EXTENSION OF AN OPEN SOURCE RESOURCE MANAGEMENT TOOL FOR HETEROGENEOUS CLOUD DATA CENTERS: IMPLEMENTATION AND EVALUATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

TAHA DOĞAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2022

Approval of the thesis:

EXTENSION OF AN OPEN SOURCE RESOURCE MANAGEMENT TOOL FOR HETEROGENEOUS CLOUD DATA CENTERS: IMPLEMENTATION AND EVALUATION

submitted by **TAHA DOĞAN** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** ______________

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** ______________

Prof. Dr. Şenan Ece Güran Schmidt
Supervisor, **Electrical and Electronics Engineering, METU** ______________

Examining Committee Members:

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Engineering Department, METU ______________

Prof. Dr. Şenan Ece Güran Schmidt
Electrical and Electronics Engineering Department, METU ______________

Prof. Dr. Cüneyt F. Bazlamaçcı
Computer Engineering Department, İzmir Institute of Technology ______________

Assist. Prof. Dr. Serkan Sarıtaş
Electrical and Electronics Engineering Department, METU ______________

Assist. Prof. Dr. Pelin Angın
Computer Engineering Department, METU ______________

Date: 11.02.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Taha Doğan

Signature :

ABSTRACT

EXTENSION OF AN OPEN SOURCE RESOURCE MANAGEMENT TOOL FOR HETEROGENEOUS CLOUD DATA CENTERS: IMPLEMENTATION AND EVALUATION

Doğan, Taha
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Şenan Ece Güran Schmidt

February 2022, 74 pages

Cloud Computing is enabled by the virtualization of computing resources to realize users' requests of virtual machines (VMs) and data processing in the scope of Infrastructure as a Service (IaaS) and Software as a Service (SaaS) respectively. The current heterogeneous cloud data centers incorporate hardware accelerators in addition to the conventional servers to offer these services more efficiently. It is an important research problem to allocate heterogeneous physical computing resources to a mixture of IaaS and SaaS requests. On the one hand, the requirements of the requests should be satisfied. On the other hand, resource use and power consumption should be efficient. OpenStack is a popular cloud management platform that focuses on IaaS. It collects the user requests and accordingly instantiates VMs in the cloud data center. The default resource allocation of OpenStack only aims for fulfilling the user request without any further optimizations regarding efficient resource use. This thesis develops, implements and evaluates an extension of OpenStack to address the resource management and allocation of heterogeneous cloud data centers to address the requirements that we list above. The default filter-based resource allocation method of OpenStack only targets fulfilling the request requirements without any optimization concern. To this end, we integrate an ILP-based cloud resource allocation method with power minimization objectives in the most recent version of OpenStack that was available at the time of writing this thesis. We develop a software architecture that interfaces the ILP solver which provides all the required messaging according to the defined APIs of OpenStack. The entire extended implementation is realized in a laboratory set-up that features cloud servers, a 40 Gbps network and FPGA cards with hardware accelerators that represent a heterogeneous cloud data center.

Keywords: Cloud computing, Hardware accelerator, Resource management, OpenStack, Nova

ÖZ

HETEROJEN BULUT VERİ MERKEZLERİ İÇİN BİR AÇIK KAYNAK KAYNAK YÖNETİM ARACININ GENİŞLETİLMESİ: GERÇEKLEŞTİRİM VE DEĞERLENDİRME

Doğan, Taha
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Şenan Ece Güran Schmidt

Şubat 2022 , 74 sayfa

Bulut Bilişim, sırasıyla Altyapının Servis Olarak Sunulması (IaaS) ve Yazılımın Servis Olarak Sunulması (SaaS) kapsamında kullanıcıların sanal makine (SM) ve veri işleme isteklerini gerçekleştirmek için bilgi işlem kaynaklarının sanallaştırılmasıyla sağlanır. Mevcut heterojen bulut veri merkezleri, bu hizmetleri daha verimli bir şekilde sunmak için geleneksel sunuculara ek olarak donanım hızlandırıcıları içerir. Heterojen fiziksel bilgi işlem kaynaklarını IaaS ve SaaS isteklerine tahsis etmek önemli bir araştırma problemidir. Bir yandan, taleplerin gereklilikleri yerine getirilmelidir. Öte yandan, kaynak kullanımı ve güç tüketimi verimli olmalıdır. OpenStack, IaaS'ye odaklanan popüler bir bulut yönetim platformudur. Kullanıcı isteklerini toplar ve buna göre bulut veri merkezindeki VM'leri başlatır. OpenStack'in varsayılan kaynak tahsisi, verimli kaynak kullanımına ilişkin herhangi bir ilave optimizasyon olmaksızın yalnızca kullanıcı talebini yerine getirmeyi amaçlar. Bu tez, yukarıda listelediğimiz gereksinimleri karşılamak için heterojen bulut veri merkezlerinin kaynak yönetimi ve tahsisine yönelik bir OpenStack uzantısı geliştirir, uygular ve değerlendirir.

OpenStack'in varsayılan filtre tabanlı kaynak tahsis yöntemi, herhangi bir optimizasyon endişesi olmadan yalnızca istek gereksinimlerini karşılamayı hedefler. Bu amaçla, bu tezin yazıldığı sırada mevcut olan en son OpenStack sürümünde güç minimizasyon hedefleriyle ILP tabanlı bir bulut kaynak tahsisi yöntemini entegre ediyoruz. OpenStack'in tanımlanmış API'lerine göre gerekli tüm mesajlaşmayı sağlayan ILP çözücüyü arayüzleyen bir yazılım mimarisi geliştiriyoruz. Genişletilmiş uygulamanın tamamı, bulut sunucuları, 40 Gbps ağ ve heterojen bir bulut veri merkezini temsil eden donanım hızlandırıcılara sahip FPGA kartları içeren bir laboratuvar kurulumunda gerçekleştirilir.

Anahtar Kelimeler: Bulut bilişim, Donanım hızlandırıcaları, Kaynak yönetimi, OpenStack, Nova

To my parents, Cafer and Leyla

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor Prof. Dr. Şenan Ece Güran Schmidt for her precious support and sincere efforts. This work has been made possible by her understanding and her leadership.

I would like to thank to my father Cafer, my mother Leyla, my elder brother Oğuz and his wife İrem for their priceless supports and faith in me.

I would like to thank ASELSAN for its technical material support and for its permission to attend the classes.

I also thank TÜBİTAK for their support with TÜBİTAK - BİDEB MSc scholarship.

I am thankful to my friends Cem Ali Kandemir, Ayhan Sefa Yıldız, Alper Yazar, Anıl Tırlıoğlu for their technical and non-technical supports throughout this study.

Finally, I would like to thank the entire ACCLOUD family for making this work fun, enjoyable and memorable.

This thesis was supported by the Scientific and Research Council of Turkey (TÜBİTAK) [Project Code 117E667-117E668].

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS	
AC	Algorithm Computer
ACCLOUD-MAN	ACCLOUD Manager
ACCLOUD	Accelerated Cloud
ACCMAN-INT	ACCLOUD Manager Interface
API	Application Programming Interface
ARM	Advanced RISC Machines
ARS	Automatic Request Sender
AWS	Amazon Web Services
CCTV	Closed-Circuit Television
CDC	Cloud Data Center
CompNC-1	Compute Node Computer - 1
CompNC-2	Compute Node Computer - 2
CtrlNC	Control Node Computer
DC	Data Center
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
HVAC	Heating, Ventilation, and Air Conditioning
IaaS	Infrastructure as a Service
ICD	Interface Control Document
ILP	Integer Linear Programming
IT	Information Technology
OS	Operating System

PaaS	Platform as a Service
PM	Physical Machine
RR	Reconfigurable Region
SaaS	Software as a Service
SoC	System on a Chip
TCP	Transmission Control Protocol
VM	Virtual Machine

CHAPTER 1

INTRODUCTION

Traditionally, computer hardware, software, application, and computation functions are managed by the users and in the environment where the users are located. In recent years, cloud computing is replacing this traditional approach to offer these functions as a service over the Internet in accordance with user demands. Cloud data centers (CDC) with thousands of servers and a high-speed networking infrastructure are established to offer such computing services. Heterogeneous hardware resources including GPUs or FPGAs are included in the CDCs for hardware acceleration to increase performance and decrease the power costs [1, 2].

It is important to note that *virtualization* is a key technology that enables using of hardware resources such as the number of processor cores, memory, and disk as independent and abstracted partitions. Virtualization is realized by adding an abstraction software called the *hypervisor* between the physical machine hardware and the operating system. To this end, in the *heterogeneous cloud computing architecture*, FPGA resources must be *virtualized* and presented to the user seamlessly with server resources.

Cloud Computing creates virtual machines on physical servers and offers them to the users as infrastructure as a service (IaaS). OpenStack [3] is an open-source platform for cloud-based data centers and private cloud systems to offer IaaS services. OpenStack maintains a database of available computing resources together with *virtual machine* (VM) configurations. It instantiates the IaaS requests by fetching VM images from a repository and instantiating them on physical servers on demand. Furthermore, OpenStack organizes the network structure together with address assignments to facilitate user access to the VMs over the Internet.

Cloud computing service providers offer different levels of abstraction and virtualization techniques and computing services with different features and scopes. In this context, the user's data is processed and the output is returned to the user, and all hardware and software resources used in this process are on the cloud side, called Software as a Service (SaaS).

An important problem for offering cloud services is the allocation of resources to user requests. The complexity of this problem increases when heterogeneous computing resources and different computing services are present. Here, the power consumption of CDC and the efficient use of CDC resources while providing the requirements of the services must be taken into consideration. SaaS requests are fulfilled under the control of the cloud service manager and independent of the user. In this case, hardware accelerators can also be used as less power-consuming alternatives for SaaS requests. OpenStack default resource management is carried out by a filter-based scheduler which places the IaaS request on the first available physical machine with enough resources without observing any optimization goals.

This thesis develops, implements, and evaluates an extension of OpenStack to address the resource management and allocation of heterogeneous cloud data centers to address the requirements that we list above. To this end, we integrate the ILP-based cloud resource allocation method in [4, 5] that we call ACCLOUD-MAN (ACcelerated CLOUD MANagement) in the most recent version of OpenStack that was available at the time of writing this thesis. ACCLOUD-MAN places the IaaS and SaaS requests that come from the users on the PMs with a power consumption minimization objective. It considers alternative VM compositions with hardware accelerators and selects the best alternative that fulfills the SaaS request requirements resulting in minimal power consumption.

We develop a software architecture that interfaces the ILP solver implementing ACCLOUD-MAN and OpenStack. This software provides all the required messaging according to the defined APIs of OpenStack. The interface to ACCLOUD-MAN implementation is over TCP sockets. Our software architecture can generate IaaS (VM) and SaaS requests according to a trace file for experimental evaluation under different request scenarios. The entire extended implementation is realized in a laboratory set-up that features cloud servers, a 40 Gbps network, and FPGA cards with hardware accelerators that represent a heterogeneous cloud data center. We conducted functional correctness tests. We demonstrate the decision capabilities of ACCLOUD-MAN by custom-designed request arrival scenarios. Furthermore, we evaluate the performance of the extended OpenStack developed in this thesis regarding heterogeneous VM instantiation reaction time. Our previous work in [6, 7] implemented and evaluated OpenStack on a single physical machine which was first virtualized as cloud physical machines then virtualized a second time by OpenStack to provide heterogeneous VMs. ACCLOUD-MAN was implemented integrated with a simulation and evaluated entirely by this simulation in [4, 5].

This thesis updates and extends our previous work in [4, 5, 6, 7] to integrate ACCLOUD-MAN in OpenStack and evaluate the entire implementation on a real cloud installation in the laboratory. It is important to note that OpenStack focuses on providing IaaS. Within the scope of this thesis, we also demonstrate a possible method of realizing SaaS with OpenStack by pre-mapping SaaS requests to IaaS VM realizations.

The remainder of the thesis is organized as follows. Chapter 2 introduces the background about CDCs, ACCLOUD, OpenStack and ACCLOUD-MAN. In Chapter 3, improvements to previous studies and deployment of ACCLOUD project are covered. In Chapter 4, tests conducted on ACCLOUD is explained in detail. Chapter 5 concludes the thesis.

CHAPTER 2

BACKGROUND AND PREVIOUS WORK

2.1 Cloud Data Centers

Data centers (DCs), in general speaking, consist of computer systems, networking infrastructures, and physical infrastructures such as power supplies, generators, and HVACs. DCs take place in the center of IT operations of today's business areas. In DCs, data are stored, processed, and accessed through a network which could be public or private. With such crucial roles of DCs, they are supposed to be cared for and kept running all the time with minimum failures as possible. In order to achieve always-on operation for DCs, they are equipped with security measures such as fences, guards, CCTVs to keep them safe from physical attacks and with software to keep them from cyber attacks. Additional to security measures, maintenance, and other IT operations such as environmental controls, server rack maintenance, power supply are required. To this end, cloud data centers (CDCs) come forward [8] as an easier way to meet these requirements.

CDCs operate very similar to DCs, there are several advantages of CDCs nonetheless. The very first advantage of CDCs is that all the operations mentioned above are handled by cloud providers which totally abstracts the cloud user from them and allows limited or zero control of physical resources. Another advantage of CDCs is the effective use of physical resources [9]. Multiple tenants can use a certain physical resource with virtualization technology. Another advantage of CDCs is making physical resources available to the customer with a pay-as-you-go pricing policy [10]. Thus, there is no need for huge investments to run such DC. The last but not the last advantage is freeing IT staff. Less human resources lead to a cut in the prices of service given which could be very important for startups [8].

When all their advantages of CDCs are taken into account, 80% of DCs are expected to be closed by 2025 [11]. Thus, CDCs, therefore cloud computing, grow at an enormous speed and there is a tendency towards cloud computing. Some cloud computing service providers, thus CDC owners, can be named as Google Cloud, AWS, and Microsoft Azure [12, 13, 14].

2.1.1 Virtualization

Virtual Machines (VMs) are used as a standard in the cloud to provide highly available and flexible services (i.e., computing as a service) and due to the maturity of *virtualization* technology. Virtualization is accomplished by adding an abstraction layer (hypervisor) between the physical machine hardware and the operating system [15]. VMs separate the computing infrastructure from the physical infrastructure and allow the platform to be customized to the needs of the end-user. Virtualized servers are completely managed by software. The management software can create or remove a new virtualized server at any time without replacing or restarting physical servers. A virtualized server is similar to a computer program. Each virtualized server must run on one physical server, and multiple virtualized servers can run simultaneously on a given physical server in complete isolation. Thus, data and calculations performed by a virtualized server cannot be observed or influenced by another server. Virtual machines run operating systems independently of each other. These operating systems can access different hardware configurations. This way hardware resources can be used better.

Virtualization enables server consolidation throughout CDC. Server consolidation means that one physical resource may serve multiple tenants or namely customers. Therefore, an increase in the number of used physical resources is achieved. Additionally, power efficiency is also increased due to many tenants being hosted on less physical resources [16].

2.1.2 Physical Resources and CDC Types: Homogeneous vs. Heterogeneous

As stated in Section 2.1, physical resources in a cloud data center can be CPUs, memory, disk, or bandwidth. CDC that contains one kind of computing resource, such as CPU, is called a homogeneous CDC. Due to a decrease in power costs and increase in performance concerns, CDCs introduce FPGAs or GPUs as computing resources. This also provides flexibility both to customers and service providers. CDCs with more than one type of computing resources are called heterogeneous.

FPGAs exist in many levels of a CDC. For instance, FPGAs are used in the infrastructure of CDCs [17] and are also used as computing resources [6, 4]. This study focuses on FPGA as a computing resource, since FPGA-based switches are transparent to the tenants.

2.1.3 Cloud Services

[18, 10] Cloud service providers serves customers with their CDCs mainly in three ways: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These services differ from each other according to the control level of the customer over provided physical resources. This is difference between services is shown in Figure 2.1 [18].

Traditional IT cases can be considered exactly owning all physical resources and maintaining the whole infrastructure.

Typically, IaaS companies also provide servers, network equipment, and basic data storage facilities (e.g., block storage on disk). An IaaS company can offer customers many additional services such as load balancers, data backup, network security, a way to start both physical and virtualized servers, and assignment of Internet addresses. An IaaS customer does not need to manage or control cloud infrastructure. The customer can choose which operating systems and applications will run and have the ability to control network access (for example, configuring a firewall). IaaS companies use operating systems that can scale up or down the customer's services and customer-dedicated facilities as needs change. For OS level management, there

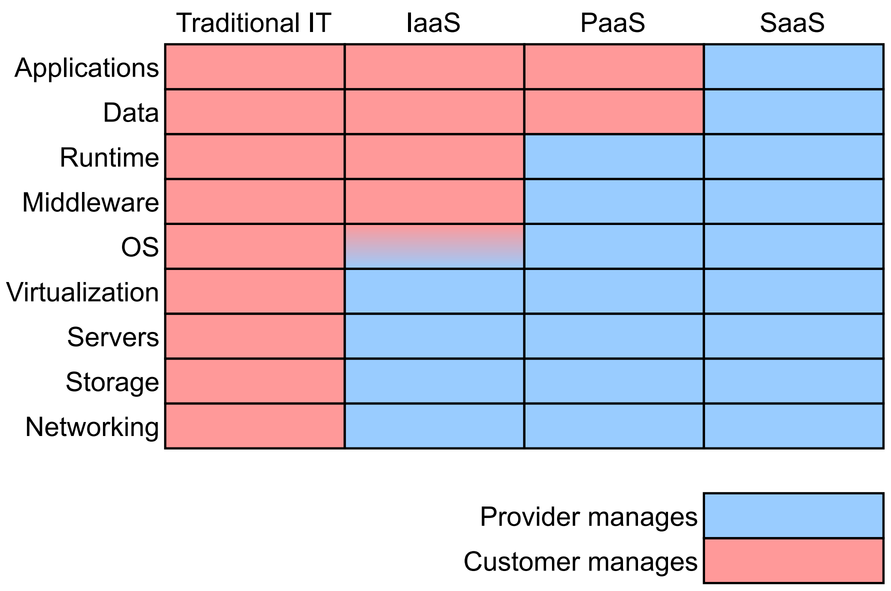


Figure 2.1: IaaS, PaaS and SaaS

are different approaches. According to IBM, the OS level is managed by the service provider [18]. On the other hand, according to Azure, OS is managed by the customer in IaaS [19]. That is why there is a transition in the OS layer for IaaS in Figure 2.1. In order to use IaaS, the customer is needed to specify exactly the kind of and amount of physical resources requested. For example, AWS and Rackspace provide IaaS to customers.

PaaS offers the customer less control over the physical resources when compared to IaaS. Nevertheless, PaaS allows the customer to use previously prepared environments without restrictions on the use of the virtual machine. As in the IaaS case, again the customer is needed to provide physical resource specifications to use PaaS. Google App Engine can be given example as a PaaS provider.

SaaS is the software that is ready-to-use from the perspective of the customer. The customer has no control or information about how the software works. Dropbox is a well-known example of SaaS.

2.1.4 Cloud Data Center Resource Management

Assigning computational resources to incoming requests in cloud computing systems is an NP-hard problem [2, 5, 4, 20]. On the other hand, request assignment formulations based on Constraint and Utility Function can use resources more efficiently or take care of the service quality that the user receives. In this case, the utility function can be for the system (for the benefit of the cloud provider) or for the user. Optimization purposes can be resource pricing, resource efficiency, energy, availability, service quality [20, 21]. [4, 20] propose a novel resource allocation method for heterogeneous cloud data centers with a goal of minimizing power consumption. Different than all previous work, this work exploits the possibility of realizing the SaaS requests of the users with different VM configurations *alternatives* that can incorporate hardware accelerators. The extension of OpenStack in this thesis implements this method. To this end, we provide the detailed formulation of this method in Section 2.3.

2.2 OpenStack

The abstraction, virtualization, and distribution of cloud resources to users involve many sub-problems. For this purpose, cloud computing control platforms are developed [22]. These platforms act as an operating system and offer cloud resources to users. Openstack [3] is such a platform used in cloud-based data centers and private cloud systems. OpenStack is an open-source software project that manages and coordinates the elements in the cloud architecture and also offers many capabilities for the cloud architecture to the programmer. OpenStack provides management and control of resources, such as computing, storage, networking, in data centers [23]. OpenStack manages and coordinates the elements in the cloud architecture and also offers many capabilities for both developer and user [24]. By default, it provides IaaS for public and private clouds.

OpenStack stands out as preferable cloud architecture since it is open source and provides flexibility to the developer. OpenStack is a popular platform preferred for adoption by 36% of the users [25]. Considering that CDCs still need large amounts of investment and have issues such as high power consumption, the use of OpenStack, which can be easily modified, becomes advantageous [26]. [27, 28] adopt OpenStack for implementing their resource allocation algorithms to reduce power and resource consumption. In addition, OpenStack architecture is used within the scope of development and improvement of FPGA introduced heterogeneous cloud systems [29].

OpenStack consists of many services that are called "projects". These services communicate with each other through APIs and with these services OpenStack does not only provide IaaS but fault management and service management other services to meet QoS. OpenStack projects have the plug-and-play feature such that a service can be deployed or removed at any time [3].

2.2.1 OpenStack Projects

As stated in Chapter 2.2, there are many projects used in OpenStack. However, some of them are required for minimal deployment. These projects are listed as follows: Keystone, Glance, Neutron, Placement, Nova.

- *Keystone* project is used as an authorization service in almost all of the OpenStack architecture. In order to make any API call within the OpenStack architecture, a valid token is taken from the Keystone service, and the validity of the relevant token is checked at the place where the API call is received.
- *Glance* project is a project that manages the image files of the virtual machines to be created. Virtual machines to be created must be initialized according to the relevant image and this capability is provided by Glance.
- *Neutron* project provides network management in OpenStack architecture. It can create virtual networks on top of physical networks for created virtual machines.
- *Placement* project is a project responsible for resource management of the OpenStack architecture. In the OpenStack Pike version used at the beginning of the project, this project did not exist, and resource management was carried out

in Nova. Recognition of heterogeneous physical resources such as Placement project FPGA and GPU in OpenStack Wallaby version allows different physical resources to be used.

- *Nova* is responsible for the creation and maintenance of virtual machines. It also provides an API that allows incoming virtual machine requests in new Nova versions to be initiated on explicitly requested servers without forwarding them to Nova Scheduler.

2.2.2 OpenStack Virtual Machine Creation

OpenStack is consisted of two main parts: controller node and compute node. Controller node OpenStack projects Keystone, Glance, and Placement run only on controller node. Neutron and Nova run on both nodes [30]. As it is understood, controller node could be responsible for many compute nodes. Compute nodes are responsible for VM deployment processes. OpenStack architecture with projects provided above is shown in Figure 2.2.

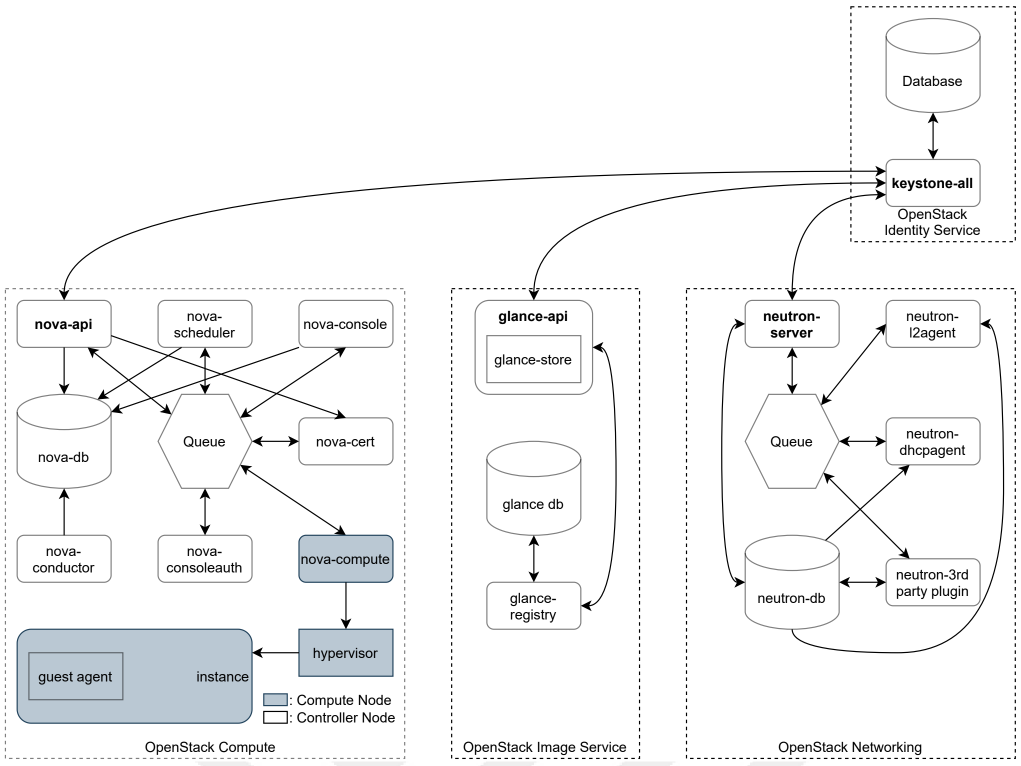


Figure 2.2: OpenStack Architecture

Assuming an authenticated VM creation request message is received by `nova-api`, simple VM deployment process occurs as follows. As first thing, `nova-api` checks whether received request is authenticated with valid Keystone token. `nova-api` sends received token to `keystone-all` and validates the token. `nova-api` sends VM info to `Queue` to queue VM creation request. `nova-scheduler` consumes VM creation request from `Queue`. `nova-scheduler` obtains current states of `nova-compute`'s from `nova-db`. Later,
`nova-scheduler` picks a `nova-compute` instance to deploy and pushes VM creation with selected `nova-compute` information to `Queue`. `nova-compute` consumes VM creation request from `Queue`. At this point `nova-compute` starts to allocate physical resources. `nova-compute` requests network configurations from `neutron-api`. `nova-compute` successively requests requested image from `glance-api`. After required components are allocated, `nova-compute` creates VM using `hypervisor`. `nova-compute` updates `nova-db` after deployment process and `nova-api` returns a successful result to VM creation message with

created VM information including unique server ID.

2.2.3 OpenStack Nova Scheduler

OpenStack provides `nova-scheduler` service [31] to determine on which host or node an IaaS VM request should launch. The default configuration of the `nova-scheduler` is filter-based. It considers the physical machines that meet the criteria of the request including their available resources and their organization as a group which can constrain the VM instantiation. The VM is instantiated on one of the Physical machines that pass through the filter. In the recent versions of OpenStack the Nova APIs support explicitly specifying the host to start the VM [32]. To this end, it is possible to bypass Nova Scheduler and implement an external resource allocation mechanism as we propose in this thesis.

2.2.4 OpenStack Implementation

OpenStack software runs on Linux machines. OpenStack can be installed different Linux distributions [30, 33].

In the OpenStack architecture, there are two main actors which are called "Controller Node" and "Compute Node". These nodes are installed in separate PMs. Whereas there is on Controller Node, there could be several Compute Nodes. Controller Node, briefly, runs management processes and is responsible for the orchestration of the whole OpenStack architecture. Compute Node(s) contain the hypervisors.

A component of Nova, namely Nova-Compute, and Neutron is running on the compute node. According to the orders from the Nova-Controller in the controller node, the relevant physical resources are assigned for the creation of virtual machines. Virtual machines are created and initialized according to their respective Glance images.

OpenStack deployment process includes installation and configuration. Firstly, the Environment configuration needs to be done, which is listed as follows [34].

- *Networking configuration*
- *NTP configuration*
- *OpenStack client installation*
- *SQL database installation*
- *Message queue installation*
- *Memcached and etcd installation*

After environment installation and configuration are completed, OpenStack projects can be installed. In this study, OpenStack version Wallaby is installed. Minimal deployment for OpenStack Wallaby is given in the following list with installation order [35].

- Identity service: *Keystone*
- Image service: *Glance*
- Placement service: *Placement*
- Compute service: *Nova*
- Networking service: *Neutron*

2.2.5 An example deployment of OpenStack in Heterogeneous CDCs: ACCLOUD Framework

“ACCLOUD (ACcelerated Cloud)" is a novel, FPGA-Accelerated Cloud Architecture that offers users hardware accelerators implemented on FPGA as computing resources with an innovative cloud computing approach. To this end, FPGA resources are virtualized as re-configurable regions (RRs) and provided through OpenStack. The OpenStack module in the SoC processor on the FPGA is implemented as embedded software that works with other OpenStack modules. An accelerator image selected by the user can be written on the RR. A novel ILP-based resource allocation method called ACCLOUD-MAN is proposed which considers accelerators as alternative implementations for SaaS requests and allocates resources with minimum power consumption. The entire architecture is implemented in the laboratory environment with cloud servers, FPGA cards and 40 Gbps Ethernet switch for functional and performance tests [36].

Nova-G is a light Nova Compute project that is developed within the scope of ACCLOUD project. Nova-G is compatible with other OpenStack services and can allocate FPGA physical resources to meet customer requests [6, 7].

Nova-G was developed at the beginning of the ACCLOUD project compatible with OpenStack Pike (OpenStack version 11). Pike version is no longer supported by the community in this rapidly developing architecture. The final implementation and experiments of ACCLOUD are completed with the latest version of Wallaby (OpenStack version 18) which features the new Placement project with the ability to recognize and use heterogeneous resources, in fact, FPGA regions. To this end, Nova-G running on the FPGA SoC ARM processor was modified to be compatible with the Wallaby Version.

2.3 ACCLOUD-MAN

ACCLOUD-MAN is a resource allocation algorithm developed for heterogeneous cloud architectures in the scope of ACCLOUD framework. Users can request resources from the CDC in two ways. For IaaS / PaaS (IPaaS) requests, the user explicitly specifies the amount of resources requested for each resource type. For SaaS requests, the user simply specifies the desired cloud application. The amount of resources required for such a SaaS request is determined by ACCLOUD-MAN, considering that there may be more than one resource alternative to fulfill a SaaS request. For example, one (or more) processor cores or one (or more) FPGA regions can be assigned to run an application to compress a 1 GB file.

2.3.1 Problem Formulation

ACCLOUD-MAN aims to allocate IPaaS and SaaS requests on PMs with minimum power consumption objective following a simple linear power consumption model [4] as given in (2.3.1) where P_{newPM} is the power of newly switched on servers and P_{comp} is the power used by newly assigned compute resource, namely CPU or FPGA.

$$\min \quad F = P_{\text{newPM}} + P_{\text{comp}}. \tag{2.3.1}$$

To this end, ACCLOUD-MAN avoids turning on new servers as long as resources are available on the on servers [4]. From (2.3.2) to (2.3.6) show that there has to be enough CPU, FPGA, Memory, Disk, and Bandwidth with given order to allocate requests. C_i, F_i, M_i, D_i, B_i show the resources CPU, FPGA, Memory, Disk, and Bandwidth in the cloud, respectively. $s_{i,j,k}$ is 1 if the request is allocated on a PM, 0 otherwise. $c_{j,k}$ (CPU), $f_{j,k}$ (FPGA), $m_{j,k}$ (Memory), $d_{j,k}$ (Disk), and $b_{j,k}$ (Bandwidth) shows the allocated resources for every decision. After decision is completed, the algorithm must end up with a unique PM to allocate the request (2.3.7).

$$\sum_{j=1}^{K}\sum_{k=1}^{n_j} c_{j,k}s_{i,j,k} \leq C_i \;\; \forall i \in PM \tag{2.3.2}$$

$$\sum_{j=1}^{K}\sum_{k=1}^{n_j} f_{j,k}s_{i,j,k} \leq F_i \;\; \forall i \in PM \tag{2.3.3}$$

$$\sum_{j=1}^{K}\sum_{k=1}^{n_j} m_{j,k}s_{i,j,k} \leq M_i \;\; \forall i \in PM \tag{2.3.4}$$

$$\sum_{j=1}^{K}\sum_{k=1}^{n_j} d_{j,k}s_{i,j,k} \leq D_i \;\; \forall i \in PM \tag{2.3.5}$$

$$\sum_{j=1}^{K}\sum_{k=1}^{n_j} b_{j,k}s_{i,j,k} \leq B_i \;\; \forall i \in PM \tag{2.3.6}$$

$$\sum_{i=1}^{N}\sum_{k=1}^{n_j} s_{i,j,k} = 1 \ \ \forall \mathbb{REQ}_j \in \mathbb{REQ}. \tag{2.3.7}$$

Regarding IaaS requests (Virtual Machines-VMs), ACCLOUD-MAN aims for placing the requests according to this objective constrained by the available resources of the physical machines. Regarding SaaS requests, ACCLOUD-MAN selects an assignment from among the previously known resource alternatives. The difference that the user observes between the alternatives is the time to finish the job. It is assumed that a preliminary study and different types of input profiles are available for SaaS software running in the cloud and alternatives are kept in a database by the service provider according to these profiles.

ACCLOUD-MAN is formulated as an Integer Linear Programming (ILP) problem and solved using TOMLAB CPLEX solver [4].

ACCLOUD-MAN was evaluated by an event-driven MATLAB simulation which takes a request trace file as input. There was no interaction with OpenStack and all the resource allocation and release together with ACCLOUD-MAN decisions were simulated.

2.3.2 Simulation realization

The request trace information includes the number of the request, the arrival time of the request, the service period of the request, and the information of the resources requested in the request for each request message. According to the arrival and departure time of the request message read from each file, ACCLOUD-MAN was advancing its simulation time. Each request arrival and end time was treated as an event, and the simulation continued until all of these events were completed. This approach was done as follows.

First, the directory with the trace information was specified in ACCLOUD-MAN.

Then the physical resources were given manually as a simulation was run. Here, physical resources were randomly assigned within certain intervals. The purpose of doing this is to be able to test it under different test conditions during the development phase of the algorithm. Randomly generated physical resources are stored in ACCLOUD-MAN with the `stPM_Array` variable.

After the physical resources are determined for ACCLOUD-MAN, the algorithm software becomes able to process the requests. At this point, requests are received with the `getRequest()` operation. Requests are read in an endless loop. This infinite loop is broken when there are no events to consume, or when the calculations for an event time out. This approach is illustrated below.

Inside the `getRequest()` operation, a new line is read from the file. Each newly read line includes the arrival time of the requests and the time that the requests stay in the cloud system, as stated before. Based on these times, this operation is created two events and is stored in order from smallest to largest according to the relevant time (the events that are closer to the current simulation time in time come first).

Each line read was analyzed and made into meaningful requests for ACCLOUD-MAN. After this stage, ACCLOUD-MAN's solution creation steps were coming and it was finding suitable solutions in the randomly generated physical resource list for incoming requests.

This approach was more than sufficient for ACCLOUD-MAN's development processes and previous work. However, in the work done in the test environment for the final product, the requests come from the file in accordance with real-time, not simulation time. Due to this working difference, it was necessary to make updates in the way requests are received in ACCLOUD-MAN.

CHAPTER 3

REALIZING ACCLOUD-MAN IN THE CLOUD DATA CENTER: IMPLEMENTATION AND INTEGRATION WITH OPENSTACK

3.1 ACCLOUD-MAN Adaptations for Real Implementation

ACCLOUD-MAN is developed and tested *integrated with the event-queue based simulation environment* [4]. To this end, ACCLOUD-MAN is not compatible with OpenStack deployed on PMs and real-time events.

Therefore, in the scope of this thesis, ACCLOUD-MAN is extended so that it becomes compatible with real-time scenarios. In Section 2.3.2, ACCLOUD-MAN is introduced before modification. In Section 3.3, applied modifications are explained.

ACCLOUD-MAN is extended with a TCP socket interface for receiving and transmitting data. On one socket, it receives PM (Physical Machine) specifications, resource allocation algorithm parameters, and requests through a socket and after processing requests it transmits decisions again over a different socket. ACCLOUD-MAN is also separated from simulation event handler approach so it can work with request events not in simulation-time but in real-time. Modified ACCLOUD-MAN structure is provided in Figure 3.1. ILP Solver parts are kept intact during these modifications.

As shown in Figure 3.1, two sockets are introduced to ACCLOUD-MAN Controller Interface. OpenStack interface socket (`Socket_OpenStack`) carries OpenStack related information such as information about PMs, algorithm parameters (shown as 1) and algorithm allocation decisions (shown as 3). Request interface socket (`Socket_Request`) is dedicated to incoming request information (shown as 2) so

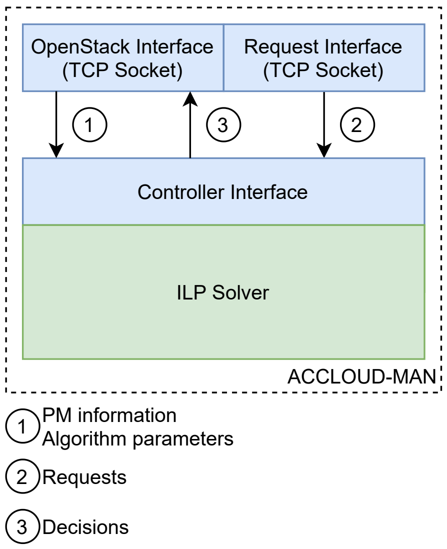

OpenStack Interface (TCP Socket)
Request Interface (TCP Socket)
1
3
2
Controller Interface
ILP Solver
ACCLOUD-MAN
1 PM information Algorithm parameters
2 Requests
3 Decisions

Figure 3.1: Modified ACCLOUD-MAN block diagram

that ACCLOUD-MAN could run and provide a suitable decision. `Socket_OpenStack` is configured as server and `Socket_Request` as client. Actors that interact with these sockets and how they work are provided in the following sections.

Modified ACCLOUD-MAN receives

- Number of PMs
- Total and free CPU cores
- Total and free FPGA regions
- Total and free memory size
- Total and free disk size
- Total and free network bandwidth

through `Socket_Request` and stores this information in `PM_Array` variable. After every decision is made, resources are deducted from `PM_Array` accordingly. Likewise, after every request departure, freed resources are added to `PM_Array` so that available resources are tracked on `PM_Array`. The default values of resource allocation algorithm parameters are available in ACCLOUD-MAN

Algorithm. Modified ACCLOUD-MAN can receive algorithm parameters through `Socket_Request`. In the case of not receiving any algorithm parameters message, ACCLOUD-MAN uses the default values.

After modified ACCLOUD-MAN gathers all information about PMs, it can respond to incoming requests. Requests are firstly processed whether it is an IaaS or SaaS request. IaaS requests are directly passed to decision stages. However, SaaS requests are assessed with an alternatives table. SaaS requests can be served with more than one alternative so ACCLOUD-MAN finds the best fit in terms of resource allocation and power usage. An example alternatives list for SaaS jobs listed in [37] is provided in Appendix A.

With these modifications, ACCLOUD-MAN is freed from reading trace data from a file and it is modified so that it becomes compatible with real-time scenario runs. It is possible to monitor up-to-date physical resource information not by removing the resources used in the decisions made, but by sending a message that queries the state of the PMs before each decision and querying it from the OpenStack architecture. This approach is not preferred as it will increase the messages to be sent over ACCMAN-INT and REST-API and cause delays before the decision. A hybrid solution to increase the reliability of the implementation against possible errors can be periodically sending state queries to the PMs and comparing their resource use state to verify the resource status maintained by ACCLOUD-MAN.

3.2 OpenStack Implementation and OpenStack Versions

OpenStack is the cloud resource management platform in the ACCLOUD framework. The initial OpenStack version for this thesis work was the Pike version of OpenStack that was the first deployment in ACCLOUD. However, Pike (OpenStack version 11) version is no longer supported by the community in this rapidly developing architecture. To this end, the final implementation and experiments in this thesis are completed with the latest version of Wallaby (OpenStack version 18) to include new capabilities added on top of the Pike version. OpenStack software runs in a distributed fashion on the controller and compute nodes. Accordingly, the OpenStack controller node is responsible for the messaging infrastructure, database management, and NTP services. Computing nodes are also responsible for providing cloud services to the user by communicating with the controller node. OpenStack consists of sub-software components called projects. We summarize below the installation steps of the current OpenStack version we are using.

In the ACCLOUD test environment, there are four workstations. One is used as a DHCP server and for other infrastructural purposes. Another one is used as a controller node and the remaining two are used as compute nodes. These workstations are connected with each other using 40Gbps network and a Mellanox router. This topology is shown in Figure 3.2.

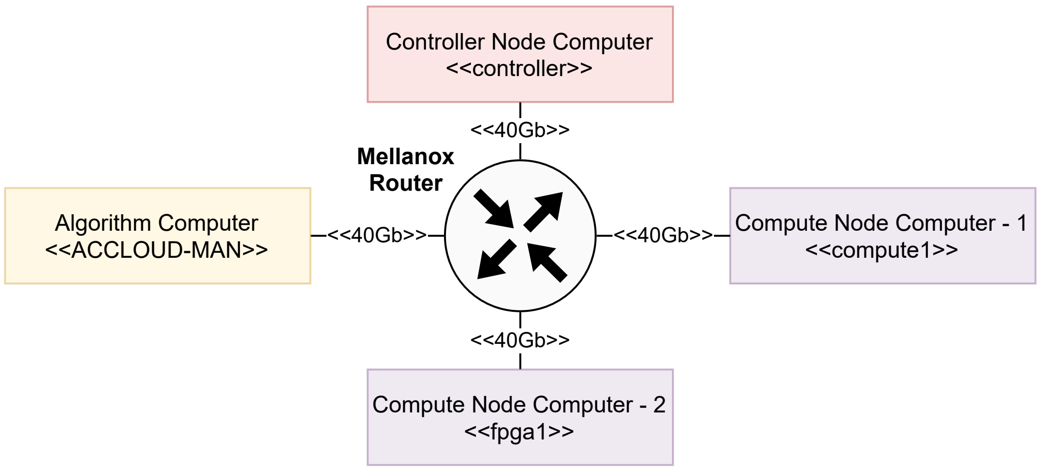


Figure 3.2: ACCLOUD: Hardware Block Diagram

Since OpenStack version Wallaby is used, Ubuntu LTS 20.04 OS is installed on all machines [38]. Firstly, network configuration is done for compute node and controller node as defined in OpenStack installation guide [39]. After that, OpenStack Wallaby is installed on all controller and compute nodes [40]. Additional applications that are used in OpenStack architecture are installed on controller node. List of the applications is provided below.

- SQL database [41]
- Message queue [42]
- Memcached [43]

- Etcd [44]

Now, OpenStack required projects for minimal deployment can be installed. Following projects are installed in the given order.

- ***Keystone:*** Project is installed and verified on controller node according to information given in the installation guide [45].
- ***Glance:*** Project is installed and verified on controller node with given information in [46].
- ***Placement:*** Project is installed on controller node using installation steps provided in [47].
- ***Nova:*** Nova is installed on controller and all compute nodes as stated in [48, 49].
- ***Neutron:*** Neutron is installed on controller and all compute nodes as stated in [50, 51]. Note that, network option #1 is installed.

Horizon project, or dashboard project, provides user interface for OpenStack administrative operations, is not installed during deployment since another application is developed, ACCMAN-INT, to meet and improve Horizon features.

3.3 ACCLOUD-MAN OpenStack Integration

In the ACCLOUD project, tenant requests are required to be processed by ACCLOUD-MAN, and the request can be served if the algorithm finds a suitable solution. A proxy software, ACCMAN-INT (ACCLOUD-MAN INTerface), is developed to establish communication between ACCLOUD-MAN and OpenStack. Apart from that, another software is developed, ARS (Automatic Request Sender), to run request scenarios in real-time. Additionally, an interface control document (ICD) for ACCLOUD-MAN is designed so that ACCLOUD-MAN, ACCMAN-INT, and ARS can communicate accordingly with the message format as defined in the ICD. Details of ACCMAN-INT and ARS are given in Chapters 3.3.1 and 3.3.2.

3.3.1 ACCMAN-INT Software

The OpenStack project Horizon provides a web user interface for virtual machine creation and it is a non-mandatory capability for the OpenStack architecture. Furthermore, Horizon has its own socket interfaces that cannot be intercepted. As the focus of this thesis is the integration of the ACCLOUD-MAN resource management with OpenStack, ACCMAN-INT software is designed and implemented between ACCLOUD-MAN and OpenStack, which has the same capabilities as Horizon except for the user interface. In this way, full competence is provided on the functions and connection flow of this interface. ACCMAN-INT software is developed from the ground up in a similar approach to Nova-G development, taking advantage of OpenStack's open-source software, in line with the documentation and messaging interface. ACCMAN-INT uses the REST API interface to obtain information about physical resources and to create or delete virtual machines.

ACCMAN-INT is developed using Python and is running on the controller node computer. The purpose of this software is to discover the physical resources in the OpenStack infrastructure, to forward them to ACCLOUD-MAN, and to send incoming customer requests (creating and deleting virtual machines) to OpenStack software.

After ACCMAN-INT establishes a TCP socket connection with ACCLOUD-MAN, it shares the resource information obtained from OpenStack with ACCLOUD-MAN. In this way, ACCLOUD-MAN can use up-to-date, available resource information while making decisions for allocations. ACCMAN-INT forwards incoming resource requests to ACCLOUD-MAN.

ACCLOUD-MAN transmits allocation decisions to ACCMAN-INT. According to the content of this decision message, ACCMAN-INT sends virtual machine creation messages to related OpenStack projects using REST API. The related message is received by Nova-Compute and then the virtual machine is created.

3.3.2 ARS Software

ARS software, which sends automatic requests, is developed to facilitate the tests by running the relevant tests automatically in the ACCLOUD test environment and repeating the simulation studies under the same conditions in the test environment. This software was developed in C++ and runs as a Windows application. This software simulates customer virtual machine creation and deletion requests from a given file in a certain format, in real-time as the Windows operating system allows.

In the TCP socket connection between ARS and ACCLOUD-MAN, ARS is defined as the server. In the current installation, ARS and ACCLOUD-MAN are running on the same computer, but this is not a requirement as a TCP socket connection is used.

When the user provides the relevant port number and the directory of the file with the requests to the ARS software, the software starts to run. The software creates and stores virtual machine creation and deletion messages by going over the entire file containing the requests. It then sorts these messages by comparing them by the time they should be sent. It then sends these messages to ACCLOUD-MAN following the relevant time stamps.

ACCLOUD-MAN awaits incoming request messages with the message ID `REQUST`, and leaving requests with the message ID `DEPART`. Details of these messages are explained in the following part of this document. In this way, customer requests are automatically simulated. At the same time, the scenario run in the simulation environment can be realized here as well.

3.3.3 ACCLOUD-MAN Interface Control Document

Interface Control Document (ICD) is a document that defines the interface of a system, or in this case an application. Since ACCLOUD-MAN has interfaces with ARS and ACCMAN-INT, an ICD is prepared to design these communications in a more controlled way. Message list is given in Table 3.1.

Details and message structure are given in the following sections.

Table 3.1: ACCLOUD-MAN ICD: Message List

Message ID	**Description**	**Direction**
`PMSTAT`	Server parameters message	ACCMAN-INT → ACCLOUD-MAN
`DECPAR`	Algorithm parameters message	ACCMAN-INT → ACCLOUD-MAN
`REQUST`	Tenant request message	ARS → ACCLOUD-MAN
`DEPART`	Tenant departure message	ARS → ACCLOUD-MAN
`DECISN`	Allocaiton decision message	ACCLOUD-MAN → ACCMAN-INT

3.3.3.1 Message Format

Message format is given in Table 3.2.

Table 3.2: ACCLOUD-MAN ICD: Message Format

Message ID	:	Field1	=	Value1	,	...	FieldN	=	ValueN	;

3.3.3.2 Server Parameter Message (`PMSTAT`)

This message is used to share a (Server) PM's power consumption parameters, total and free resources. This message is sent to ACCLOUD-MAN for every PM located in the cloud server for every time system is powered on. Message fields are given in Table 3.3

Table 3.3: PMSTAT Message Detail

Field	Type	Description
PMCOUNT	int	Total number of PMs in CDC
PMID	int	Index of PM whose parameters are provided in this message instance
PIDLE	int	Idle power consumption of PM
PIDLEPERCORE	int	Idle power consumption of PM per CPU core
PCPUCORE	int	Power consumption CPU per core
PFPGAREG	int	Power consumption FPGA per region
TOTCPU	int	Total number of CPU cores
FRECPU	int	Free number of CPU cores
TOTFPGA	int	Total number of FPGA regions
FREFPGA	int	Free number of FPGA regions
TOTMEM	int	Total memory size
FREMEM	int	Free memory size
TOTDSK	float32	Total disk size
FREDSK	float32	Free disk size
TOTBWD	float32	Total network bandwidth
FREBWD	float32	Free network bandwidth

Message Example:

```
PMSTAT:PMCOUNT=4,PMID=1,PIDLE=2,PIDLEPERCORE=1,
PCPUCORE=65,PFPGAREG=2,TOTCPU=8,FRECPU=4,TOTFPGA=4,
FREFPGA=4,TOTMEM=256,FREMEM=128,TOTDSK=15000,
FREDSK=11500,TOTBWD=40,FREBWD=35;
```

3.3.3.3 Algorithm Parameter Message (**`DECPAR`**)

This message includes ACCLOUD-MAN algorithm decision parameters. When ACCLOUD systems start to operate this message could be sent to ACCLOUD-MAN. Otherwise, ACCLOUD-MAN will use the parameters with default values. Message details are given in Table 3.4. These parameters are used to increase the resource allocation efficiency and decrease the run time of the ILP formulation in ACCLOUD-MAN in CDCs with thousands of servers in [4, 5]. We use the default values of 1 for these parameters in the scope of this thesis.

Table 3.4: `DECPAR` Message Detail

Field	**Type**	**Description**
`GAMMA`	`int`	Algorithm bundle size
`M`	`int`	Number of assessed PM for each request

Message Example:
`DECPAR:GAMMA=3,M=4;`

3.3.3.4 Tenant Request Message (**`REQUST`**)

This message includes details of the tenant request. ACCLOUD-MAN receives this message and later tries to decide a proper allocation for the request. Message details are given in Table 3.5.

Table 3.5: `REQUST` Message Detail

Field	**Type**	**Description**
`REQID`	`int`	Tenant's ID
`ARRTIME`	`float32`	Tenant's arrival time
`DRTN`	`float32`	Tenant's requested service time
`SIZE`	`int`	Tenant's size information (obsolete)
`SWID`	`int`	Tenant's request type
`CPU`	`int`	Tenant's requested CPU
`FPGA`	`int`	Tenant's requested FPGA region
`MEM`	`int`	Tenant's requested memory space
`DSK`	`float32`	Tenant's requested disk space
`BWD`	`float32`	Tenant's network bandwidth

Message Example:

```
REQUST:REQID=2,ARRTIME=22.946,DRTN=5555.359,
SIZE=0,SWID=2,CPU=0,FPGA=0,MEM=0,DSK=0,BWD=0;
```

`SWID` could be 0, 1, 2, or 3. Corresponding meanings are given in the following list.

- `SWID = 0`: Request's service type is IaaS. `CPU`, `FPGA`, `MEM`, `DSK`, and `BWD` fields are going to be given ACCLOUD-MAN as input.
- `SWID = 1`: Request's service type is SaaS for social media purposes. `CPU`, `FPGA`, `MEM`, `DSK`, and `BWD` fields are going to be ignored.
- `SWID = 2`: Request's service type is SaaS for video streaming purposes. `CPU`, `FPGA`, `MEM`, `DSK`, and `BWD` fields are going to be ignored.
- `SWID = 3`: Request's service type is SaaS for collabrative working purposes. `CPU`, `FPGA`, `MEM`, `DSK`, and `BWD` fields are going to be ignored.

3.3.3.5 Tenant Departure Message (**DEPART**)

This message includes details of tenant departure. ACCLOUD-MAN receives this message and handles operations to flag as free the resources that are used by the tenant. Message details are given in Table 3.6.

Table 3.6: DEPART Message Detail

Field	Type	Description
REQID	int	Tenant's ID

Message Example:

```
DEPART:REQID=1;
```

3.3.3.6 ACCLOUD-MAN Decision Message (**DECISN**)

This message includes details of ACCLOUD-MAN algorithm output. Message details are given in Table 3.7.

Table 3.7: DECISN Message Detail

Field	Type	Description
DECCNT	int	Number of decided requests
REQID	int	ID of decided tenant's request
PMID	int	Decided PM index
SWID	int	Decided service index
CPU	int	Decided CPU
FPGA	int	Decided FPGA region
MEM	int	Decided memory
DSK	float32	Decided disk
BWD	float32	Decided network bandwidth

Message Example:

```
DECISN:DECCNT=1,REQID=2,SWID=1,CPU=2,FPGA=0,MEM=8,
DSK=60,BWD=1;
```

3.4 ACCLOUD-MAN Implementation in Laboratory Environment

An objective of this thesis work is to deploy a fully functional heterogeneous cloud data center architecture that uses ACCLOUD-MAN as a resource allocation method. The functional demonstration of this integration is carried out on a Laboratory set-up that implements the entire ACCLOUD framework.

To this end, ACCLOUD-MAN and OpenStack are required to communicate with each other while utilizing FPGA compute resources which are managed by Nova-G on each FPGA node.

In Figure 3.3, ACCLOUD software deployment scheme is given.

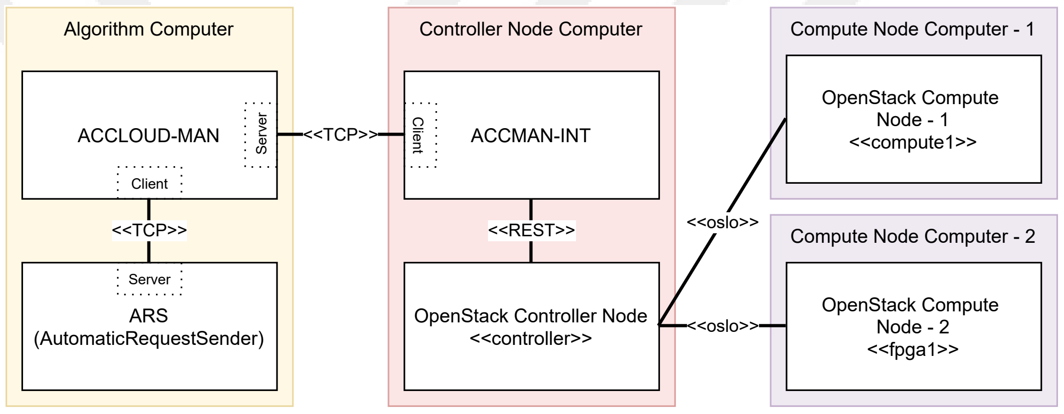


Figure 3.3: ACCLOUD: Software Block Diagram

As can be seen in Figure 3.3, there are five PMs in the ACCLOUD laboratory setup.

- Algorithm Computer (AC)
- Controller Node Computer (CtrlNC)
- Compute Node Computer-1 (CompNC-1)
- Compute Node Computer-2 (CompNC-2)

ACCLOUD-MAN software is developed on MATLAB and is deployed on AC. OpenStack controller services are installed on the CtrlNC. OpenStack compute services are installed on CompNC-1 and CompNC-2. These PMs are connected with 40G Ethernet cables. The hardware connection diagram is provided in Figure 3.2.

OpenStack CtrlNC, CompNC-1 and CompNC-2 are named as *controller*, *compute1* and *fpga1*, respectively. OpenStack services are introduced using these computer names.

Next, we explain the operation of the ACCLOUD-MAN integrated with OpenStack on the ACCLOUD laboratory set-up in detail.

AC, CtrlNC, CompNC-1, and CompNC-2 are powered on. ACCLOUD-MAN and ACCMAN-INT are run.

ACCMAN-INT requests a authorization token from controller node. ACCMAN-INT prepares message with payload (`auth_payload`). Prepared message is sent to Keystone project using REST API provided by OpenStack [52]. Note that, admin username and password are configured in installation steps. (Appendix B.1, Appendix B.2)

The authorization token is obtained from `X-Subject-Token` field in the response message. `X-Subject-Token` is stored in `auth_token` variable, which will be used to communicate with other OpenStack projects.

ACCMAN-INT obtains information about compute nodes' physical resources from Nova-controller. Firstly ACCMAN-INT requests PM list from Placement project and response is stored in `resource_list`. (Appendix B.3)

After then, ACCMAN-INT requests available PM list from Nova-compute project and it is stored in `status_nodes` variable. (Appendix B.4)

The physical resource information that can be used is obtained by checking the `resource_providers` information stored in the Nova-database under the `resource_list` data structure for each computing node in the ACCMAN-INT software, with the availability status kept in the status_nodes data structure. For each `resource_provider`, `inventory_item physical` resource list information is obtained. In addition, physical resource list information of `usage_item` is obtained for each `resource_provider`. In this way, necessary information is obtained for creating the`PMSTAT` message in the ACCMAN-INT software and sharing it with ACCLOUD-MAN. These procedures are shown below. ACCMAN-INT stores up and running Nova-compute list in `status_compute_nodes`. (Appendix B.5)

ACCMAN-INT obtains `inventory_item` for every suitable `resource_provider`. (Appendix B.6)

ACCMAN-INT could send decision parameters γ and M at this point with `DECPAR` message. If this message is not sent to ACCLOUD-MAN, it uses their default values.

After gathering all required information about, ACCMAN-INT prepares and sends `PMSTAT` message for every PM. (Appendix B.7)

ACCMAN-INT obtains flavor and image list using `auth_token`. ACCMAN-INT requests flavor list from Nova-compute [53] service and image list from Glance service [54]. They are stored in `flavor_list` and `image_list`, respectively. (Appendix B.8, Appendix B.9)

After ACCLOUD-MAN is received all `PMSTAT` messages, it waits for tenant request messages. For automatic test purposes, ARS software is run. ARS sends tenant request messages to ACCLOUD-MAN. ACCLOUD-MAN sends decision messages `DECISN` to ACCMAN-INT. ACCMAN-INT sends VM creation request for every received `DECISN` message. Incoming `DECISN` message is parsed and processed with `decisn_msg_handler` operation. (Appendix B.10)

Then, ACCMAN-INT prepares a VM creation request message using `auth_token` with `send_openstack_server_create_msg` [55]. (Appendix B.11)

When VM creation process is finished, Nova-compute sends a response message to the creation message. If the VM deployment is successful, the response message contains a unique server identifier, which is stored in `server_list`. After some time, tenant's departure message is received by ACCMAN-INT. Departure requests are processed in `depart_message_handler` operation. (Appendix B.12)

ACCMAN-INT obtains unique VM identifier from `server_list` and send VM deletion message to Nova-compute using `auth_token` [56]. ACCLOUD-MAN also releases allocated recources in its own database. (Appendix B.13)

Scenario explained in detail above is also shown as sequence diagram in Figure 3.4.

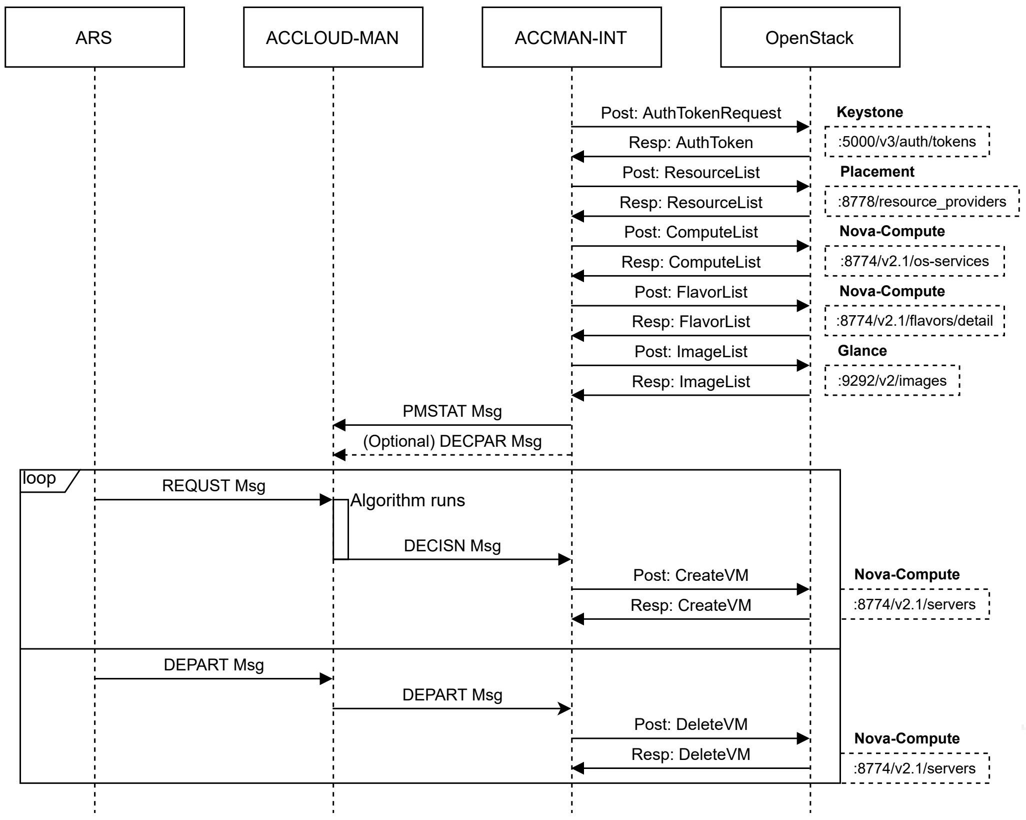


Figure 3.4: ACCLOUD: Sequence Diagram

CHAPTER 4

EVALUATION OF OPENSTACK EXTENDED WITH ACCLOUD-MAN

We evaluate our OpenStack implementation extended with ACCLOUD-MAN resource management method. To this end, we perform laboratory tests on the setup for the ACCLOUD architecture that is shown in Figure 4.1. We note that this set-up has all of the components of the architecture including the entire FPGA development with the hardware accelerators, an on-chip communication protocol, and an on-chip switch. The set-up includes a 40 Gbps switch for interconnecting the cloud nodes. To this end, our software development is tested in a very realistic heterogeneous cloud environment for functional correctness. Furthermore, we conduct performance tests on this realistic set-up. In Section 4.1, a scenario for functional correctness is defined and the scenario is tested on ACCLOUD Architecture setup (ACCLOUD Setup). In Section 4.2, ACCLOUD-MAN decision correctness is tested with a custom-designed scenario. In Section 4.3, we measure reaction time to serve incoming cloud requests by the extended OpenStack implementation. In Section 4.4, we observe the performance of setup and software over time.

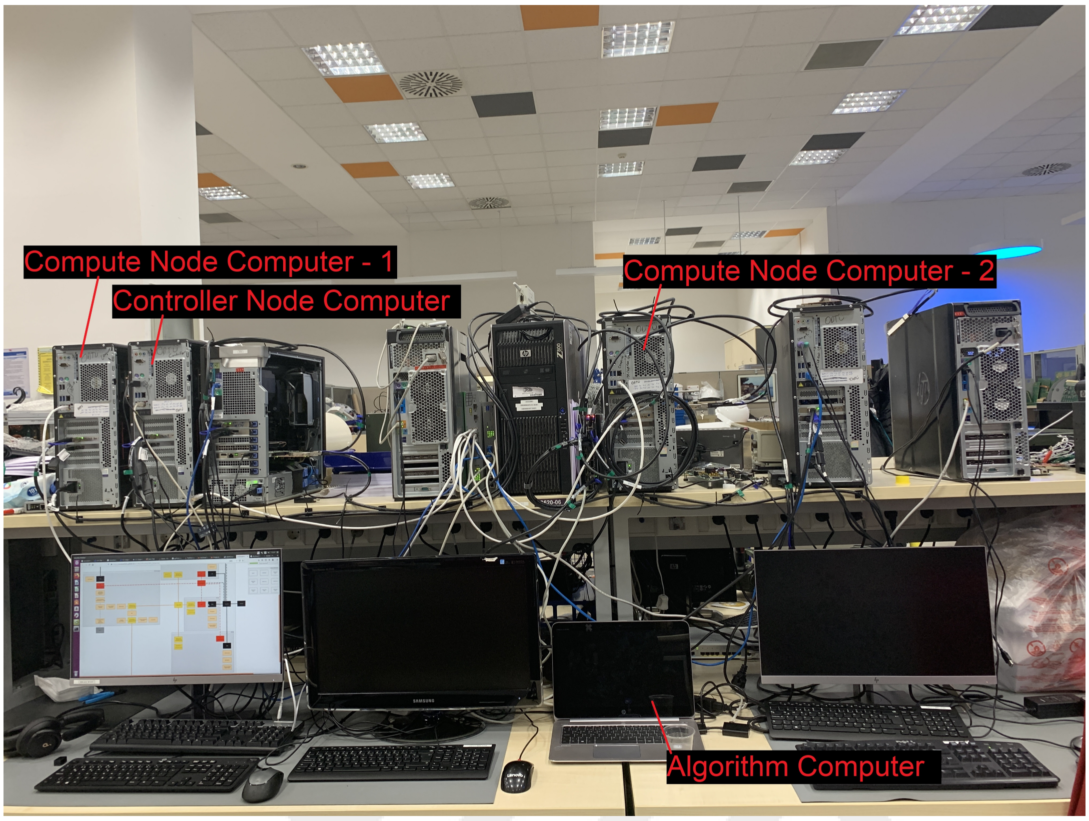


Figure 4.1: ACCLOUD-MAN waits ACCMAN-INT connection

4.1 Test - 1: Functional Correctness Test and Results

In this section, functional correctness for our OpenStack extension is tested on the ACCLOUD Setup. In ACCLOUD Architecture, different request types and request handling methods are possible. An incoming request could be either I(P)aaS or SaaS request. IaaS requests could be either for CPU or FPGA compute resource. Accordingly, a scenario is defined in Table 4.1. The scenario includes different types of service requests with different time of arrivals and departures aiming at all possible events that can occur in ACCLOUD deployment. Here a requested departure indicates the termination of the created Virtual Machine (VM).

The current ACCLOUD setup has two compute nodes, one with 192 virtual CPU cores available and the other one with 2 configurable FPGA regions available. *Customer #1* requests an IaaS with 1 virtual CPU core at time $t = 5$.

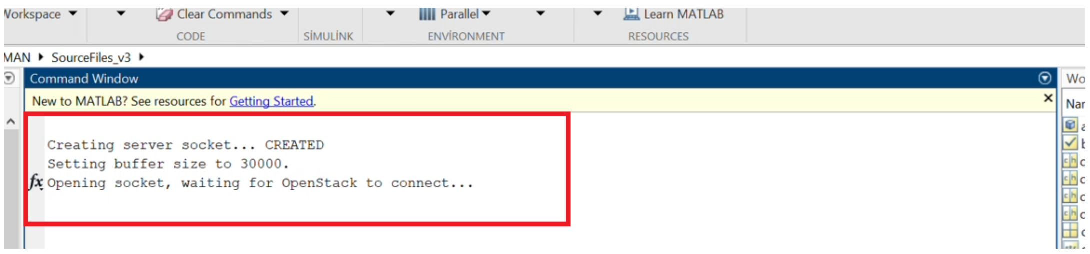
Table 4.1: ACCLOUD Functional Correctness Test Scenario

Customer #	**Arrival** (s)	**Duration** (s)	**Departure** (s)	**Service**	**Compute Resource Type**
1	5	90	95	IaaS	CPU
2	20	60	80	IaaS	FPGA
3	85	45	130	IaaS	CPU
4	100	45	145	SaaS	Type #1
5	110	50	160	IaaS	CPU

At $t = 10$, *Customer #2* arrives and requests a SaaS. Since there are available compute CPU and FPGA compute resources, The ACCLOUD-MAN method chooses the best alternative with minimum power concerns.

We use the alternatives table given in Appendix A for the SaaS requests as in [4]. Procedure given in Section 3.4 is used to complete this test. Each step is shown and explained in detail. "[]" in the figure captions indicates the software from which the screenshot was taken.

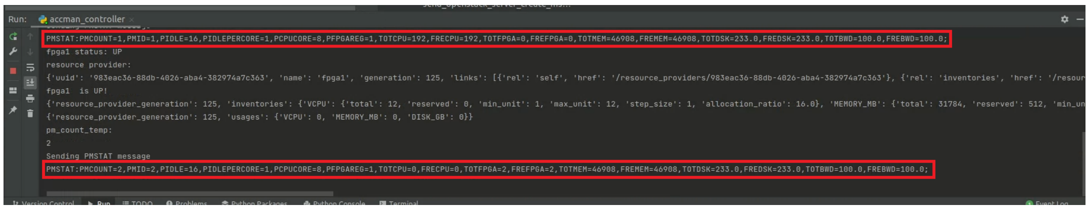
Firstly, ACCLOUD-MAN and ARS are run. After ACCLOUD-MAN initialization is completed, it waits for ACCMAN-INT to connect through `Socket_OpenStack`.



```
Creating server socket... CREATED
Setting buffer size to 30000.
Opening socket, waiting for OpenStack to connect...
```

Figure 4.2: [ACCLOUD-MAN] ACCLOUD-MAN waits ACCMAN-INT connection

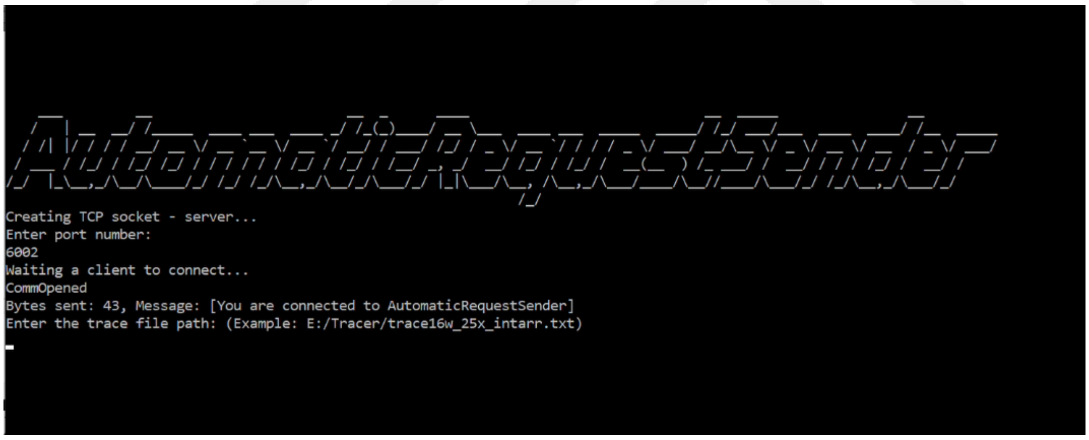
When ACCMAN-INT software is started, it establishes a connection with ACCLOUD-MAN and immediately starts to discover physical resources using OpenStack resource_providers API . When all the up and running PMs, images and flavors are discovered, ACCMAN-INT sends `PMSTAT` messages to ACCLOUD-MAN for each PM as shown in Figure 4.3.



```
PMSTAT:PMCOUNT=1,PMID=1,PIDLE=16,PIDLEPERCORE=1,PCPUCORE=8,PFPGAREG=1,TOTCPU=192,FRECPU=192,TOTFPGA=0,FREFPGA=0,TOTMEM=46908,FREMEM=46908,TOTDSK=233.0,FREDSK=233.0,TOTBWD=100.0,FREBWD=100.0;
fpga1 status: UP
resource provider:
{'uuid': '983eac36-88db-4026-aba4-382974a7c363', 'name': 'fpga1', 'generation': 125, 'links': [{'rel': 'self', 'href': '/resource_providers/983eac36-88db-4026-aba4-382974a7c363'}, {'rel': 'inventories', 'href': '/resour
fpga1  is UP!
{'resource_provider_generation': 125, 'inventories': {'VCPU': {'total': 12, 'reserved': 0, 'min_unit': 1, 'max_unit': 12, 'step_size': 1, 'allocation_ratio': 16.0}, 'MEMORY_MB': {'total': 31784, 'reserved': 512, 'min_un
{'resource_provider_generation': 125, 'usages': {'VCPU': 0, 'MEMORY_MB': 0, 'DISK_GB': 0}}
pm_count_temp:
2
Sending PMSTAT message
PMSTAT:PMCOUNT=2,PMID=2,PIDLE=16,PIDLEPERCORE=1,PCPUCORE=8,PFPGAREG=1,TOTCPU=0,FRECPU=0,TOTFPGA=2,FREFPGA=2,TOTMEM=46908,FREMEM=46908,TOTDSK=233.0,FREDSK=233.0,TOTBWD=100.0,FREBWD=100.0;
```

Figure 4.3: [ACCMAN-INT] PMSTAT messages are sent to ACCLOUD-MAN

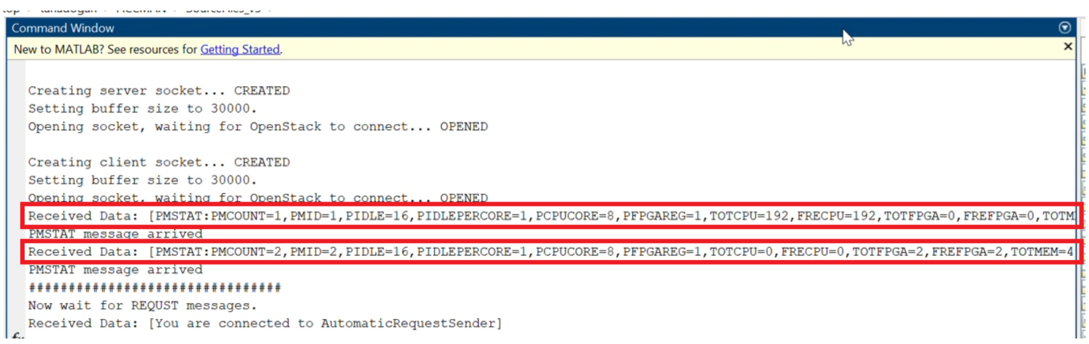
ARS software also establishes connection with ACCLOUD-MAN right after ACCMAN-INT connection is established with ACCLOUD-MAN. ARS software waits for user input to send REQUST and DEPART messages. User input is expected to be a trace file path, as shown in Figure 4.4.



```
Creating TCP socket - server...
Enter port number:
6002
Waiting a client to connect...
CommOpened
Bytes sent: 43, Message: [You are connected to AutomaticRequestSender]
Enter the trace file path: (Example: E:/Tracer/trace16w_25x_intarr.txt)
```

Figure 4.4: [ARS] ARS waits for trace path

It can be observed in Figure 4.5 that ACCLOUD-MAN receives PMSTAT messages and prints out that it is waiting for REQUST messages. This means that, ACCLOUD-MAN is successfully completed its initialization steps.

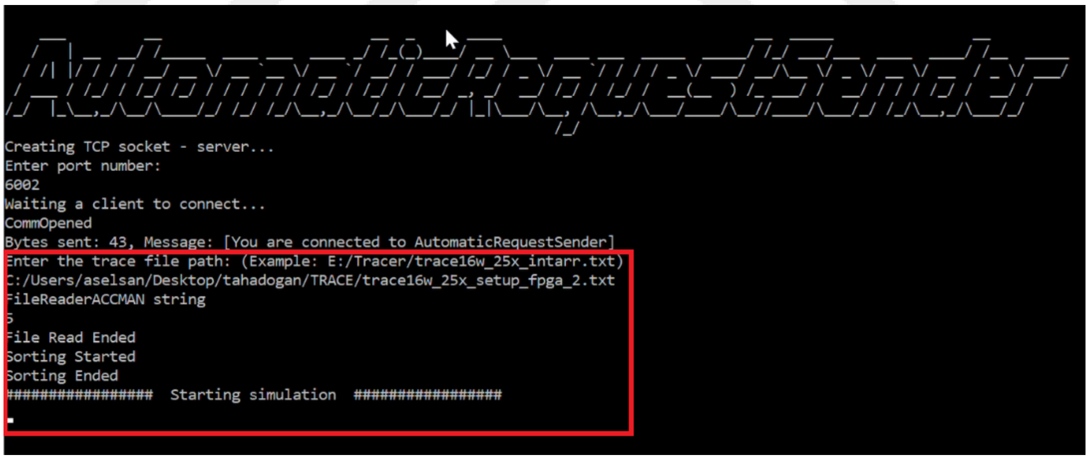


```
Creating server socket... CREATED
Setting buffer size to 30000.
Opening socket, waiting for OpenStack to connect... OPENED

Creating client socket... CREATED
Setting buffer size to 30000.
Opening socket, waiting for OpenStack to connect... OPENED
Received Data: [PMSTAT:PMCOUNT=1,PMID=1,PIDLE=16,PIDLEPERCORE=1,PCPUCORE=8,PFPGAREG=1,TOTCPU=192,FRECPU=192,TOTFPGA=0,FREFPGA=0,TOTM
PMSTAT message arrived
Received Data: [PMSTAT:PMCOUNT=2,PMID=2,PIDLE=16,PIDLEPERCORE=1,PCPUCORE=8,PFPGAREG=1,TOTCPU=0,FRECPU=0,TOTFPGA=2,FREFPGA=2,TOTMEM=4
PMSTAT message arrived
################################
Now wait for REQUST messages.
Received Data: [You are connected to AutomaticRequestSender]
```

Figure 4.5: [ACCLOUD-MAN] ACCLOUD-MAN receives PMSTAT messages

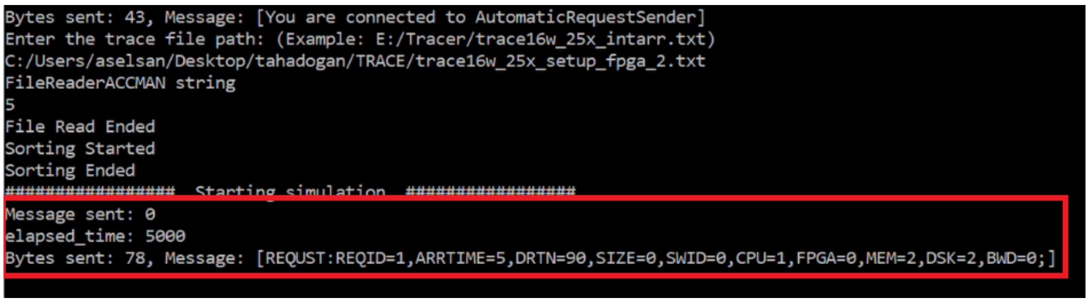
At this point, user can enter the trace file path any time to run scenario defined in the provided path. When the path is entered to ARS, it goes over all events and prepares its own event list. This can be observed in Figure 4.6. "Starting simulation" print means that ARS trace file reading and preparation is completed successfully and the scenario is about to start.



```
Creating TCP socket - server...
Enter port number:
6002
Waiting a client to connect...
CommOpened
Bytes sent: 43, Message: [You are connected to AutomaticRequestSender]
Enter the trace file path: (Example: E:/Tracer/trace16w_25x_intarr.txt)
C:/Users/aselsan/Desktop/tahadogan/TRACE/trace16w_25x_setup_fpga_2.txt
FileReaderACCMAN string
5
File Read Ended
Sorting Started
Sorting Ended
################  Starting simulation  ################
```

Figure 4.6: [ARS] ARS software starts to simulate real-time behaviors of customers

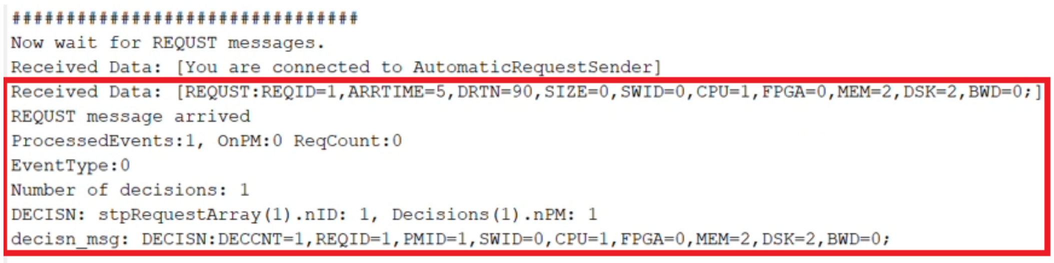
ARS transmits the first `REQUST` message to ACCLOUD-MAN when sending time has come after "starting simulation" print is printed. For every sent message, ARS prints out message details as shown in Figure 4.7.



```
Bytes sent: 43, Message: [You are connected to AutomaticRequestSender]
Enter the trace file path: (Example: E:/Tracer/trace16w_25x_intarr.txt)
C:/Users/aselsan/Desktop/tahadogan/TRACE/trace16w_25x_setup_fpga_2.txt
FileReaderACCMAN string
5
File Read Ended
Sorting Started
Sorting Ended
################  Starting simulation  ################
Message sent: 0
elapsed_time: 5000
Bytes sent: 78, Message: [REQUST:REQID=1,ARRTIME=5,DRTN=90,SIZE=0,SWID=0,CPU=1,FPGA=0,MEM=2,DSK=2,BWD=0;]
```

Figure 4.7: [ARS] ARS sends `REQUST` message to ACCLOUD-MAN

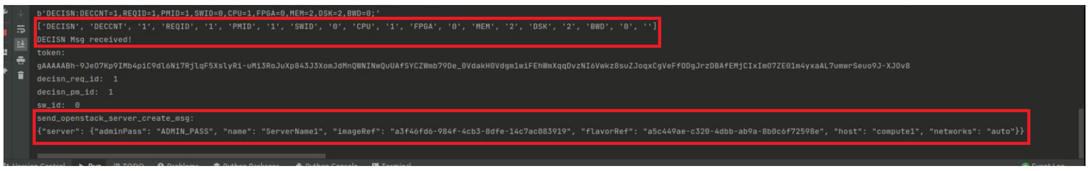
ACCLOUD-MAN receives the request message from ARS as shown in Figure 4.8. ACCLOUD-MAN produces a decision right after `REQUST` message is received. This decision is sent to ACCMAN-INT with `DECISN` message as shown in Figure 4.8. From the Table 4.1, the first customer request is an IaaS request with CPU compute resource. Because CPU resource is requested, ACCLOUD-MAN decides the CompNC-1, which is the one with CPU compute capability, to serve the customer as expected.



```
################################
Now wait for REQUST messages.
Received Data: [You are connected to AutomaticRequestSender]
Received Data: [REQUST:REQID=1,ARRTIME=5,DRTN=90,SIZE=0,SWID=0,CPU=1,FPGA=0,MEM=2,DSK=2,BWD=0;]
REQUST message arrived
ProcessedEvents:1, OnPM:0 ReqCount:0
EventType:0
Number of decisions: 1
DECISN: stpRequestArray(1).nID: 1, Decisions(1).nPM: 1
decisn_msg: DECISN:DECCNT=1,REQID=1,PMID=1,SWID=0,CPU=1,FPGA=0,MEM=2,DSK=2,BWD=0;
```

Figure 4.8: [ACCLOUD-MAN] ACCLOUD-MAN receives `REQUST` message from ARS and sends `DECISN` message to ACCMAN-INT

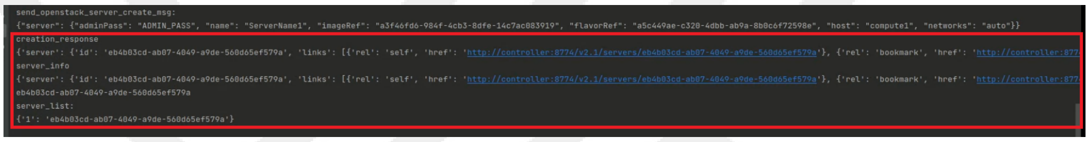
ACCMAN-INT receives `DECISN` message from ACCLOUD-MAN. ACCMAN-INT prepares a OpenStack API message to create VM with the information in received `DECISN` message. The VM creation message that is sent to Nova-Compute service can be observed in Figure 4.9.



```
b'DECISN:DECCNT=1,REQID=1,PMID=1,SWID=0,CPU=1,FPGA=0,MEM=2,DSK=2,BWD=0;'
['DECISN', 'DECCNT', '1', 'REQID', '1', 'PMID', '1', 'SWID', '0', 'CPU', '1', 'FPGA', '0', 'MEM', '2', 'DSK', '2', 'BWD', '0', '']
DECISN Msg received!
token:
gAAAAABh-9JeO7Kp9IMb4piC9dl6Ni7RjlqF5XslyRi-uMi3RoJuXp843J3XomJdMnQWNINwQuUAfSYCZWmb79De_0VdakH0Vdgm1wiFEhWmXqqDvzNI6Vwkz8suZJoqxCgVeFfODgJrzDBAfEMjCIxImO7ZE01m4yxaAL7umwrSeuo9J-XJOv8
decisn_req_id:  1
decisn_pm_id:  1
sw_id:  0
send_openstack_server_create_msg:
{"server": {"adminPass": "ADMIN_PASS", "name": "ServerName1", "imageRef": "a3f46fd6-984f-4cb3-8dfe-14c7ac083919", "flavorRef": "a5c449ae-c320-4dbb-ab9a-8b0c6f72598e", "host": "compute1", "networks": "auto"}}
```

Figure 4.9: [ACCMAN-INT] ACCMAN-INT receives DECISN message from ACCLOUD-MAN and sends REST API message to Nova-Compute

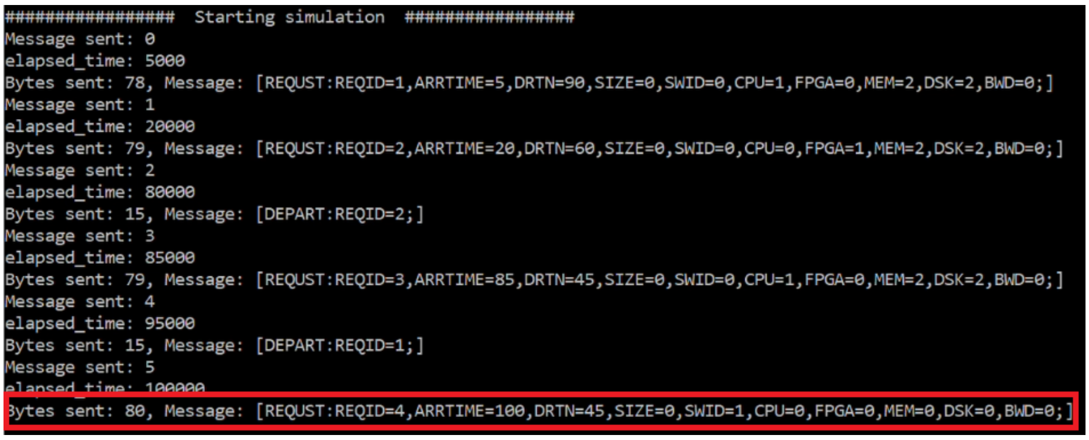
Nova-Compute service returns response message to ACCMAN-INT after VM creation process. If the process is successful, response message includes the unique ID. Otherwise, response message contains the error message. The unique ID, that can be seen as "eb4b03cd-...f579a" in Figure 4.10, is also stored for further operations.



```
send_openstack_server_create_msg:
{"server": {"adminPass": "ADMIN_PASS", "name": "ServerName1", "imageRef": "a3f46fd6-984f-4cb3-8dfe-14c7ac083919", "flavorRef": "a5c449ae-c320-4dbb-ab9a-8b0c6f72598e", "host": "compute1", "networks": "auto"}}
creation_response
{'server': {'id': 'eb4b03cd-ab07-4049-a9de-560d65ef579a', 'links': [{'rel': 'self', 'href': 'http://controller:8774/v2.1/servers/eb4b03cd-ab07-4049-a9de-560d65ef579a'}, {'rel': 'bookmark', 'href': 'http://controller:877
server_info
{'server': {'id': 'eb4b03cd-ab07-4049-a9de-560d65ef579a', 'links': [{'rel': 'self', 'href': 'http://controller:8774/v2.1/servers/eb4b03cd-ab07-4049-a9de-560d65ef579a'}, {'rel': 'bookmark', 'href': 'http://controller:877
eb4b03cd-ab07-4049-a9de-560d65ef579a
server_list:
{'1': 'eb4b03cd-ab07-4049-a9de-560d65ef579a'}
```

Figure 4.10: [ACCMAN-INT] ACCMAN-INT receives response message from Nova-Compute for VM creation message

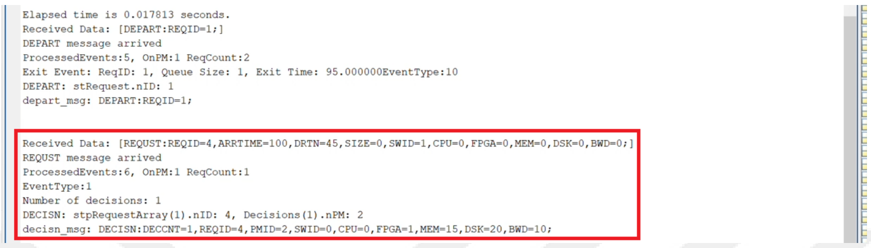
After customer #1 request is served, customer #2 and #3 requests are sent at $t = 20s$ and $t = 85s$ to ACCLOUD-MAN. Customer #2 leaves the cloud at $t = 80s$. Customer #4's REQUST message is sent to ACCLOUD-MAN from ARS at $t = 100s$. ARS print for sending the secont REQUST message is shown in Figure 4.11.



```
#################  Starting simulation  #################
Message sent: 0
elapsed_time: 5000
Bytes sent: 78, Message: [REQUST:REQID=1,ARRTIME=5,DRTN=90,SIZE=0,SWID=0,CPU=1,FPGA=0,MEM=2,DSK=2,BWD=0;]
Message sent: 1
elapsed_time: 20000
Bytes sent: 79, Message: [REQUST:REQID=2,ARRTIME=20,DRTN=60,SIZE=0,SWID=0,CPU=0,FPGA=1,MEM=2,DSK=2,BWD=0;]
Message sent: 2
elapsed_time: 80000
Bytes sent: 15, Message: [DEPART:REQID=2;]
Message sent: 3
elapsed_time: 85000
Bytes sent: 79, Message: [REQUST:REQID=3,ARRTIME=85,DRTN=45,SIZE=0,SWID=0,CPU=1,FPGA=0,MEM=2,DSK=2,BWD=0;]
Message sent: 4
elapsed_time: 95000
Bytes sent: 15, Message: [DEPART:REQID=1;]
Message sent: 5
elapsed_time: 100000
Bytes sent: 80, Message: [REQUST:REQID=4,ARRTIME=100,DRTN=45,SIZE=0,SWID=1,CPU=0,FPGA=0,MEM=0,DSK=0,BWD=0;]
```

Figure 4.11: [ARS] ARS sends the second REQUST message to ACCLOUD-MAN

The ACCLOUD-MAN software receives `REQUST` message for customer #4 from ARS. This request is a SaaS request. Since the incoming request message is a SaaS type (in social media type, `SWID = 1`), ACCLOUD-MAN checks the number of physical resources it can fulfill according to the table given in Appendix A. In Figure 4.12, ACCLOUD-MAN chooses CompNC-2 that is the one with FPGA compute resource since it is the best option with minimum power consumption.

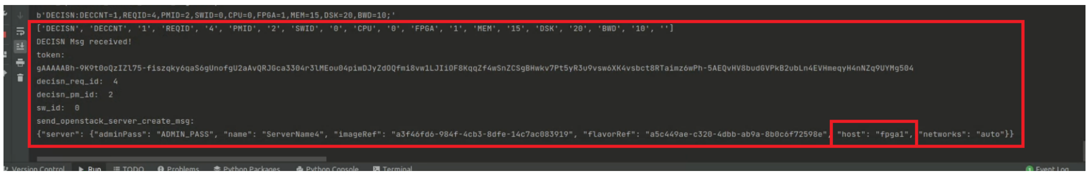


```
Elapsed time is 0.017813 seconds.
Received Data: [DEPART:REQID=1;]
DEPART message arrived
ProcessedEvents:5, OnPM:1 ReqCount:2
Exit Event: ReqID: 1, Queue Size: 1, Exit Time: 95.000000EventType:10
DEPART: stRequest.nID: 1
depart_msg: DEPART:REQID=1;

Received Data: [REQUST:REQID=4,ARRTIME=100,DRTN=45,SIZE=0,SWID=1,CPU=0,FPGA=0,MEM=0,DSK=0,BWD=0;]
REQUST message arrived
ProcessedEvents:6, OnPM:1 ReqCount:1
EventType:1
Number of decisions: 1
DECISN: stpRequestArray(1).nID: 4, Decisions(1).nPM: 2
decisn_msg: DECISN:DECCNT=1,REQID=4,PMID=2,SWID=0,CPU=0,FPGA=1,MEM=15,DSK=20,BWD=10;
```

Figure 4.12: [ACCLOUD-MAN] ACCCLOUD-MAN prepares and sends `DECISN` message to ACCMAN-INT for second `REQUST` message

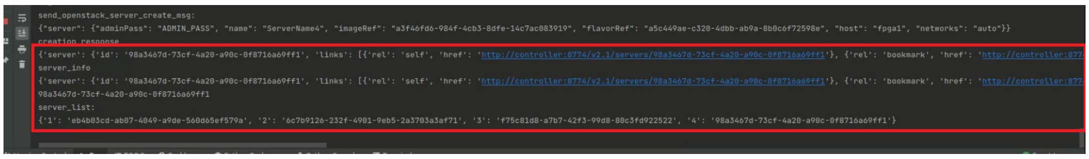
ACCMAN-INT software receives `DECISN` message and, as in the previous request messages, sends a Nova-Compute REST API message to create of VM according to the content of the received message. VM creation message sent to Nova-Compute can be observed in Figure 4.13.



```
b'DECISN:DECCNT=1,REQID=4,PMID=2,SWID=0,CPU=0,FPGA=1,MEM=15,DSK=20,BWD=10;'
['DECISN', 'DECCNT', '1', 'REQID', '4', 'PMID', '2', 'SWID', '0', 'CPU', '0', 'FPGA', '1', 'MEM', '15', 'DSK', '20', 'BWD', '10', '']
DECISN Msg received!
token:
gAAAAABh-9K9t0oQzIZl75-fiszqky6qaS6gUnofgU2aAvQRJGca3304r3lMEou04piwDJyZdOQfmi8vw1LJIiOF8KqqZf4wSnZCSgBHwkv7Pt5yR3u9vsw6XK4vsbct8RTaimz6wPh-5AEQvHV8budGVPkB2ubLn4EVHmeqyH4nNZq9UYMg504
decisn_req_id:  4
decisn_pm_id:  2
sw_id:  0
send_openstack_server_create_msg:
{"server": {"adminPass": "ADMIN_PASS", "name": "ServerName4", "imageRef": "a3f46fd6-984f-4cb3-8dfe-14c7ac083919", "flavorRef": "a5c449ae-c320-4dbb-ab9a-8b0c6f72598e", "host": "fpga1", "networks": "auto"}}
```

Figure 4.13: [ACCMAN-INT] ACCMAN-INT receives `DECISN` message from ACCLOUD-MAN and sends REST API message to Nova-Compute

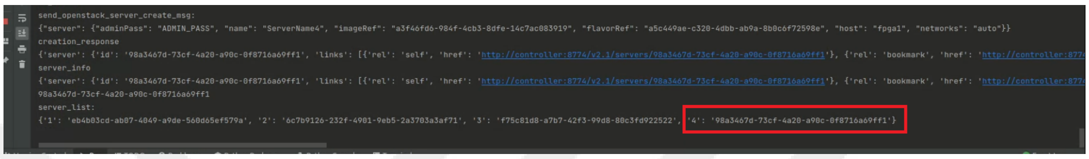
ACCMAN-INT receives VM creation response from Nova-Compute for the second customer request along with its unique identifier number, which can be seen in Figure 4.14.



```
send_openstack_server_create_msg:
{"server": {"adminPass": "ADMIN_PASS", "name": "ServerName4", "imageRef": "a3f46fd6-984f-4cb3-8dfe-14c7ac083919", "flavorRef": "a5c449ae-c320-4dbb-ab9a-8b0c6f72598e", "host": "fpga1", "networks": "auto"}}
creation_response
{'server': {'id': '98a3467d-73cf-4a20-a90c-0f8716a69ff1', 'links': [{'rel': 'self', 'href': 'http://controller:8774/v2.1/servers/98a3467d-73cf-4a20-a90c-0f8716a69ff1'}, {'rel': 'bookmark', 'href': 'http://controller:877
server_info
{'server': {'id': '98a3467d-73cf-4a20-a90c-0f8716a69ff1', 'links': [{'rel': 'self', 'href': 'http://controller:8774/v2.1/servers/98a3467d-73cf-4a20-a90c-0f8716a69ff1'}, {'rel': 'bookmark', 'href': 'http://controller:877
98a3467d-73cf-4a20-a90c-0f8716a69ff1
server_list:
{'1': 'eb4b03cd-ab07-4049-a9de-560d65ef579a', '2': '6c7b9126-232f-4901-9eb5-2a3703a3af71', '3': 'f75c81d8-a7b7-42f3-99d8-80c3fd922522', '4': '98a3467d-73cf-4a20-a90c-0f8716a69ff1'}
```

Figure 4.14: [ACCMAN-INT] VMs are created in OpenStack

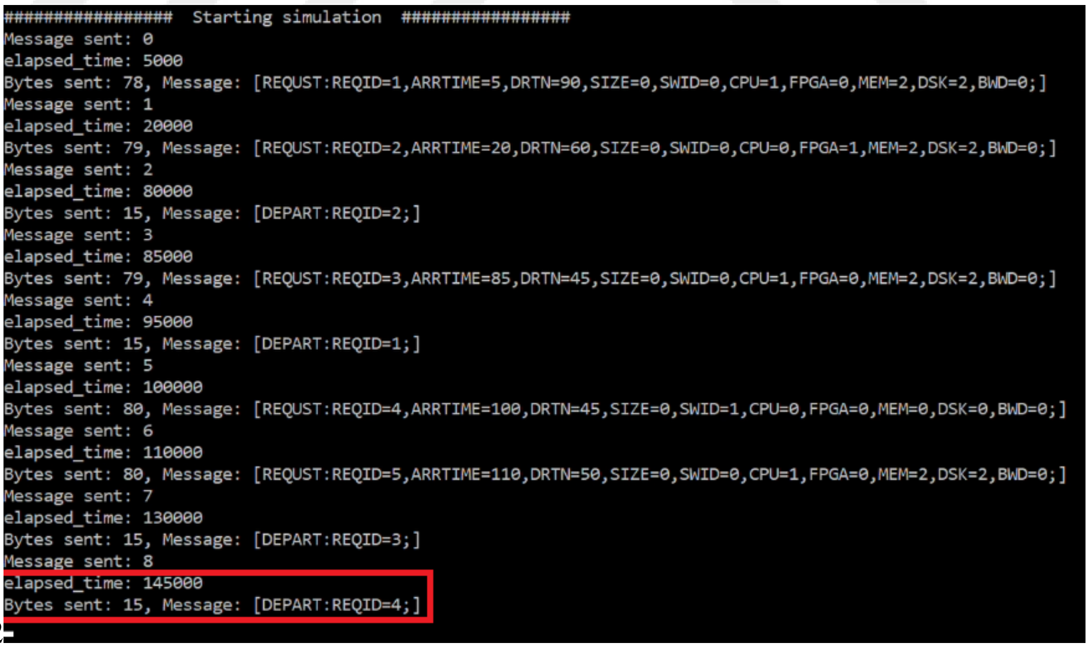
ACCMAN-INT software stores a list of created VMs according to `REQID` in `DECISN` message numbers and unique VM ID. The list is shown in Figure 4.15.



```
send_openstack_server_create_msg:
{"server": {"adminPass": "ADMIN_PASS", "name": "ServerName4", "imageRef": "a3f46fd6-984f-4cb3-8dfe-14c7ac083919", "flavorRef": "a5c449ae-c320-4dbb-ab9a-8b0c6f72598e", "host": "fpga1", "networks": "auto"}}
creation_response
{'server': {'id': '98a3467d-73cf-4a20-a90c-0f8716a69ff1', 'links': [{'rel': 'self', 'href': 'http://controller:8774/v2.1/servers/98a3467d-73cf-4a20-a90c-0f8716a69ff1'}, {'rel': 'bookmark', 'href': 'http://controller:877
server_info
{'server': {'id': '98a3467d-73cf-4a20-a90c-0f8716a69ff1', 'links': [{'rel': 'self', 'href': 'http://controller:8774/v2.1/servers/98a3467d-73cf-4a20-a90c-0f8716a69ff1'}, {'rel': 'bookmark', 'href': 'http://controller:877
98a3467d-73cf-4a20-a90c-0f8716a69ff1
server_list:
{'1': 'eb4b03cd-ab07-4049-a9de-560d65ef579a', '2': '6c7b9126-232f-4901-9eb5-2a3703a3af71', '3': 'f75c81d8-a7b7-42f3-99d8-80c3fd922522', '4': '98a3467d-73cf-4a20-a90c-0f8716a69ff1'}
```

Figure 4.15: [ACCMAN-INT] List of VMs created

At $t = 145$, departure event for customer #4 takes place. ARS sends `DEPART` message to ACCLOUD-MAN for customer #4 as shown in Figure 4.16.

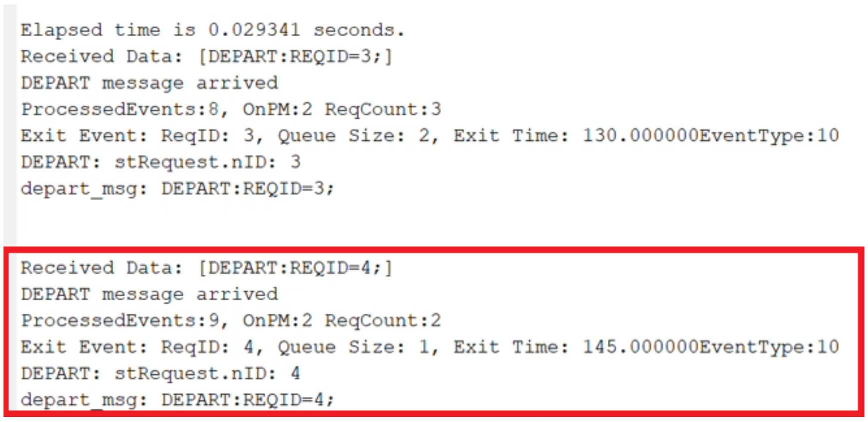


```
#################  Starting simulation  #################
Message sent: 0
elapsed_time: 5000
Bytes sent: 78, Message: [REQUST:REQID=1,ARRTIME=5,DRTN=90,SIZE=0,SWID=0,CPU=1,FPGA=0,MEM=2,DSK=2,BWD=0;]
Message sent: 1
elapsed_time: 20000
Bytes sent: 79, Message: [REQUST:REQID=2,ARRTIME=20,DRTN=60,SIZE=0,SWID=0,CPU=0,FPGA=1,MEM=2,DSK=2,BWD=0;]
Message sent: 2
elapsed_time: 80000
Bytes sent: 15, Message: [DEPART:REQID=2;]
Message sent: 3
elapsed_time: 85000
Bytes sent: 79, Message: [REQUST:REQID=3,ARRTIME=85,DRTN=45,SIZE=0,SWID=0,CPU=1,FPGA=0,MEM=2,DSK=2,BWD=0;]
Message sent: 4
elapsed_time: 95000
Bytes sent: 15, Message: [DEPART:REQID=1;]
Message sent: 5
elapsed_time: 100000
Bytes sent: 80, Message: [REQUST:REQID=4,ARRTIME=100,DRTN=45,SIZE=0,SWID=1,CPU=0,FPGA=0,MEM=0,DSK=0,BWD=0;]
Message sent: 6
elapsed_time: 110000
Bytes sent: 80, Message: [REQUST:REQID=5,ARRTIME=110,DRTN=50,SIZE=0,SWID=0,CPU=1,FPGA=0,MEM=2,DSK=2,BWD=0;]
Message sent: 7
elapsed_time: 130000
Bytes sent: 15, Message: [DEPART:REQID=3;]
Message sent: 8
elapsed_time: 145000
Bytes sent: 15, Message: [DEPART:REQID=4;]
```

2

Figure 4.16: [ARS] ARS sends `DEPART` message to ACCLOUD-MAN for customer #2

ACCLOUD-MAN receives `DEPART` message and sends another `DEPART` message to the ACCMAN-INT to terminate related VM to release used physical resources. Received and sent `DEPART` message is shown in Figure 4.17.

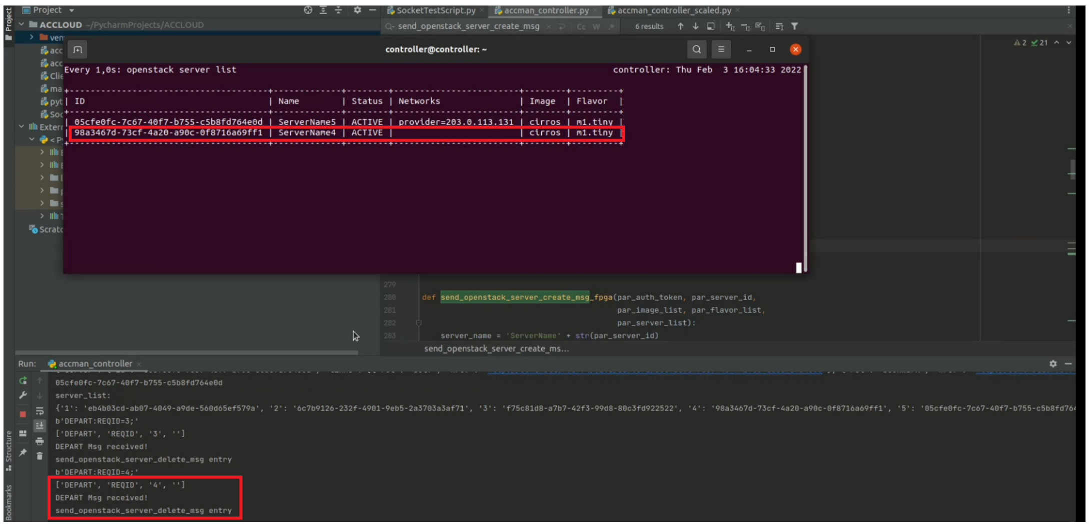


```
Elapsed time is 0.029341 seconds.
Received Data: [DEPART:REQID=3;]
DEPART message arrived
ProcessedEvents:8, OnPM:2 ReqCount:3
Exit Event: ReqID: 3, Queue Size: 2, Exit Time: 130.000000EventType:10
DEPART: stRequest.nID: 3
depart_msg: DEPART:REQID=3;

Received Data: [DEPART:REQID=4;]
DEPART message arrived
ProcessedEvents:9, OnPM:2 ReqCount:2
Exit Event: ReqID: 4, Queue Size: 1, Exit Time: 145.000000EventType:10
DEPART: stRequest.nID: 4
depart_msg: DEPART:REQID=4;
```

Figure 4.17: [ACCLOUD-MAN] ACCLOUD-MAN receives `DEPART` message from ARS and sends to ACCMAN-INT

ACCMAN-INT receives `DEPART` message. ACCMAN-INT prepares and sends VM deletion message according to received `DEPART` message. In Figure 4.18, it is also observed that VM for customer #4 is up and running.



```
Every 1,0s: openstack server list                                    controller: Thu Feb  3 16:04:33 2022

+--------------------------------------+-------------+--------+--------------------------+--------+---------+
| ID                                   | Name        | Status | Networks                 | Image  | Flavor  |
+--------------------------------------+-------------+--------+--------------------------+--------+---------+
| 05cfe0fc-7c67-40f7-b755-c5b8fd764e0d | ServerName5 | ACTIVE | provider=203.0.113.131   | cirros | m1.tiny |
| 98a3467d-73cf-4a20-a90c-0f8716a69ff1 | ServerName4 | ACTIVE |                          | cirros | m1.tiny |
+--------------------------------------+-------------+--------+--------------------------+--------+---------+
```

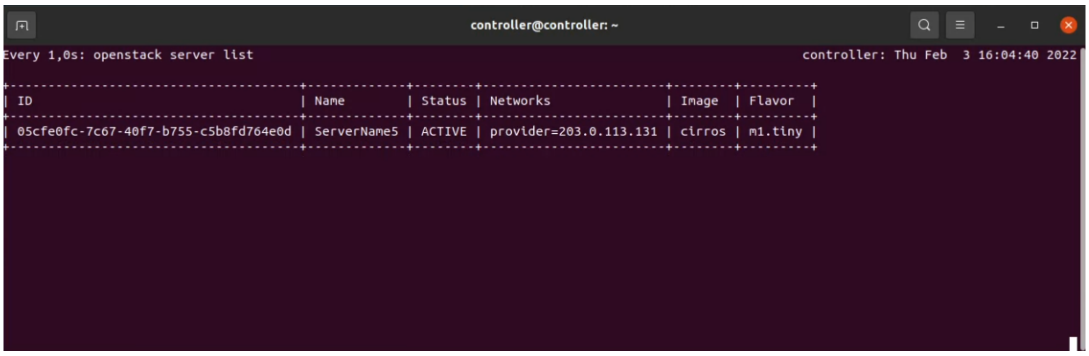
```
def send_openstack_server_create_msg_fpga(par_auth_token, par_server_id,
                                          par_image_list, par_flavor_list,
                                          par_server_list):
    server_name = 'ServerName' + str(par_server_id)
```

```
05cfe0fc-7c67-40f7-b755-c5b8fd764e0d
server_list:
{'1': 'eb4b03cd-ab07-4049-a9de-560d65ef579a', '2': '6c7b9126-232f-4901-9eb5-2a3703a3af71', '3': 'f75c81d8-a7b7-42f3-99d8-80c3fd922522', '4': '98a3467d-73cf-4a20-a90c-0f8716a69ff1', '5': '05cfe0fc-7c67-40f7-b755-c5b8fd76
b'DEPART:REQID=3;'
['DEPART', 'REQID', '3', '']
DEPART Msg received!
send_openstack_server_delete_msg entry
b'DEPART:REQID=4;'
['DEPART', 'REQID', '4', '']
DEPART Msg received!
send_openstack_server_delete_msg entry
```

Figure 4.18: [ACCMAN-INT] ACCMAN-INT receives DEPART message

It can be observed in Figure 4.19 that VM is deleted from the VM list after OpenStack

Nova-Compute has performed deletion operations. If deletion operation is successful, Nova-Compute does not return any response.

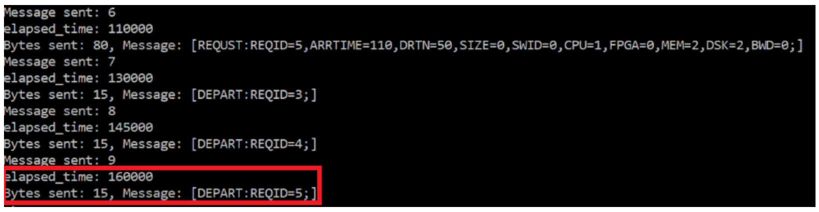


```
controller@controller: ~
Every 1,0s: openstack server list                                      controller: Thu Feb  3 16:04:40 2022

+--------------------------------------+-------------+--------+-------------------------+--------+---------+
| ID                                   | Name        | Status | Networks                | Image  | Flavor  |
+--------------------------------------+-------------+--------+-------------------------+--------+---------+
| 05cfe0fc-7c67-40f7-b755-c5b8fd764e0d | ServerName5 | ACTIVE | provider=203.0.113.131  | cirros | m1.tiny |
+--------------------------------------+-------------+--------+-------------------------+--------+---------+
```

Figure 4.19: [ACCMAN-INT] VM created for customer #2 is deleted by Nova-Compute

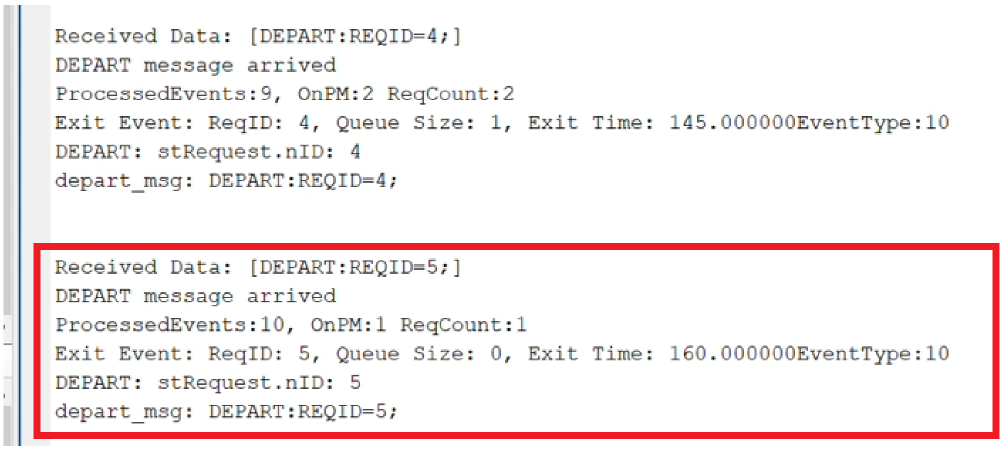
As last event of this scenario, ARS sends `DEPART` message to ACCLOUD-MAN for customer #5 as shown Figure 4.20.



```
Message sent: 6
elapsed_time: 110000
Bytes sent: 80, Message: [REQUST:REQID=5,ARRTIME=110,DRTN=50,SIZE=0,SWID=0,CPU=1,FPGA=0,MEM=2,DSK=2,BWD=0;]
Message sent: 7
elapsed_time: 130000
Bytes sent: 15, Message: [DEPART:REQID=3;]
Message sent: 8
elapsed_time: 145000
Bytes sent: 15, Message: [DEPART:REQID=4;]
Message sent: 9
elapsed_time: 160000
Bytes sent: 15, Message: [DEPART:REQID=5;]
```

Figure 4.20: [ARS] ARS sends `DEPART` message for customer #1

ACCLOUD-MAN sends the related `DEPART` message to ACCMAN-INT. ACCLOUD-MAN receives `DEPART` message for customer #5 from ARS and forwards it to ACCMAN-INT as shown in Figure 4.21.

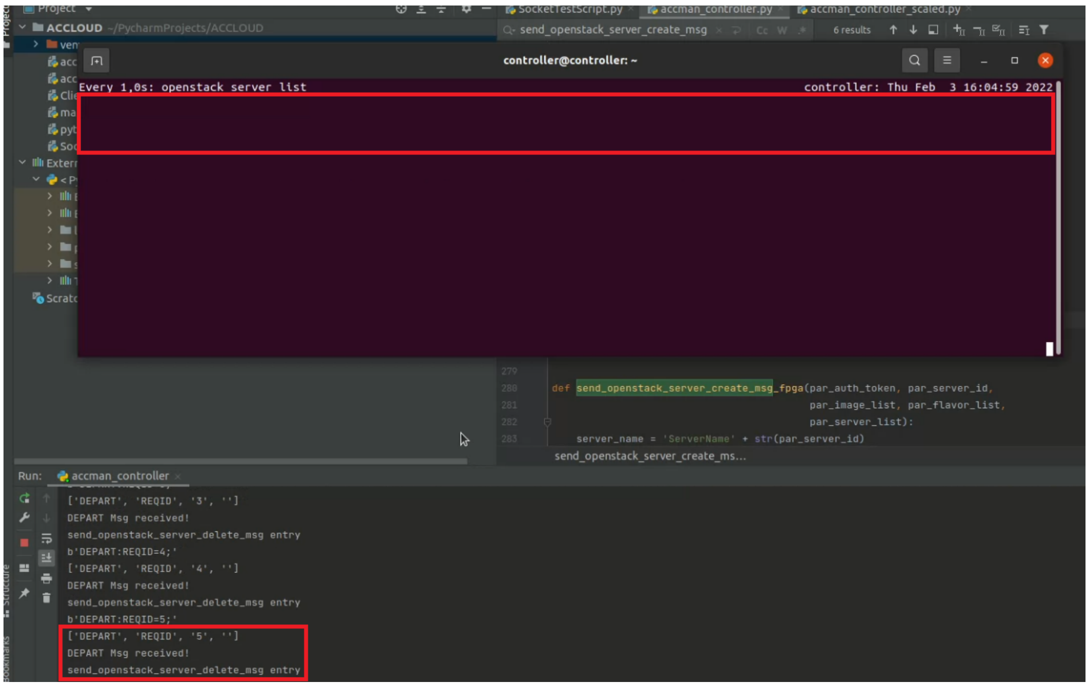


```
Received Data: [DEPART:REQID=4;]
DEPART message arrived
ProcessedEvents:9, OnPM:2 ReqCount:2
Exit Event: ReqID: 4, Queue Size: 1, Exit Time: 145.000000EventType:10
DEPART: stRequest.nID: 4
depart_msg: DEPART:REQID=4;

Received Data: [DEPART:REQID=5;]
DEPART message arrived
ProcessedEvents:10, OnPM:1 ReqCount:1
Exit Event: ReqID: 5, Queue Size: 0, Exit Time: 160.000000EventType:10
DEPART: stRequest.nID: 5
depart_msg: DEPART:REQID=5;
```

Figure 4.21: [ACCLOUD-MAN] ACCLOUD-MAN receives `DEPART` message and forwards it to ACCMAN-INT

ACCMAN-INT receives `DEPART` message for customer #5 and sends corresponding Nova-Compute VM deletion REST API message. With this deletion, the scenario is completed and all created VMs during deleted. In Figure 4.22, it can be observed that all VMs that are created during this test run are deleted.



```
Every 1,0s: openstack server list                    controller: Thu Feb  3 16:04:59 2022
```

```
def send_openstack_server_create_msg_fpga(par_auth_token, par_server_id,
                                          par_image_list, par_flavor_list,
                                          par_server_list):
    server_name = 'ServerName' + str(par_server_id)
```

```
['DEPART', 'REQID', '3', '']
DEPART Msg received!
send_openstack_server_delete_msg entry
b'DEPART:REQID=4;'
['DEPART', 'REQID', '4', '']
DEPART Msg received!
send_openstack_server_delete_msg entry
b'DEPART:REQID=5;'
['DEPART', 'REQID', '5', '']
DEPART Msg received!
send_openstack_server_delete_msg entry
```

Figure 4.22: [ACCMAN-INT] All VMs created for test run are deleted in ACCLOUD

This test is recorded and can be accessed using links [57, 58, 59].

4.2 Test - 2: ACCLOUD-MAN Minimum Power Test

This test evaluates ACCLOUD-MAN's capability in the extended OpenStack for selecting resource allocation alternatives for SaaS requests to achieve minimum power consumption. The test scenario is provided in Table 4.2.

Table 4.2: ACCLOUD Minimum Power Test Scenario

Customer #	**Arrival** (s)	**Duration** (s)	**Departure** (s)	**Service**	**Compute Resource Type**
1	5	10	15	IaaS	CPU
2	20	10	30	IaaS	CPU
3	35	10	45	IaaS	CPU
4	50	10	60	IaaS	FPGA
5	55	10	65	IaaS	CPU
6	70	30	100	SaaS	Type #1
7	75	30	105	SaaS	Type #1
8	80	30	110	SaaS	Type #1
9	115	10	125	IaaS	CPU
10	120	10	130	IaaS	CPU

In the scenario given in Table 4.2, customers' #6 to #8 requests are SaaS requests where ACCLOUD can assess different alternatives. At time $t = 70s$ customer #6 sends a Type-1 SaaS request. From the ACCLOUD-MAN Alternatives Table given in Appendix A, Type-1 request has two alternatives, one with 2 CPU cores and the other one with 1 FPGA region. Since FPGA alternative has less power consumption, ACCLOUD-MAN prefers FPGA alternative over CPU for customer #6. Customer #6 stays in the cloud for $30s$ therefore customers #7 and #8 will be present at the same time since their time of arrivals are at $t = 75s$ and $t = 80s$, respectively. For customer #7, ACCLOUD-MAN decides to allocate another FPGA region. However, after customer #7's allocation, there are no available FPGA regions. Thus, customer #8 has no other option than CPU alternative. Here we note that this experiment is designed to show the capability of extended OpenStack and no real software is instantiated on the VMs created for SaaS requests. This test scenario also shows that, it is possible to serve several VM at the same time within this test setup, in fact three VMs at the same time in this test. As long as there are enough free resources in each type such as CPU, FPGA, memory, or disk a VM can be deployed according to ACCLOUD-MAN decisions.

To this end, OpenStack extended with ACCLOUD-MAN can make use of resource alternatives for SaaS requests and allocate power-efficient resources. Here, we note that OpenStack is designed to provide IaaS services. This experiment further demonstrates that employing a table for mapping SaaS requests to suitable VMs can facilitate offering SaaS services in OpenStack.

This test is recorded and can be accessed using links [60, 61].

4.3 Test - 3: ACCLOUD Deployment Time Test

In this test, ACCLOUD's deployment time introduced to the customers is investigated. Deployment time is defined as the elapsed time between the time of the request is received from ACCLOUD-MAN and the time of VM is created for the request. The delay introduced by the TCP communication is neglected in the deployment time measurement since it is in the order of $0.1ms$. To express the aim of this test more specifically, ACCLOUD-MAN's and ACCMAN-INT's integrated deployment time performance is measured.

50 IaaS with CPU compute resource VM requests are sent to the extended OpenStack with ACCLOUD-MAN. In Figure 4.23 (a), deployment time measurement for ACCMAN-INT measurements is shown. For the same test scenario, the decision delay of ACCLOUD-MAN decision process is shown in Figure 4.23 (b). Although ACCLOUD-MAN's decision delay is relatively small, the total deployment delay introduced by ACCLOUD is shown in Figure 4.23 (c).

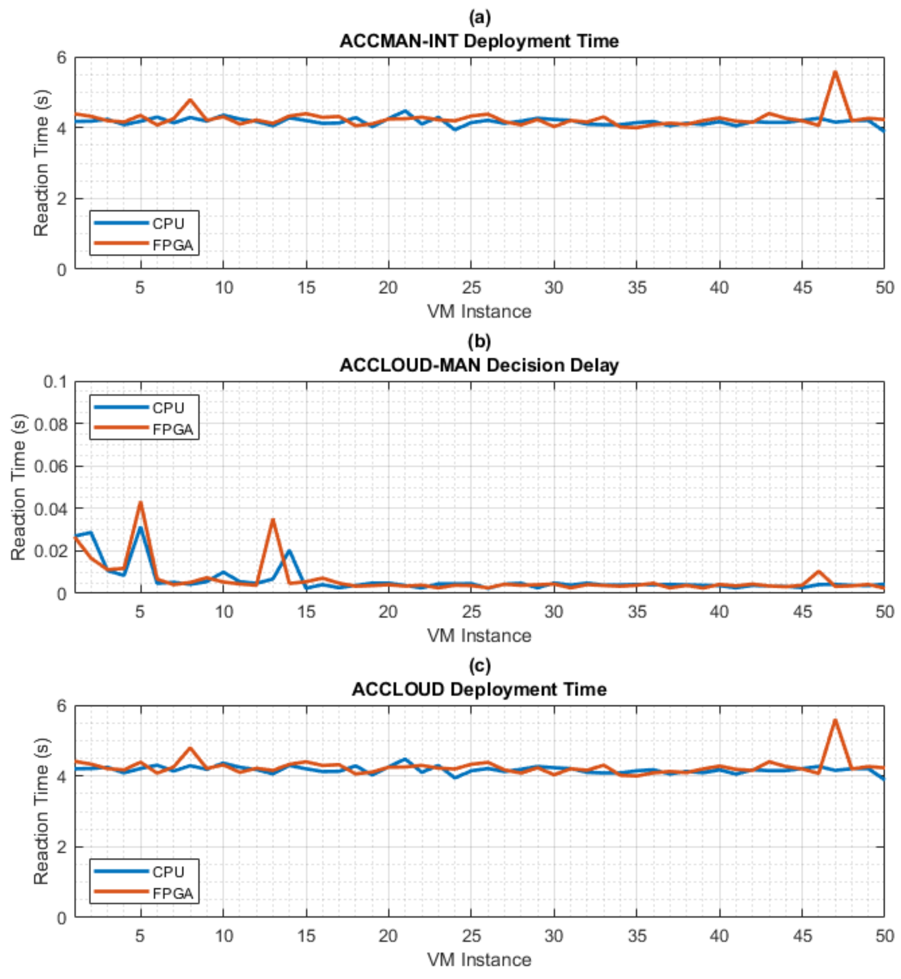


Figure 4.23: ACCLOUD Deployment Time Measurements

For the deploymeny time measurements given in Figure 4.23, minimum deployment time, maximum deployment time and average deployment time values are provided in Table 4.3.

Table 4.3: Deployment Time of ACCLOUD

CPU	**Time (s)**
Min. deployment time	3.88
Max. deployment time	4.48
Ave. deployment time	4.17
FPGA	**Time (s)**
Min. deployment time	3.99
Max. deployment time	5.6
Ave. deployment time	4.25

In the previous study on OpenStack support of heterogeneous resources [6], OpenStack VM creation time is measured as approximately 3 seconds. We emphasize that in [6], measurements are completed in a simulated OpenStack deployment. Simulated OpenStack deployment means that VMs created with Oracle VirtualBox are used as OpenStack nodes. Unlike [6], this study PMs are utilized as OpenStack nodes and physical connections are established. Despite deployment differences, this test yields consistent results with simulated VM creation reaction time. The difference between average deployment time measurement in this study and [6] is interpreted as following dissimilarities.

- OpenStack nodes are installed on different machines
- OpenStack versions are different (Pike is used in [6], Wallaby is used in this study)

4.4 Test - 4: ACCLOUD Durability Test

In this test, it is observed whether any errors occur when ACCLOUD operates for long periods of time. In the test run, as a scenario, customers who come every half hour for 8 hours are designed to receive service approximately one hour. As a result of this test, it is observed that the setup stays operational for 8 hours without any problems. One of the important points in this test is to use an up-to-date `auth_token` for OpenStack messaging. For this reason, it is necessary for long-term studies to make a new token request at certain intervals. Through this study, it has been shown that the ACCLOUD setup can work for long periods of time.

CHAPTER 5

CONCLUSION AND FUTURE WORK

OpenStack [3] is an open-source platform for the resource management of cloud-based data centers. OpenStack maintains a database of available computing resources together with *virtual machine* (VM) configurations. It collects the user Infrastructure as a Service (IaaS) Requests and instantiates VM images that are fetched from a repository. The conventional computing resources for VM construction comprise CPU, memory, and disk. In the recent heterogeneous cloud data centers, these resources are extended by hardware accelerators that are implemented on FPGA. Cloud computing services are offered in different levels of abstractions and virtualization. An important service in this context is Software as a Service (SaaS) which processes the user's data and returns the output is returned to the user. The cloud resource allocation and management methods are expected to fulfill the requirements of these service varieties while utilizing the heterogeneous computing resources efficiently and observing power concerns. The current default resource management of OpenStack is focused on fulfilling the resource requirements of IaaS requests and creating VMs accordingly without any efficiency and power optimization of the cloud data center.

In this thesis we integrate the ILP-based cloud resource allocation method in [4, 5] that we call ACCLOUD-MAN (ACcelerated CLOUD MANagement) in the most recent version of OpenStack to develop an extended OpenStack implementation. ACCLOUD-MAN considers alternative VM compositions for SaaS requests that exploit the power efficiency of hardware accelerators. ACCLOUD-MAN allocates the resources to the incoming user requests with a power minimization objective. Our OpenStack extension is carried out on the most recent version of OpenStack that was available at the time of writing this thesis. To this end, we design a software architecture that interfaces the ILP solver of ACCLOUD-MAN and OpenStack following the OpenStack APIs. The entire implementation is installed in a realistic laboratory set-up with cloud servers, FPGA Cards with hardware accelerators, and a 40 Gbps network.

Our functional tests on this laboratory set-up show that our extended OpenStack implementation sends and receives all messages to the ILP solver and the decision of ACCLOUD-MAN is realized with the correct message sequence according to the API. The decision capabilities of ACCLOUD-MAN among SaaS VM alternatives are demonstrated with custom-designed request arrival scenarios. We evaluate the performance of the extended OpenStack developed in this thesis regarding heterogeneous VM instantiation reaction time. Our results show that the overhead of ACCLOUD-MAN and the integration software is small with respect to the VM starting times. Furthermore, we observe that laboratory set-up and developed software promise long hours of operations without any errors.

The future work includes incorporating the user request collection interface of OpenStack in our extended version to have a fully transparent integration. Furthermore, it is possible to collect instantaneous machine utilization statistics and add migration capabilities to the entire framework for further power optimizations.

REFERENCES

[1] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[2] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, *et al.*, "A cloud-scale acceleration architecture," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, p. 7, IEEE Press, 2016.

[3] "Open source cloud computing platform software - openstack." `https://www.openstack.org/software/`. OpenStack, Accessed: 2022-02-01.

[4] N. U. Ekici, "Optimal dynamic resource allocation for heterogenous cloud data centers," Master's thesis, Middle East Technical University, 2019.

[5] N. U. Ekici, K. W. Schmidt, A. Yazar, and E. G. Schmidt, "Resource allocation for minimized power consumption in hardware accelerated clouds," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–8, IEEE, 2019.

[6] A. Erol, "Generalized resource management for heterogeneous cloud data centers," Master's thesis, Middle East Technical University, 2019.

[7] A. Erol, A. Yazar, and E. G. Schmidt, "Openstack generalization for hardware accelerated clouds," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–8, IEEE, 2019.

[8] "Data centers." `https://www.ibm.com/uk-en/cloud/learn/data-centers`. Gartner, Accessed: 2022-01-16.

[9] C.-T. Yang, K.-L. Huang, J.-C. Liu, Y.-W. Su, and W. C.-C. Chu, "Implementation of a power saving method for virtual machine management in cloud," in

2013 International conference on cloud computing and big data, pp. 283–290, IEEE, 2013.

[10] "What is cloud computing? – amazon web services (aws)." `https://aws.amazon.com/what-is-cloud-computing`. Amazon, Accessed: 2022-01-04.

[11] "The data center is (almost) dead." `https://www.gartner.com/smarterwithgartner/the-data-center-is-almost-dead, Gartner`. (Accessed on 01/16/2022).

[12] "Google cloud products." `https://cloud.google.com/products`. Google Cloud, Accessed: 2022-01-05.

[13] "Cloud products." `https://aws.amazon.com/products/`. AWS, Accessed: 2022-01-05.

[14] "Cloud products." `https://azure.microsoft.com/en-us/services/`. Microsoft Azure, Accessed: 2022-01-05.

[15] S. A. Babu, M. Hareesh, J. P. Martin, S. Cherian, and Y. Sastri, "System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and xenserver," in *2014 Fourth International Conference on Advances in Computing and Communications*, pp. 247–250, IEEE, 2014.

[16] N. Jain and S. Choudhary, "Overview of virtualization in cloud computing," in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, pp. 1–4, 2016.

[17] F. Yazıcı, "A novel flexible on-chip switch architecture for reconfigurable hardware accelerators," Master's thesis, Middle East Technical University, 2021.

[18] "Iaas vs. paas vs. saas – ibm." `https://www.ibm.com/cloud/learn/iaas-paas-saas`. IBM, Accessed: 2022-01-05.

[19] "Cloud products." `https://azure.microsoft.com/en-us/overview/what-is-iaas/#overview`. Azure, Accessed: 2022-01-05.

[20] A. Yousafzai, A. Gani, R. M. Noor, M. Sookhak, H. Talebian, M. Shiraz, and M. K. Khan, “Cloud resource allocation schemes: review, taxonomy, and opportunities,” *Knowledge and Information Systems*, vol. 50, no. 2, pp. 347–381, 2017.

[21] M. Liaqat, A. Naveed, R. L. Ali, J. Shuja, and K.-M. Ko, “Characterizing dynamic load balancing in cloud environments using virtual machine deployment models,” *IEEE Access*, vol. 7, pp. 145767–145776, 2019.

[22] M. Mahjoub, A. Mdhaffar, R. B. Halima, and M. Jmaiel, “A comparative study of the current cloud computing technologies and offers,” in *2011 First International Symposium on Network Cloud Computing and Applications*, pp. 131–134, IEEE, 2011.

[23] J. Ahmed, A. Malik, M. U. Ilyas, and J. S. Alowibdi, “Instance launch-time analysis of openstack virtualization technologies with control plane network errors,” *Computing*, vol. 101, no. 8, pp. 989–1014, 2019.

[24] S. Lima, A. Rocha, and L. Roque, “An overview of openstack architecture: a message queuing services node,” *Cluster Computing*, vol. 22, no. 3, pp. 7087–7098, 2019.

[25] “Cloud adoption statistics.” `https://www.centerprise.co.uk/news/cloud-adoption-statistics-you-need-to-know`. Centerprise International, Accessed on 01/25/2022.

[26] A. Corradi, M. Fanelli, and L. Foschini, “Vm consolidation: A real case based on openstack cloud,” *Future Generation Computer Systems*, vol. 32, pp. 118–127, 2014.

[27] F. F. Moges and S. L. Abebe, “Energy-aware vm placement algorithms for the openstack neat consolidation framework,” *Journal of Cloud Computing*, vol. 8, no. 1, pp. 1–14, 2019.

[28] J. Jeon, J. H. Park, and Y.-S. Jeong, “Resource utilization scheme of idle virtual machines for multiple large-scale jobs based on openstack,” *Applied Sciences*, vol. 9, no. 20, p. 4327, 2019.

[29] S. Byma, J. G. Steffan, H. Bannazadeh, A. Leon-Garcia, and P. Chow, “Fpgas in the cloud: Booting virtualized hardware accelerators with openstack,” in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 109–116, IEEE, 2014.

[30] “Openstack installation guide.” `https://docs.openstack.org/install-guide/`. OpenStack, Accessed: 2022-01-16.

[31] “Compute schedulers — nova 24.1.0.dev206 documentation.” `https://docs.openstack.org/nova/latest/admin/scheduling.html`. OpenStack, Accessed: 2022-02-02.

[32] “Compute api — nova documentation.” `https://docs.openstack.org/api-ref/compute/?expanded=create-server-detail`. OpenStack, Accessed: 2022-02-02.

[33] “Openstack installation guide: Preface.” `https://docs.openstack.org/install-guide/preface.html`. OpenStack, Accessed: 2022-01-19.

[34] “Openstack installation guide: Environment.” `https://docs.openstack.org/install-guide/environment.html`. OpenStack, Accessed: 2022-01-19.

[35] “Minimal deployment for wallaby.” `https://docs.openstack.org/install-guide/openstack-services.html#minimal-deployment-for-wallaby`. OpenStack, Accessed: 2022-01-30.

[36] “About / hakkında - accloud.” `http://accloud.eee.metu.edu.tr/about.html`. (Accessed on 01/27/2022).

[37] “Cisco global cloud index: Forecast and methodology, 2016–2021.” `https://virtualization.network/Resources/Whitepapers/0b75cf2e-0c53-4891-918e-b542a5d364c5_white-paper-c11-738085.pdf`. Cisco, Accessed: 2022-01-27.

[38] “Supported versions.” `https://ubuntu.com/openstack/docs/supported-versions`. Ubuntu, Accessed: 2022-02-01.

[39] "Environment networking." https://docs.openstack.org/install-guide/environment-networking.html. OpenStack, Accessed: 2022-02-01.

[40] "Openstack packages for ubuntu." https://docs.openstack.org/install-guide/environment-packages-ubuntu.html. OpenStack, Accessed: 2022-02-01.

[41] "Sql database for ubuntu." https://docs.openstack.org/install-guide/environment-sql-database-ubuntu.html. OpenStack, Accessed: 2022-02-01.

[42] "Message queue for ubuntu." https://docs.openstack.org/install-guide/environment-messaging-ubuntu.html. OpenStack, Accessed: 2022-02-01.

[43] "Memcached for ubuntu." https://docs.openstack.org/install-guide/environment-memcached-ubuntu.html. OpenStack, Accessed: 2022-02-01.

[44] "Etcd for ubuntu." https://docs.openstack.org/install-guide/environment-etcd-ubuntu.html. OpenStack, Accessed: 2022-02-01.

[45] "Keystone: Install and configure." https://docs.openstack.org/keystone/wallaby/install/keystone-install-ubuntu.html. OpenStack, Accessed: 2022-02-01.

[46] "Glance: Install and configure (ubuntu)." https://docs.openstack.org/glance/wallaby/install/install-ubuntu.html. OpenStack, Accessed: 2022-02-01.

[47] "Install and configure placement for ubuntu." https://docs.openstack.org/placement/wallaby/install/install-ubuntu.html. OpenStack, Accessed: 2022-02-02.

[48] "Nova: Install and configure controller node for ubuntu." https://docs.openstack.org/nova/wallaby/install/

controller-install-ubuntu.html. OpenStack, Accessed: 2022-02-01.

[49] "Nova: Install and configure a compute node for ubuntu." https://docs.openstack.org/nova/wallaby/install/compute-install-ubuntu.html. OpenStack, Accessed: 2022-02-01.

[50] "Neutron: Install and configure controller node." https://docs.openstack.org/neutron/wallaby/install/controller-install-ubuntu.html. OpenStack, Accessed: 2022-02-01.

[51] "Neutron: Install and configure compute node." https://docs.openstack.org/neutron/wallaby/install/compute-install-ubuntu.html. OpenStack, Accessed: 2022-02-01.

[52] "Api examples using curl." https://docs.openstack.org/keystone/pike/api_curl_examples.html. OpenStack, Accessed: 2022-01-30.

[53] "Compute api: List flavors." https://docs.openstack.org/api-ref/compute/#list-flavors. OpenStack, Accessed: 2022-01-30.

[54] "Openstack api documentation: Glance." https://docs.openstack.org/api-ref/image/v2/index.html?expanded=list-images-detail#images. OpenStack, Accessed: 2022-01-30.

[55] "Openstack api documentation: Create server." https://docs.openstack.org/api-ref/compute/?expanded=create-server-detail. OpenStack, Accessed: 2022-01-30.

[56] "Openstack api documentation: Delete server." https://docs.openstack.org/api-ref/compute/?expanded=delete-server-detail. OpenStack, Accessed: 2022-01-30.

[57] "(video) accloud tez deneyi (thesis experiment) #1: Accloud-man." https://youtu.be/lT5BnpA0NIQ. ACCLOUD, Accessed: 2022-02-02.

[58] “(video) accloud tez deneyi (thesis experiment) #1: Accman-int.” https://youtu.be/OVopa5-wpOM. ACCLOUD, Accessed: 2022-02-02.

[59] “(video) accloud tez deneyi (thesis experiment) #1: Ars.” https://youtu.be/gLdzSk-JCJQ. ACCLOUD, Accessed: 2022-02-02.

[60] “(video) accloud kontrol düzlemi deney 2: Accloud-man.” https://www.youtube.com/watch?v=SnjVkXaXH0c&ab_channel=accloud. ACCLOUD, Accessed: 2022-02-02.

[61] “(video) accloud kontrol düzlemi deney 2: Accman-int.” https://www.youtube.com/watch?v=WP6Kfe68KS0&ab_channel=accloud. ACCLOUD, Accessed: 2022-02-02.

Appendix A

ACCLOUD-MAN ALTERNATIVES TABLE FOR SAAS REQUEST

- Job Type: ***Social Media***
  - Alternative #1
    - CPU cores: 0
    - FPGA regions: 1
    - Memory size: 15 GB
    - Disk size: 20 GB
    - Network bandwidth: 10 Gbps
  - Alternative #1
    - CPU cores: 2
    - FPGA regions: 0
    - Memory size: 15 GB
    - Disk size: 20 GB
    - Network bandwidth: 10 Gbps
- Job Type: ***Video streaming***
  - Alternative #1
    - CPU cores: 0
    - FPGA regions: 2
    - Memory size: 15 GB
    - Disk size: 20 GB
    - Network bandwidth: 20 Gbps

  - Alternative #2
    * CPU cores: 2
    * FPGA regions: 1
    * Memory size: 15 GB
    * Disk size: 20 GB
    * Network bandwidth: 20 Gbps
  - Alternative #3
    * CPU cores: 4
    * FPGA regions: 0
    * Memory size: 15 GB
    * Disk size: 20 GB
    * Network bandwidth: 20 Gbps
- Job Type: ***Collaboration***
  - Alternative #1
    * CPU cores: 4
    * FPGA regions: 0
    * Memory size: 7.5 GB
    * Disk size: 20 GB
    * Network bandwidth: 20 Gbps

Appendix B

ACCMAN-INT CODE SNIPPETS

B.1 Token Payload

```
auth_payload = {
    "auth": {
        "identity": {
            "methods": ["password"],
            "password": {
                "user": {
                    "name": "admin",
                    "domain": {"id": "default"},
                    "password": "ADMIN_PASS"
                }
            }
        },
        "scope": {
            "project": {
                "name": "admin",
                "domain": {"id": "default"}
            }
        }
    }
}
```

B.2 Token Request Code

```
# get keystone auth token
def get_auth_token():
    # Get Keystone token
    res = requests.post('http://controller:5000/v3/auth/tokens',
                        headers={'content-type': 'application/json
                            '},
                        data=json.dumps(auth_payload))
                            token = res.headers['X-Subject-Token']
    return token
...

auth_token = get_auth_token()
```

B.3 Request PM list from Placement Project

```
# get host list (list of physical machines)
def get_pm_list(auth_token_pm_list):
    res = requests.get('http://controller:8778/resource_providers',
                       headers={'content-type': 'application/json',
                                'X-Auth-Token': auth_token_pm_list
                                },
                       )
    resource_provider_response = json.loads(res.text)
    return resource_provider_response
...

resource_list = get_pm_list(auth_token)
```

B.4 Request Available PM list from Nova-Compute

```
# get a list of services from nova to check which nodes are up and
    running
def get_running_nodes(auth_token_running_nodes):
    res = requests.get('http://controller:8774/v2.1/os-services',
                       headers={'content-type': 'application/json',
                                'X-Auth-Token':
                                    auth_token_running_nodes,
                                'OpenStack-API-Version': 'compute
                                    2.88'
                                })
    res_parsed = json.loads(res.text)
    status = res_parsed['services']

    return status
...

status_nodes = get_running_nodes(auth_token)
```

B.5 Get Compute Node Status

```
pm_count = 0
status_compute_nodes = {}
for i in range(len(status_nodes)):
    if status_nodes[i]['binary'] == 'nova-compute':
        status_compute_nodes[pm_count] = status_nodes[i]
        pm_count += 1
```

B.6 Get Inventory Items

```
for resource_provider in resource_list['resource_providers']:
    is_up = check_compute_node_up(resource_provider['name'],
        status_compute_nodes)
    pm_id = pm_count_temp
    if is_up:
        pm_mapping.append(resource_provider['name'])
        inventory_items = get_inventory_items(resource_provider,
            auth_token)
        usage_items = get_usage_items(resource_provider, auth_token
            )
        inventories = inventory_items['inventories']
        total_cpu = (inventories['VCPU']['total'] -
                    inventories['VCPU']['reserved']) * inventories
                        ['VCPU']['allocation_ratio']
        total_mem = (inventories['MEMORY_MB']['total'] -
                    inventories['MEMORY_MB']['reserved']) *
                        inventories['MEMORY_MB']['allocation_ratio
                        ']
        total_dsk = (inventories['DISK_GB']['total'] -
                    inventories['DISK_GB']['reserved']) *
                        inventories['DISK_GB']['allocation_ratio']
        usages = usage_items['usages']
        used_cpu = usages['VCPU']
        used_mem = usages['MEMORY_MB']
        used_dsk = usages['DISK_GB']
        free_cpu = total_cpu - used_cpu
        free_mem = total_mem - used_mem
        free_dsk = total_dsk - used_dsk
```

B.7 Prepare and Send `PMSTAT` Message

```
send_pmstat_message(accman_socket, pm_count_temp, pm_id,
                    total_cpu, total_fpga, total_mem, total_dsk,
                    free_cpu, free_fpga, free_mem, free_dsk)

# prepare and send PMSTAT message ,
def send_pmstat_message(par_accman_socket, par_pm_count, par_pm_id,
               par_total_cpu, par_total_fpga, par_total_mem,
                   par_total_dsk,
               par_free_cpu, par_free_fpga, par_free_mem, par_free_dsk
               ):

    # This message is the one we send to the ACCLOUD-MAN
    pmstat_message = 'PMSTAT:' + \
               'PMCOUNT=' + str(int(par_pm_count)) + \
               ',PMID=' + str(par_pm_id) + \
               ',PIDLE=16,PIDLEPERCORE=1,PCPUCORE=8,PFPGAREG=1' + \
               ',TOTCPU=' + str(int(par_total_cpu)) + \
               ',FRECPU=' + str(int(par_free_cpu)) + \
               ',TOTFPGA=' + str(int(par_total_fpga)) + \
               ',FREFPGA=' + str(int(par_free_fpga)) + \
               ',TOTMEM=' + str(int(par_total_mem)) + \
               ',FREMEM=' + str(int(par_free_mem)) + \
               ',TOTDSK=' + str(float(par_total_dsk)) + \
               ',FREDSK=' + str(float(par_free_dsk)) + \
               ',TOTBWD=100.0,FREBWD=100.0' + \
               ';'
    par_accman_socket.sendall(bytes(pmstat_message, 'utf-8'))
```

B.8 Get Flavor List

```
flavor_list = get_flavor_list(auth_token)
...

def get_flavor_list(par_auth_token):
    # Get flavors and populate the flavor list with the flavor
        class objects
    res = requests.get('http://controller:8774/v2.1/flavors/detail
        ',
                        headers={'content-type': 'application/json',
                         'X-Auth-Token': par_auth_token,
                         'OpenStack-API-Version': 'compute 2.88'
                         })
    json_response = json.loads(res.text)
    flavor_list_temp = []
    for flavor in json_response['flavors']:
        dummy_dict = {
            'name': flavor['name'],
            'id': flavor['id'],
            'ram': flavor['ram'],
            'disk': flavor['disk'],
            'vcpus': flavor['vcpus']
        }
        flavor_list_temp.append(dummy_dict)
    return flavor_list_temp
```

B.9 Get Image List

```
image_list = get_image_list(auth_token)
...

def get_image_list(par_auth_token):
    # Get images and populate the image list with the image class
        objects
    res = requests.get('http://controller:9292/v2/images',
            headers={'content-type': 'application/json',
                     'X-Auth-Token': par_auth_token
                     })
    json_response = json.loads(res.text)
    image_list_temp = []
    for image in json_response['images']:
        dummy_dict = {
            'name': image['name'],
            'id': image['id'],
            'min_ram': image['min_ram'],
            'min_disk': image['min_disk']
        }
        image_list_temp.append(dummy_dict)
    return image_list_temp
```

B.10 Decision Message Handler

```
decisn_msg_handler(accman_msg_parsed, auth_token, image_list,
    flavor_list, server_list,
                           server_count, pm_mapping)
def decisn_msg_handler(par_accman_msg_parsed, par_auth_token,
    par_image_list, par_flavor_list, par_server_list,
                           par_server_count, par_pm_mapping):
     decisn_req_id = get_req_id_decisn_msg(par_accman_msg_parsed)
     decisn_pm_id = get_pm_id_decisn_msg(par_accman_msg_parsed)
     sw_id = get_sw_id_decisn_msg(par_accman_msg_parsed)

     server_info = send_openstack_server_create_msg(par_auth_token,
         decisn_req_id, decisn_pm_id,
                           par_image_list, par_flavor_list,
                               par_server_list)
     par_server_count += 1
```

B.11 VM Creation Handler

```
def send_openstack_server_create_msg(par_auth_token, par_server_id,
    par_pm_id,
                                     par_image_list,
                                        par_flavor_list,
                                     par_server_list):
    host_name = ''
    if par_pm_id == '1':
        host_name = 'compute1'
    elif par_pm_id == '2':
        host_name = 'fpga1'
    server_name = 'ServerName' + str(par_server_id)
    server_req_temp = {'adminPass': 'ADMIN_PASS',
                       'name': server_name,
                       'imageRef': par_image_list[0]['id'],
                       'flavorRef': par_flavor_list[0]['id'],
                       'host': host_name,
                       'networks': 'auto'
                       }
    server_request = {'server': server_req_temp}
    server_request_json = json.dumps(server_request)
    res = requests.post('http://controller:8774/v2.1/servers',
                        headers={'content-type': 'application/json
                            ',
                                 'X-Auth-Token': par_auth_token,
                                 'OpenStack-API-Version': 'compute
                                    2.88'
                                 },
                        data=json.dumps(server_request))
    creation_response = json.loads(res.text)
    par_server_list[par_server_id] = creation_response['server']['
        id']
    return creation_response
```

B.12 Depart Message Handler

```
depart_msg_handler(accman_msg_parsed, auth_token, server_list)
...

def depart_msg_handler(par_accman_msg_parsed, par_auth_token,
    par_server_list):
     depart_req_id = get_req_id_depart_msg(par_accman_msg_parsed)
     send_openstack_server_delete_msg(par_auth_token,
         par_server_list[depart_req_id])

def send_openstack_server_delete_msg(par_auth_token, server_id):
     res = requests.delete('http://controller:8774/v2.1/servers/' +
         server_id,
                              headers={'content-type': 'application/
                                  json',
                                        'X-Auth-Token': par_auth_token,
                                        'OpenStack-API-Version': '
                                            compute 2.88'
                                        },
                              )
```

B.13 VM Deletion Handler

```
def send_openstack_server_delete_msg(par_auth_token, server_id):
     res = requests.delete('http://controller:8774/v2.1/servers/' +
         server_id,
                    headers={'content-type': 'application/json',
                              'X-Auth-Token': par_auth_token,
                              'OpenStack-API-Version': 'compute 2.88'
                              },
                    )
```