs only on public cloud.
   In this test case jobs with and without deadline were scheduled in a multi-cloud

	Deadline Breached		
	yes	no	total
Cloud Type Private	0	0	0
Cloud Type Public	8	4	12
Cloud Type Total	8	4	

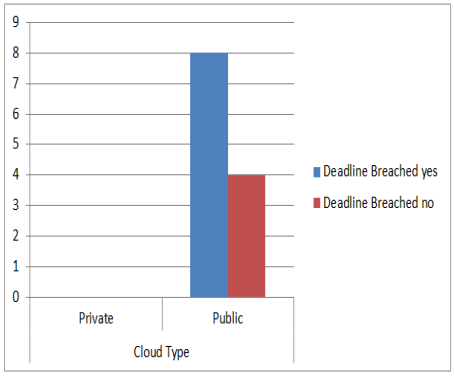

Figure 10: Evaluation 4

environment(AWS, Openstack) with user preference set to 100% on the public cloud. As there are limited cores available on public cloud some of the jobs will fail to execute, as per our testing environment only 4 computing core is available in the private cloud. In this test case, the same observation is observed as only 12 jobs with deadline were scheduled for execution and the remaining 12 non deadline-based jobs were failed and notified by the scheduler. As shown in figure 10, scheduling jobs in public cloud executes the jobs with a deadline but most of the jobs violate the deadline constraint. So based on this test execution it can be concluded that both public and private cloud can give the best efficiency for job scheduling.

5. Mixed jobs only on private cloud.
   In this test case jobs with and without deadline were scheduled in a multi-cloud environment(AWS, Openstack) with user preference set to 100% on the private cloud. As there are limited cores available in the private cloud, and resource provisioning algorithm focuses on prioritizing jobs with the deadline for execution. In this experiment out of 24 jobs, 12 jobs fail to execute as failed jobs are without a deadline and there are a limited number of private cloud core available for the scheduler to use. As shown in figure 11, jobs with a deadline are scheduled on private cloud and

	Deadline Breached			
		yes	no	total
Cloud Type	Private	1	11	12
	Public	0	0	0
	Total	1	11	

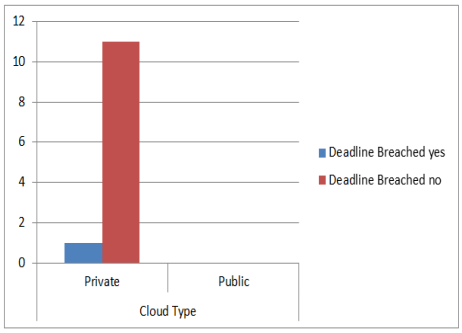

Figure 11: Evaluation 5

most of the jobs do not violet the deadline constraint.

7 Conclusion and Future Work

Interconnected cloud had been developed to handle the issues of single cloud environment such as availability and scalability. Users can opt to use multi-cloud to get computing resources and efficiently utilize these resources. As soon as there is a shortage of resources, additional computing resources can be configured and registered with the scheduler to be utilized. Jobs/workload with deadline constraint is needed to be handled at priority as they might have important process or operation to be executed.
User preference based resource provisioning algorithm aims to increase the Quality of service (QoS) for cloud computing. Introduction of user preference parameter not only provides the user with the flexibility of selecting a cloud environment for job execution but also increases the performance of scheduler in certain scenarios. As observed in the second evaluation and third evaluation of the developed scheduler, jobs with a deadline and sufficient computing cores available on the cloud environment can give 100% success in job execution and satisfying the deadline constraint of the job. Jobs tend to fail when there is less number of virtual machines registered with the scheduler. While considering user preference there is also an additional challenge that is faced by scheduler while provisioning jobs with a deadline that is it fails to complete execution of job under defined deadline. This issue can be resolved by registering virtual machines which have higher system configuration(i.e. more computing cores). Jobs with extremely low deadline even though the actual execution time is a lot more than the execution time tends to be a job with deadline constraint violated. As per evaluation 1, 4 and 5, it is clear that the number of jobs should be less than or equal to the collective number of cores available on the particular cloud type (i.e. private or public).
For future work, the developed scheduler can be optimized to use the registered virtual machines efficiently but it can also be further enhanced by having additional ability to spawn new virtual machines on cloud-based on previous jobs performance. Ability to increase the resource pool and reduce it based on jobs provisioned to the cloud will enhance the performance of the scheduler and will also remove the overhead of the user to manage the cloud environment. With the feature of increasing resource pool, the agents deployed on the cloud environment can also be modified to consider communication

between virtual machines present in a different cloud environment. This will enable agents to become more intelligent and can optimize there performance and jobs allocations effectively with every job.

References

[1] A. NADJARAN TOOSI, R. N. CALHEIROS, and R. BUYYA, "Interconnected cloud computing environments: Challenges, taxonomy, and survey.," *ACM Computing Surveys*, vol. 47, no. 1, pp. 7:1 – 7:47, 2014.

[2] S. Kwang Mong, "Agent-based cloud computing.," *IEEE Transactions on Services Computing, Services Computing, IEEE Transactions on, IEEE Trans. Serv. Comput*, vol. 5, no. 4, pp. 564 – 577, 2012.

[3] C.-T. Fan, Y.-S. Chang, and S.-M. Yuan, "Vm instance selection for deadline constraint job on agent-based interconnected cloud.," *Future Generation Computer Systems*, vol. 87, pp. 470 – 487, 2018.

[4] U. Siddiqui, G. Tahir, A. Rehman, Z. Ali, R. Rasool, and P. Bloodsworth, "Elastic jade: Dynamically scalable multi agents using cloud resources.," *2012 Second International Conference on Cloud and Green Computing, Cloud and Green Computing (CGC), 2012 Second International Conference on, International Conference on Cloud and Green Computing*, pp. 167 – 172, 2012.

[5] M. G. Hafez and M. S. Elgamel, "Agent-based cloud computing: A survey.," *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud), Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th International Conference on, ficloud*, pp. 285 – 292, 2016.

[6] F. De la Prieta, S. Rodríguez-González, P. Chamoso, J. M. Corchado, and J. Bajo, "Survey of agent-based cloud computing applications.," *Future Generation Computer Systems*, vol. 100, pp. 223 – 236, 2019.

[7] A. Bhattacharya, "Mobile agent based elastic executor service.," *2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE), Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, pp. 351 – 356, 2012.

[8] S. C. Nayak, S. Parida, C. Tripathy, and P. K. Pattnaik, "An enhanced deadline constraint based task scheduling mechanism for cloud environment.," *Journal of King Saud University - Computer and Information Sciences*, 2018.

[9] S. C. Nayak and C. Tripathy, "Deadline sensitive lease scheduling in cloud computing environment using ahp.," *Journal of King Saud University - Computer and Information Sciences*, vol. 30, no. 2, pp. 152 – 163, 2018.

[10] S. C. Nayak and C. Tripathy, "Deadline based task scheduling using multi-criteria decision-making in cloud environment.," *Ain Shams Engineering Journal*, vol. 9, no. 4, p. 3315, 2018.

[11] I. A. Moschakis and H. D. Karatza, "Multi-criteria scheduling of bag-of-tasks applications on heterogeneous interlinked clouds with simulated annealing.," *The Journal of Systems Software*, vol. 101, pp. 1 – 14, 2015.

[12] S. Abdi, L. PourKarimi, M. Ahmadi, and F. Zargari, "Cost minimization for deadline-constrained bag-of-tasks applications in federated hybrid clouds.," *Future Generation Computer Systems*, vol. 71, pp. 113 – 128, 2017.

[13] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds.," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 973 – 985, 2013.

[14] M. Malawski, K. Figiela, and J. Nabrzyski, "Cost minimization for computational applications on hybrid cloud infrastructures.," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1786 – 1794, 2013.

[15] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds.," *Future Generation Computer Systems*, vol. 48, no. Special Section: Business and Industry Specific Cloud, pp. 1 – 18, 2015.

[16] W.-J. Wang, Y.-S. Chang, W.-T. Lo, and Y.-K. Lee, "Adaptive scheduling for parallel tasks with qos satisfaction for hybrid cloud environments.," *Journal of Supercomputing*, vol. 66, no. 2, pp. 783 – 811, 2013.

[17] S. Singh and I. Chana, "Resource provisioning and scheduling in clouds: Qos perspective.," *Journal of Supercomputing*, vol. 72, no. 3, pp. 926 – 960, 2016.

[18] A. Nadjaran Toosi, R. O. Sinnott, and R. Buyya, "Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using aneka.," *Future Generation Computer Systems*, vol. 79, no. Part 2, pp. 765 – 775, 2018.

[19] S. Tuli, R. Sandhu, and R. Buyya, "Shared data-aware dynamic resource provisioning and task scheduling for data intensive applications on hybrid clouds using aneka.," *Future Generation Computer Systems*, vol. 106, pp. 595 – 606, 2020.

[20] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds.," *Future Generation Computer Systems*, vol. 28, no. 6, pp. 861 – 870, 2012.

[21] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy, and R. Buyya, "Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka.," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 58 – 65, 2012.

[22] B. Fabio, P. Agostino, and R. Giovanni, "Developing multi-agent systems with jade.," *Intelligent Agents VII Agent Theories Architectures and Languages : 7th International Workshop, ATAL 2000 Boston, MA, USA, July 7–9, 2000 Proceedings*, p. 89, 2001.

[23] Jcraft.com, "Java Secure Channel." `http://www.jcraft.com/jsch/`, 2020. [Online; accessed 09-August-2020].