A Framework for Private Cloud Infrastructure Monitoring

By

Selamawit Belete

A Thesis Submitted for Partial Fulfillment of the Requirements for the Degree of Master of Science in

Telecommunication Engineering

School of Electrical and Computer Engineering

Addis Ababa Institute of Technology

Advisor: Dr. Mesfin Kifle

November 2018

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Telecommunication Engineering Graduate Program

Approval of the Thesis

Submitted By: - Selamawit Belete

Thesis Title: A Framework for Private Cloud Infrastructure Monitoring

Date of Submission: October 16, 2018

The final reading approval of the thesis is granted by:

Mesfin Kifle (PhD)	_____
Advisor	Signature
Surafel Lemma (PhD)	_____
Evaluator	Signature
Ephraim Teshale (PhD)	_____
Evaluator	Signature

Abstract

Cloud computing platforms consists large number of physical and virtual resources. With the increasing number of resources, cloud computing platforms management has become more and more complex. Hence, proper resource monitoring is needed. The knowledge of the environment, which we want to monitor help a lot on the monitoring process. Specifically, in private cloud the environment is known; the application which run on top of the infrastructure and the end users are predetermined. Relatively, this knowledge did not consider in private cloud monitoring systems so far. In monitoring systems, it is a well-known practice to define a static threshold as the alert condition, but it would emit some important information if the alert threshold is set to a progressive value (dynamic).

This research paper primarily reviews different papers, white papers books and websites to understand state-of-the-art private cloud infrastructure monitoring and current monitoring practices. Then, a new framework is proposed which incorporate private cloud environment knowledge. Next to that, the solution has validated using actual cloud resource utilization, bitbrain workload trace data which is collected from a real cloud environment. This research work verified that the environmental knowledge of private cloud helps to implement dynamic threshold which help to increase resource utilization efficiency and QoS through elasticity.

The new proposed framework has two contributions; first it allows private cloud owners to get private cloud-based monitoring system for their cloud environment and researchers in the area will get a better insight about private cloud monitoring.

Keywords: *Private cloud monitoring, cloud monitoring framework, Infrastructure monitoring*

Table of Contents

Acknowledgment

It has not always been easy to start something and finish it successfully, but with the will of God everything is possible. Hence, I would like to thank my God for giving the opportunity, strength, knowledge and ability to undertake this research study and to persevere and complete it successfully. Without His blessings, this all achievement would not have been possible. Next to God, my amazing mentor and advisor Dr. Mesfin Kifle has always been there providing invaluable guidance, inspiration and suggestions. I can't thank you enough for your generosity and incredible support. The third acknowledgement goes to my company, I would like to thank ethio telecom for giving me this chance. I would also like to express my gratitude to AAiT, electrical and computer engineering department, specifically my teachers for their support and guidance. The last but not the least, I would like to thank my families, my mother, my father, my sisters and my brothers, thank you for your unconditional love and support. You always help me to get through all difficult times and I couldn't imagine anything I have in life without you by myside so I always grateful for that and love you.

List of Tables

List of Figures

Acronyms

API	Application Programing Interface
Avg	Average
BPMN	Business Process Model and Notation
CA	Collection Agent
CMC	Cloud monitoring Checks
CPU	Central Processing Unit
CT	Current Threshold
CU	Current Utilization
DaaS	Desktop as a Service
DB	Data Base
EMA	external monitoring architecture
FA	Federation Agent
HLD	High Level Design
HT	High Threshold
I/O	Input/ Output
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IMA	Internal Monitoring Architecture
IMM	Infrastructure Monitoring Manager
InPs	Infrastructure Providers
IP	Internet Protocol
ISO	International Organization for Standardization
IT	Information Technology
KPI	Key Performance Indicator
LLD	Low Level Design
LT	Low Threshold
P2P	Point-to-Point
PaaS	Platform as a Service
PCIM	Private Cloud Infrastructure Monitoring

PCMONS	Private Cloud Monitoring System
PHP	Hypertext Preprocessor
PT	Previous Threshold
PU	Previous Utilization
QoS	Quality of Service
RA	Root Agent
RQ	Research Question
RTM	Run-Time Monitor
RTT	Round Trip Time
SaaS	Software as a Service
SLA	Service Level Agreement
SoaML	Service Oriented Architecture Modeling Language
SP	Service Provider
SysML	System Modeling Language
TOG	The Open Group
UML	Unified Modeling Language
VM	Virtual Machine
XMI	Extensible Markup Language

Chapter 1: Introduction

1.1. Background

This section is aiming to give some background information about primary computing paradigms: cluster, grid and cloud computing. Next to computing paradigms, cloud computing deployment models, service models and enabling technologies are discussed. Then cloud monitoring from private cloud and public cloud monitoring aspect are also argued and the final discussion is about monitoring tools.

1.1.1. Computing paradigms: Cluster, Grid and Cloud computing

A cluster is a collection of parallel or distributed computers which are interconnected using high-speed networks, such as gigabit Ethernet, SCI, Myrinet and Infiniband. These computers working together in the execution of compute intensive and data intensive tasks that would not be feasible to execute on a single computer. Clusters are used mainly for high availability, load-balancing and for compute purpose [1].

Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements [1].

Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreement. Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. The comparison of the three computing paradigms is shown in Figure 17 (check Appendix G).

1.1.2. Cloud deployment models

Two major bodies play a key role in cloud deployment models: cloud consumers and cloud providers. Cloud Consumers is a person or organization that maintains a business relationship with

and uses service from cloud providers. Cloud providers are a person, organization, or entity responsible for making a service available to cloud consumers [2].

Depending on the relationship between the provider and the consumer, a cloud can be classified into four deployment models: public, private community and hybrid cloud [3].

Public cloud: the one most commonly referred to, is owned and operated by independent vendors and accessible to the general public.

Private cloud: is an internal utilization of cloud technologies which is maintained in-house and solely accessible to internal users within an organization.

Community cloud: is shared by several organizations and supports a specific community that has shared concerns (e.g. mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.

Hybrid cloud is a combination of two or more types of clouds (private, community, or public). For example, an organization may bridge its internally operated private cloud with other public clouds together by standardized or proprietary technology in order to satisfy business needs

1.1.3. Cloud service models

Cloud computing has three main service models, these are Software as a Service, Platform as a Service and Infrastructure as a Service.

Software as a Service (SaaS): The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. Applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, apart from limited user-specific application configuration settings [2].

Platform as a Service (PaaS): The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but

has control over the deployed applications and possibly configuration settings for the application-hosting environment [2].

Infrastructure as a Service (IaaS): The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer can deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls) [2].

1.1.4. Cloud enabling technologies

Two major cloud enabling technologies will discuss here i.e. virtualization and load balancing. Virtualization is a foundational element of cloud computing. Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time sharing, partial or complete machine simulation, emulation, quality of service, and many others [4]. Today, the primary focus for virtualization continues to be on servers. However, virtualizing storage and networks is emerging as a general strategy. Results from a Gartner survey of 505 data center managers' worldwide reports that planned or in-process virtualization of infrastructure workloads increased from approximately 60 percent in 2012 to almost 90 percent in 2014. This continuing growth makes cloud computing an obvious next step for many organizations. Virtualization plays a major role in cloud computing as it provides a virtual storage and computing services to the cloud clients which is only possible through virtualization [5].

Load balancing helps in the fair allocation of computing resources to achieve a high level of user satisfaction and proper use of resources. High resource utilization and proper load balancing help minimize resource consumption [6]. Load balancing is a method that has helped networks and resources, to provide maximum throughput with minimal response time. Load balancing is performed at two levels in cloud computing [6]:

- The level of the virtual machine, the mapping is made between applications that are loaded in the cloud on the virtual machine. The load balancer assigns the requested virtual machine to physical computers, which balances the load of multiple applications from the PC.

- A host level, a mapping between the virtual machine and host resources that allow processing of several incoming application requests.

1.1.5. Cloud monitoring

Stackify define cloud monitoring in short as: the process of evaluating, monitoring, and managing cloud-based services, applications, and infrastructure [7]. Cloud is the environment where we give everything as a service over the internet and services are on-demand, elastic and scalable, so monitoring this dynamic environment is crucial. Monitoring of cloud environment is a task of paramount importance for both providers and consumers. In one side, it is a key tool for controlling and managing hardware and software infrastructures; on the other side, it provides information and Key Performance Indicators (KPIs) for both platforms and applications. The continuous monitoring of the cloud and of its SLAs (for example, in terms of availability, delay, etc.) supplies both the providers and the consumers with information such as the performance and QoS offered through the Cloud. Usually, a cloud has large number of resources on data centers which are located in different areas. Such resources must be continuously monitored, since cloud entities (e.g., SPs, InPs) need information related to these resources, mainly for two reasons. First to evaluate the status of services hosted in the cloud. Second to use the information about resources to perform control activities (e.g., allocation, migration) [8].

A. Private Vs public cloud monitoring

Public and private clouds are a cloud computing deployment models which address the same functionalities in different environment and for different purposes. Since both models are intending to give the same service models (IaaS, PaaS and SaaS), the monitoring principle also the same in private and public cloud for some extent. However, as the two models deploy in different environment for different purpose, the monitoring will also affect. In public cloud monitoring, an organization which owns the cloud is mainly worried about monitoring the services which host by the organization. The monitoring mostly aims on whether the organization utilize the resource substantially or not. Furthermore, the monitoring also focuses on whether the cloud users are getting the resource they want as per the SLAs or not [9]. Normally knowledge about the environment we want to monitor is very important. For instance, in private cloud the users, applications and policy/rules applied in the company are known since the cloud is owned and used

by the same organization. This helps to acquire a better knowledge about the environment which we want to monitor. What kind of users we have, what kind of applications we run, and what kind of policies and rules do we have to consider is known in private cloud. This research will use this environmental knowledge to design a new monitoring framework. These all are matters which directly affect our monitoring in private cloud but not deeply in public cloud, below the reasons are discussed in detail:

Users' perspectives: public cloud is a multi-tenant environment, users may lease a resource for a given period and then they go whenever they want, then others come and so on. In public cloud we do not have any permanent users hence monitoring is not considering users in the monitoring process. In the other hand, private cloud environment has defined users since the cloud itself is owned and used by the same organization. In private cloud users are the one who provision a new resource [10] as they need (in public this is done by the cloud owners) so monitoring system should notify users for new resources and available resources. Also, the monitoring system should know which resource is newly occupied by which users and what kind of monitoring parameters should apply (automatically).

Application perspectives: In public cloud, the application which own and run by the users on top of the infrastructure is indefinite, since it's belongs to the users and the monitoring process don't consider it at all. However, in private cloud since the applications are known the monitoring process can consider the applications. For example, applications' which varies resource usage frequently can monitor easily by putting check point for the change, 78% CPU utilization might be the threshold for this application at the beginning, but it might go high or lower than this because of some triggering conditions. So, monitoring system can change the threshold accordingly (dynamically) depending on the application behavior. Therefore, resource can be assigning not only because the user is requested but also because of the applications demand and based on their utilization behavior. This is one advantage we have in private cloud to enhance resource utilization.

Policy and rules: Cloud users and applications are predefined in private cloud and that makes easy to impose central monitoring policy and rules to protect resource misuses. It also helps to limit users' power over the resources since in private cloud users are the one who provision resources. Policy and rules can have in public cloud as well but it's impossible to govern specially the users with owners' policy for monitoring purpose.

B. Cloud infrastructure monitoring

Cloud infrastructure monitoring refers to the systematic process of information collecting and analyzing to track the performance and availability of cloud infrastructure components, such as servers, storage, network and virtualization - to maintain high uptimes, reliability and a good end user experience [11]. Monitoring of cloud computing infrastructures is an imperative necessity for cloud providers and administrators to analyze, optimize and discover what is happening in their own infrastructures. Current monitoring solutions do not fit well for this purpose mainly due to the incredible set of new requirements imposed by the requirements associated to cloud infrastructures [12]. Cloud infrastructure monitoring need special attention because the other two cloud service models (SaaS and PaaS) are highly dependent on the performance of the underline infrastructure and the performance should be monitored regularly. In the other hand, the infrastructure is heterogeneous in nature and highly scalable environment so, such environment makes monitoring a very challenging task and need special care.

C. A survey on cloud monitoring tools

Many cloud monitoring tools have been adopted from grid computing; and there are many commercial and open source monitoring platforms. A paper which performs a survey in cloud monitoring [13] tries to evaluate commercial and open source monitoring tools using monitoring properties: scalability, elasticity, adaptability, timeliness, automaticity, comprehensiveness, extensibility, intrusiveness, reliability, resilience, availability and accuracy. The survey result is discussed in Table 1 and Table 2.

Table 1: Commercial cloud monitoring tools

Tools	Scalability	Elasticity	Adaptability	Timeliness	Autonomicity	Comprehensiveness	Extensibility	Intrusiveness	Resilience	Reliability	Availability	Accuracy
CloudSleuth				✓						✓		
CloudHarmony				✓		✓						
Cloudstone											✓	✓
Cloud CMP											✓	✓

CloudCLimate				✓					✓		✓	
Cloudyn				✓					✓		✓	✓
Up.time				✓					✓		✓	✓
CLoudfloor				✓					✓		✓	✓
CloudCruiser				✓					✓		✓	✓
Boundary				✓					✓		✓	✓
New Relic				✓					✓		✓	✓

Table 2: Open source cloud monitoring tools

Tools	**Scalability**	**Elasticity**	**Adaptability**	**Timeliness**	**Autonomicity**	**Comprehensiveness**	**Extensibility**	**Intrusiveness**	**Resilience**	**Reliability**	**Availability**	**Accuracy**
Nagios							✓					
OpenNebula	✓		✓									
CloudStack Zenpack				✓								
Nimbus					✓							
PCMONS							✓					
DARGOS			✓				✓	✓				
Hyperic-HQ	✓					✓						
Sensu		✓					✓					

From this evaluation we can easily understand that open source solutions failed to fulfill some monitoring properties, in the other hand commercial solution perform better [13]. Especially PCMONS, the only private cloud monitoring system only fulfil extensibility.

The monitoring property which is related to this proposed solution is elasticity; elasticity is one of the monitoring properties which is not achieved by most monitoring system as we saw it earlier. This research will design a monitoring framework which can improve resource utilization and QoS of private cloud by attaining elasticity property.

1.2. Motivation

Currently cloud is a widely used computing paradigm, but it is not applied widely in Ethiopia yet. Government and non-government organization have their own data centers which are not shared and not utilized properly; we can take ethio telecom case as an example. I got the exposure to see ethio telecoms' current data centers closely while I'm studying ethio telecom data center for one of my course project. Resources are not shared among applications within the company and because of that new infrastructure resource provisioning is done while the resources are actually exist in another companies' data centers or in another applications (resources are dedicated for applications). Most organizations are not comfortable to share their infrastructures with other organizations for some known reasons. But meanwhile these organizations will change their data centers into private cloud and they should do that too. In this process, monitoring plays the major role to increase resource utilization, quality and many more. Hence, this research is motivated by the following three reasons:

- Future tendency of widely deployment of private cloud in the country.
- Enormous importance of monitoring system in such computing environment is another motivation. Because cloud computing environment is highly scalable, very virtualized, computational service are on demand and high reliability is required.
- Less attention is given by international researchers for private cloud monitoring system (I found only one research which focused on private cloud monitoring).

1.3. Statement of the Problem

Cloud computing has many characteristics and that makes cloud monitoring a challenging task with that of its predecessor grid computing [13]. Many papers argue that additional effort is needed towards native cloud monitoring system that is because most monitoring systems used in cloud are directly adopted from grid computing [13]. Both grid and cloud have many things in common, but cloud is more scalable, has huge resources and many users while grid is limited with a group of users and less scalable too.

Many scientific papers are written to enhance current cloud monitoring systems, but private cloud monitoring is overshadowed by pubic cloud monitoring. A research has been done to introduce a private cloud monitoring system (PCMONS), which is intending to use as a private cloud

monitoring tool. In PCMONS, users can use the monitoring tool to monitor virtual resources, the integration layer allow them to access the resources by abstracting infrastructure details [10]. In public cloud users are not able to monitor cloud resources only the cloud owners can do the monitoring; but PCMONS can allow the users to monitor the resources. Having this in mind, this thesis work will try to cover what other private cloud monitoring researches miss in their work to have a better resource utilization efficiency and QoS in private cloud. Environmental knowledge is important in this regard; what kind of environment we want to monitor? In what circumstances are we trying to monitor? Accordingly, this environment knowledge is something we should take advantage to enhance resource utilization efficiency and QoS. Private cloud has its own unique environment, the application and users which use the infrastructure are known. Plus, we can easily apply any required rules because we know what kind of applications and users are using the infrastructure.

Both public and private clouds want to exploit the underline infrastructure effectively; in every possible way enhancing the utilization efficiency is worthy. One research shows that static threshold as the alert condition have its own impact on monitoring system [14]. Because it would generate unprecedented floods of false alarms if the alert threshold is set to a conservative value, while it would emit some important information if the alert threshold is set to a progressive value. The paper further argues that there is no versatile approach to defining an alert condition that suits to all the situations. It requires expert knowledge on the target system to define a well-suited alert condition [14]. In private cloud the target system (the environment which we want to monitor) is known; hence this proposed solution will implement dynamic threshold as alert condition. In the other hand, resource utilization in cloud is dynamic in nature so if we are able to monitor this dynamic utilization behavior with dynamic threshold; we can ensure better resource utilization efficiency and QoS.

Thus, the aim of this research paper is to reveal new insight by enhancing current private cloud infrastructure resource utilization efficiency, QoS and decrease false alarms using dynamic threshold. To do that, a new framework will be introduce which can consider the private cloud environmental knowledge; Figure 1 shows comprehensive view of the statement of the problem

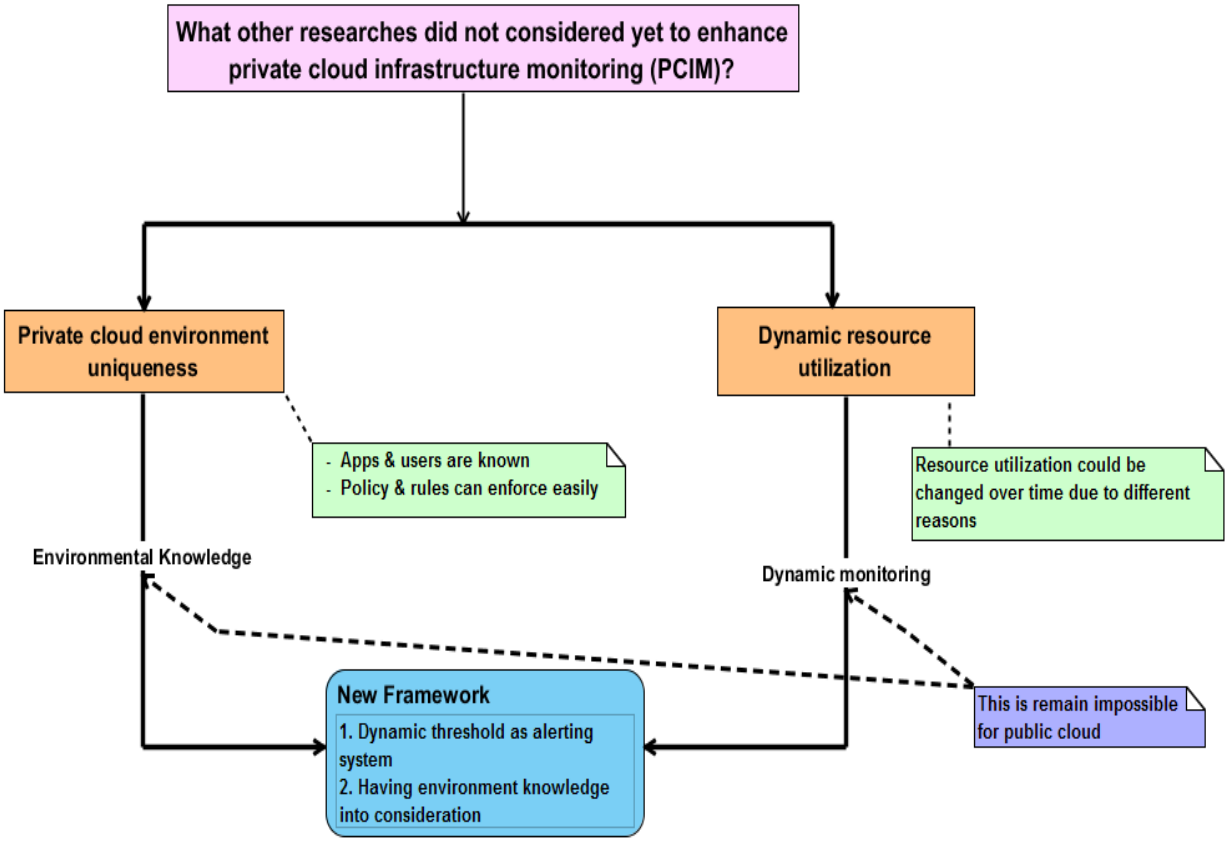


Figure 1: Statement of the problem

Hypothesis and research questions

Hypothesis: It's possible to enhance resource utilization efficiency and QoS of private cloud by using dynamic threshold as system alert condition and automatic elasticity to manage resource utilization.

RQ1: How can we enhance existing resource utilization efficiency and QoS in private cloud monitoring by using dynamic threshold as system alert condition?

RQ2: How dynamic threshold in private cloud monitoring can be affected by private cloud owner's business behavior and policies/ rules?

1.4. Objective

1.4.1. General Objective

The general objective of this research is to enhance resource utilization efficiency and QoS of private cloud by deploying dynamic threshold as system alert condition.

1.4.2. Specific Objective

The following specific objectives will help to achieve the general objective of the research

- Understand state-of-the art cloud monitoring.
- Understand cloud infrastructure monitoring.
- Understand cloud infrastructure deployment platforms.
- Understand the impact of dynamic threshold over private infrastructure monitoring.
- Develop a new framework.
- Evaluate the framework.

1.5. Method

To fulfill the general and specific objectives of this research, different methods are used:

A systematic literature review

From the beginning of the research up to the end different (many as possible) published papers, white papers, books and official web sites will review. The literatures help to have better understanding about the area, they also guide how the research should go to achieve the objective of the research and how similar works are done so far.

Tools

To design the new framework, this research uses visual paradigm (version 14.0) designing tool. Visual Paradigm is a software tool designed for software development teams to model business information system and manage development processes. Visual Paradigm supports key industry modeling languages and standards such as Unified Modeling Language (UML), SysML, SoaML, BPMN, XMI, etc. It offers complete tool-set software companies need for requirements capturing, process analysis, system design, database design, and etc [15]. The other tool this research used to implement and evaluate the proposed solution is NetBeans [16]. NetBeans is an open-source

integrated development environment (IDE) for developing with Java, PHP, C++, and other programming languages. NetBeans is also referred to as a platform of modular components used for developing Java desktop applications. The other and very important thing that used on the evaluation is BitBrain data, BitBrain [17] is a cloud service provider company and we use this company data to evaluate the proposed system (BitBrain will explain in detail in Chapter 5).

1.6. Scope and Limitation

This thesis work is intended to enhance current private cloud infrastructure monitoring solutions in order to have a better resource utilization and QoS. Hence, the research is not aiming to develop a new full-fledged monitoring system from scratch for private cloud. Rather this research tries to enhance the already exist one by applying dynamic threshold in the system alert condition. Monitoring data collection agents is not the focus of this work, the focus will be on federation and root agents (see Chapter 4 for more information about collection, federation and root agents).

The new proposed solution will not test in real private cloud environment since such environment does not exist in Ethiopia. Beside the solution does not take any specific company/ organization as a case.

1.7. Contribution

This research intends to enhance resource utilization efficiency and QoS of current private cloud monitoring practice by introducing a new framework from a new perspective. The framework could help:

- Private cloud owners will have private cloud-based monitoring system to monitor their infrastructure.
- Researchers in the area could use the framework to develop cloud native private cloud monitoring systems.
- To take advantage of cloud computing environment, ethio telecom is on the way to adopt private cloud in the near feature. Monitoring in cloud environment is very important. However, there are no enough private cloud-based monitoring system (as of my knowledge only one private cloud monitoring has introduced so far). Hence, this research work will contribute a lot regarding monitoring this cloud environment. This proposed solution is

specifically design for private cloud and the environment unique characteristics has considered in the monitoring process. Private cloud is expensive with that of other cloud deployment models, hence frequent provisioning of new resources is not advisable therefore resource utilization should be efficient. Thus, this proposed framework will help ethio telecom to realize better resource utilization efficiency and QoS.

1.8. Thesis Organization

The rest of this research paper is organized as follows: Chapter 2 will be a systematic literature review, literatures will discuss in four categories which are cloud infrastructure, cloud monitoring, cloud monitoring properties, cloud infrastructure monitoring and dynamic threshold. Chapter 3 is related work, in this Chapter the most related papers for this research will discuss. Chapter 4 is about system design part. Chapter 5 is evaluation; the newly proposed solution will evaluate using evaluation tools and evaluation result will discuss. Chapter 6 is all about conclusion, recommendations and future works, and finally there will be Reference and Appendix.

Chapter 2: Literature Review

2.1. Overview

Many papers have been written regarding cloud computing, infrastructure as a service and monitoring; in this research different literatures and white papers are reviewed which are directly related to this paper work. The papers are classified and discussed in four categories: the first category is talking about cloud infrastructure, the second category discusses papers regarding cloud monitoring, the third and fourth category is about cloud monitoring properties & parameters and cloud infrastructure monitoring respectively.

2.2. Cloud Infrastructure

Cloud infrastructure refers to the hardware and software components which are servers, virtualization, storage, network, management, security, and backup & recovery [18].

CDW LLC [19] in their white paper they discussed about cloud infrastructure with respect to cloud deployment models: whether an organization chooses to move its infrastructure to a private or public cloud depends on many variables, namely cost, security and compliance requirements. If all things were equal, every organization would probably choose for a private cloud, the most secure of the bunch however, it is also the most expensive. With a private cloud, organizations pay for a computing infrastructure dedicated solely to their core focus. Private clouds can be hosted on the organization's premises or by an IaaS provider. But in either case, they don't share resources with other tenants. This is a viable option for enterprises that must cooperate with specific compliance, auditing or governance regulations. The paper further discusses about services which we get from IaaS. For instance, compute as a service: provides compute capacity that includes servers, operating system access, firewalls, routers and load balancing on demand. Desktops as a service: DaaS is an IaaS cloud created solely for hosting and serving virtual desktops. Storage as a service: Storage is one of those necessities that only grow over time. It can be a constant struggle to maintain enough storage capacity and manage it effectively. Networking as a service: This is the newest entrant in the IaaS category. The idea is to offer networking resources on demand to support virtual networks resources such as firewalls, load balancing and WAN acceleration services.

2.3. Cloud Monitoring

Monitoring of cloud is a task of paramount importance for both cloud providers and consumers. Different researches contribute a lot in this area to create a better insight about cloud monitoring; below about five papers will discuss.

Giuseppe Aceto et al [20] tried to provide background and definitions for key concepts of cloud monitoring. They put cloud monitoring in two abstraction levels, the first abstraction level is high-level monitoring which is related to information on the status of the virtual platform. This information is collected at the middleware, application and user layers by providers or consumers through platforms and services operated by themselves or by third parties. The other abstraction level is low-level monitoring, this abstraction level is related to information collected by the provider and usually not exposed to the consumer, and it is more concerned with the status of the physical infrastructure of the whole cloud. The authors further discuss about monitoring test, and they categorized in to two computational based and network based. Network-based tests are related to the monitoring of network-layer metrics. This set includes round-trip time (RTT), jitter, throughput, packet/data loss, available bandwidth, capacity, traffic volume, etc. In the other hand computation-based tests are operated by the provider or sometimes demanded to third parties.

Guilherme Da Cunha Rodrigues et al [8] in the other hand define monitoring abstraction level in a different way as monitoring views, the view of resources depends on who wants to obtain the information, i.e., InPs, SPs, or customers. InPs are the owners of the infrastructure, and normally are concerned about the infrastructure's correct operation and efficient utilization. SPs, in general, can obtain information about the virtual layer, such as response times and latencies observed across different elements of the platform, and how it relates to the performance observed by customers. Customers, in turn, can see information about the high-level application/services they are using. The other concept they discussed is monitoring focus; Monitoring focus can be divided using two methods: by cloud model or by goal. The first one refers to the service model: SaaS, PaaS, or IaaS. The second refers to the goal/objective of the monitoring performed by InPs, SPs, or customers (e.g., SLA, QoS).

Studies have been surveying cloud monitoring solutions and state-of-the-art, and some of them did classification on them. These taxonomies help us to have a better insight about cloud monitoring.

And here we discuss two recent researches which conduct their research focusing on providing monitoring solution taxonomies. Hassan Jamil Syed et al [9] presents a high-level general taxonomy of cloud monitoring. After comprehensively reviews the state-of-the-art of cloud monitoring solutions they propose a thematic taxonomy and conduct comparative analysis of State-of-Art of cloud monitoring solutions. Jose M. Alcaraz Calero et al [21] also follow the same approach; however, they go a little bit deeper than the first one. For instance, the first paper did the taxonomy as follows:

- Monitoring perspective: cloud user's perspective, cloud provider's perspective
- Monitoring purpose: billing, performance monitoring, efficient use of resource
- Type of cloud: private cloud, public cloud
- Monitoring architecture: centralized architecture, distributed architecture
- Communication model: push, pull, hybrid
- Monitoring platform overhead: considered, not considered
- License: open source, non-open source
- Scalability: considered, not considered

And then they did comparative analysis on commercial and open source monitoring tools by taking the taxonomies as a metrics. The second paper took monitoring perspective and adds architectural view on it: cloud user's perspective architecture and cloud provider's perspective architecture. First, they provide two broad taxonomies; internal monitoring architecture (IMA) which is more suitable for cloud provides and external monitoring architecture (EMA) is more fitted for cloud users. These two categories further classified into traditional IMA, extended IMA, extended and adaptive IMA, concentrated EMA and sparse EMA, then the author rate these five architectures based on performance, usage of resource, security and scalability. The decomposition is not over yet, those five architectures further classify into 12 different possible architecture. Finally, the authors recommend the best architectures out of the 12 architectures from cloud consumers, cloud providers, virtual machines and physical machines perspectives.

Different researcher introduces different approach to monitor cloud environment. For instance, Simin Caia et al. [22], introduce a new monitoring design which can solve scalability problem. They enhance current Openstack framework by adding auto-scaling functionality using Data Aggregation Taxonomy (DAGGTAX). This design is suitable for video streaming and other

resource intensive platforms.In the other hand Manoj Kesavulu et al [23] proposes an approach to extract end-user usage of cloud services from their interactions with the interfaces provided to access the services called User-level Usage Monitoring. They want to propose this solution because monitoring solutions introduced by the time has not considered user-level usage monitoring and its implications on the service and infrastructural resource usage. The authors further argued that new services, applications and the monitoring solutions has need to follow the principles set forward by the cloud standards like ISO and TOG. Both standards specify different monitoring activities and capabilities; the authors summarize and present seven principles for monitoring in the cloud and according to them ISO standard did not cover one principle while TOG cover only three out of seven. The authors did comparative analysis of ten monitoring tools based on three monitoring levels, status of user-level monitoring and monitoring techniques.

2.4. Cloud Monitoring Properties

Giuseppe Aceto et al [13] in their survey paper they present cloud monitoring from four key perspectives: monitoring properties, basic concepts, need for monitoring and open issues & future directions (see Figure 2). Twelve cloud monitoring properties has discussed and by taking those properties as a metric eight open source and eleven commercial monitoring tools are evaluated. According to the analysis they perform; resilience, reliability, availability and accuracy have not been addressed by open source monitoring solutions.

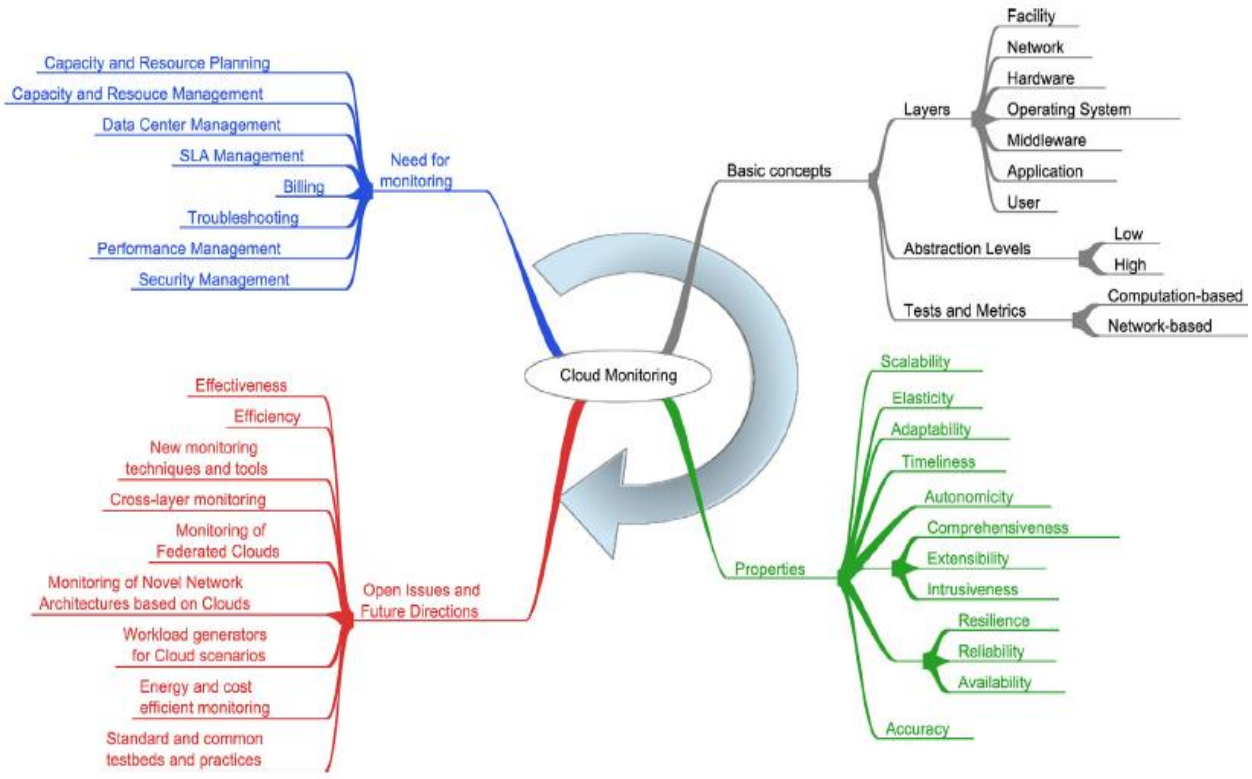


Figure 2: Cloud monitoring properties and parameters

2.5. Cloud Infrastructure Monitoring

Juan Gutierrez-Aguado et al [12] state that current monitoring tools assumes monitored resources will remain always with the same IP address then, if such address is not responding, it is assumed that resources are shutdown or failing. However, this is not true at all in cloud infrastructures where IP addresses are being highly reused and dynamically assigned to different VMs in a matter of seconds. These facts make difficult for cloud providers to implement effective monitoring solutions for their infrastructure. They argued that their work is the first attempt to integrate both the control and data planes of a cloud computing infrastructure with an existing monitoring tool to fit in the monitoring of the completely new life-cycle associated to virtual cloud infrastructures. Two components are added in current monitoring tools in order to solve the problem Infrastructure Monitoring Manager (IMM) and Cloud monitoring Checks (CMC). IMM captures the messages from the communication middleware; they are processed producing a set of actions to dynamically adapt the configuration of the monitoring architecture with the up-to-date information (changes in

the topology, changes in the IP re-assignation, and changes in the VM states). CMC enables the monitoring tool to communicate with any VMs of the cloud infrastructure regardless its location and/or owner. The solutions fulfill all the requirements identified for public cloud infrastructures.

Mingwei LinQ et al [24] they concern was with the increasing number of physical computer nodes, cloud computing platforms become more and more complex. Hence, a better data collection protocol is needed than the very known ones i.e. push, pull and hybrid. The push protocol can be categorized into the periodic push protocol and the event-driven push protocol respectively. In the periodic push protocol, the monitored nodes send the status value of resource to the monitoring node every time interval, while, in the event-driven push protocol, the monitored nodes end the status value of resource to the monitoring node if the status value of resource changes and the change degree is larger than a threshold. Again, the pull protocol can be categorized into the periodic pull protocol and the event-driven pull protocol, respectively. In the periodic pull protocol, the monitoring node sends the request to the monitored nodes every time interval, and then the monitored nodes send the status value of resource back to the monitoring node.

In the event-driven protocol, the monitoring node sends the request to the monitored nodes if the change degree of the status value of resource is larger than a threshold, and then the monitored nodes send the status value of resource back to the monitoring node. Regarding data coherence and communication overhead push protocol perform high on both parameters and pull protocol is low. Hybrid protocol improve the perform with high data coherence and low communication head but one drawback in hybrid protocol was there is no upper limit and lower limit to control the dynamic time interval and then the dynamic time interval may be too small or too large. However, in the new introduced protocol i.e. adaptive push protocol the authors introduce the upper limit and lower limit to protect the dynamic time interval from being too big or too small by using exponential weighted moving average to update the dynamic threshold. Their result show that our proposed hybrid push protocol performs better than previous data delivery protocols in terms of the data coherence and the communication overhead.

Dirk Paessler [25] in this white paper ‘monitoring of a private cloud’, the author tries to see private cloud monitoring from two perspective; from users’ perspective and server perspective and the author argue that network monitoring is a foundation for private cloud planning. A private cloud – like any other cloud – depends on the efficiency and dependability of the IT infrastructure. The

capacity requirements of each application in the planning stages of the cloud must look in order to calculate resources to meet the demand. These application requirements have an implication on the monitoring as well. Despite all these facts other paper in the other hand introduces private cloud monitoring system, PCMONS which is introduced by Shirlei Aparecida de Chaves et al [10] this monitoring tool is specifically design for private cloud environment. The authors argue that since private cloud is used by a single company and the common goal is to take advantage of the existing facilities, the monitoring system should integrate seamlessly into organizations' existing management, infrastructure and operations. Leverage the installed software and hardware base as well as the skills and experience of IT administrators are also important things to consider during private cloud monitoring system architecture.

2.6. Dynamic Threshold

This research is done by Francisco J.Muniz and Osvaldo S. F. deCarvalho; The paper describes the design and implementation of a dynamic threshold policy (to be used to classify local node availability, for load-balancing purpose) [26]. The experiments were built upon the Extended Gradient algorithm, running over Linux/X86 computing nodes. The local processor availability is classified as lightly-loaded or heavily-loaded by comparing each node's current CPU 'busyness'-level (High Threshold (HT) and a Low Threshold (LT)). The authors argue that this static threshold values affect the load balancing process whether the threshold has low or high values. As the thresholds have very low values, all the nodes in the cluster will reach the HT too soon; consequently, frequent load balancing activity will perform, and that activity may degrade the performance significantly. In the other condition, when the thresholds have very high values, but in doing so, effective load-balancing will not be carried out when the system turns out to be lightly-loaded.

Therefore, choosing the appropriate values for thresholds can be quite complicated, since they depend not only on the application features, e.g. process granularity, but also on the system characteristics, such as: latency and bandwidth; number of available processors; relative processing speed among processors. In this work a real implementation (including dynamic load-balancing mechanism and application) is used to show that dynamic threshold strategies can work in practice. According to the authors, a substantial improvement in machine performance, in terms of execution time - on average 4.6% shorter, was achieved in favor of the dynamic threshold

policy. The percentages of improvement were derived by comparing the code execution times for both dynamic and static threshold policies.

The other research work is written by Xinkui Zhao, Jianwei Yin, Chen Zhi and Zuoning Chen. This paper is discussed about SimMon; SimMon is a project that builds on a cloud computing environment simulation toolkit named CloudSim. SimMon offers three novel features [14]. First, it is comprehensive. SimMon considers several functional and performance-related issues, which comprehensively covers the requirements on the key stages of a cloud monitoring system. Second, it is scalable. The scale of simulated monitoring systems and the size of the collected data are changeable. Third, it supports fully customization. SimMon is designed with a loose-coupled principle, thus users can customize strategies in different stages of a cloud monitoring system with SimMon. The authors raised a very important point regarding threshold values in monitoring system, and this the key reason why SimMon is discussed in this specific research work. The authors argue that using static threshold as the alert condition is a well-known practice. In this practice, the alert function is triggered when the value of corresponding metric exceeds the threshold. The value of the threshold has an important impact on monitoring systems. It would generate unprecedented floods of false alarms if the alert threshold is set to a conservative value, while it would emit some important information if the alert threshold is set to a progressive value. Unfortunately, there is no versatile approach to defining an alert condition that suits to all the situations. It requires expert knowledge on the target system to define a well-suited alert condition. This research paper will work to introduce dynamic threshold as an alert condition for private cloud.

2.7. Summary

The papers discussed in this Chapter helps to understand current state-of-art and will help this research to have inclusive view about cloud monitoring, cloud infrastructure, cloud infrastructure monitoring and cloud monitoring properties. The reviewed papers are the base for cloud monitor hence, they are all taken as an input and will consider together with the new perspective which this research wants to bring.

Chapter 3: Related Work

3.1. Overview

In this Chapter, a few research papers which are more related to this specific work will discuss. The research papers are discussed from design perspective, private cloud monitoring perspective and current design evaluation trends. Private cloud is one of cloud deployment model which own and use by a single organization, Figure 3 gives a clear view about what is private cloud and the main components which needs to consider during monitoring. In private cloud users and applications are considered as part of the cloud since they are predefined and known. But, in public cloud the applications are belongs to the end users (unless it's SaaS cloud) and the users and application are unknown specially for IaaS cloud.

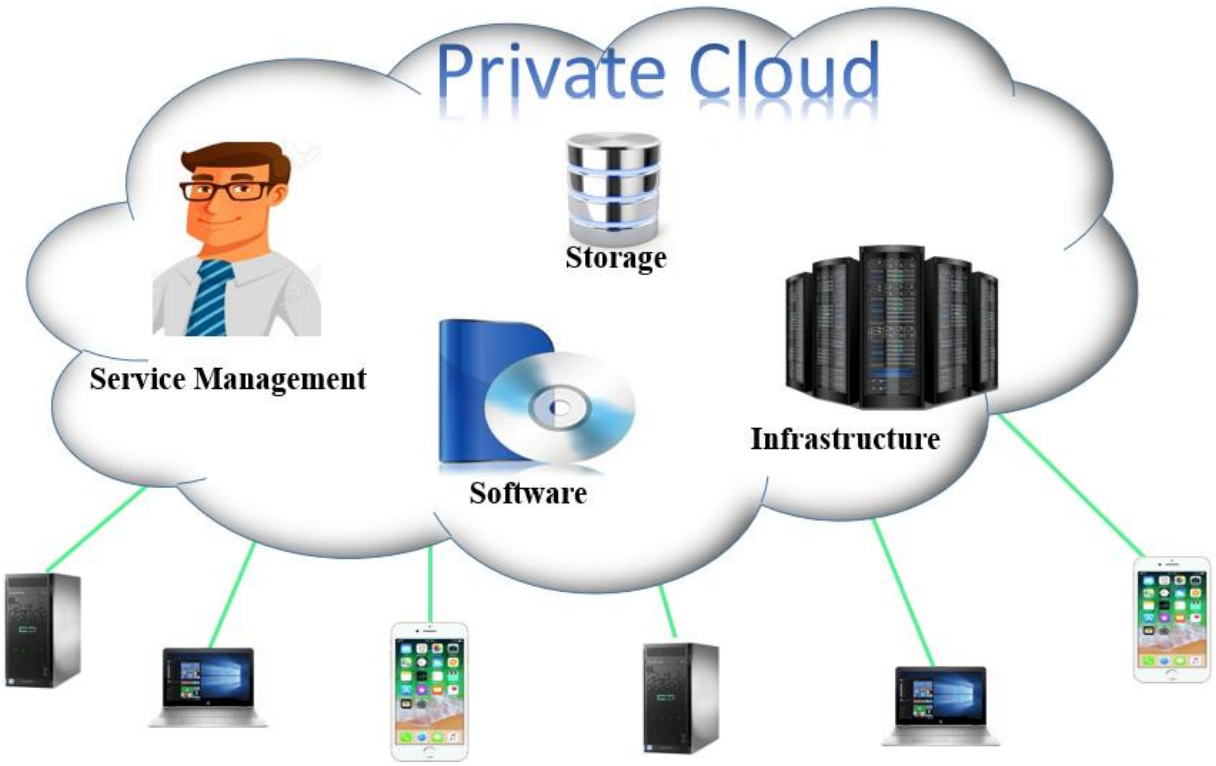


Figure 3: Private cloud

3.2. Monitoring design

Jonathan Stuart Ward and Adam Barker [27] tried to design a monitoring system called Varanus which fulfill set of monitoring requirements that presented on the paper in three categories. The first requirement category is general requirement and it's concerned about cost reduction and operate monitoring in all cloud hosts without non-cloud hosts involvement. The next part of requirement category is resource aware requirement, this requirement is stated that the monitoring tool shouldn't be operate on a dedicated VMs rather it should run on the monitored VMs without intrusiveness and in scalable manner at the same time. Fault tolerance is the third requirement category, under this category the authors point out fault, termination and elasticity shouldn't be a problem in the monitoring process. The last category is autonomic: any form of human intervention at run time shouldn't be there and autonomic computational placement is required to use the collected data for decision making.

Varanus is comprised of four components: the coordination service, the collection service, the storage service and the analysis service. The coordination service is the foundation upon which the other services operate, providing a means for the components to communicate and provides VM registration, configuration storage, decision making and agreement. The data collection service is comprised of a small daemon which operates on each monitored host which collects metrics and values and transmits them to the storage service. The storage service runs across elected (or specifically dedicated) VMs and provides a mechanism for the memory storage and processing of large volumes of time series data. The analysis service consumes data from the storage service to detect irregularities, failure, bottlenecks, plan optimizations and other user definable behavior. According to the authors they are able to introduce autonomic monitoring system which can collect and analyze monitoring data in a timely fashion and reduce the burden on human administrators. When they compare their monitoring tool with that of NAGIOS, the widely used monitoring tool, they claim that Varanus offers superior out of the box performance for cloud monitoring. Nagios can certainly be used effectively for cloud monitoring; however, this requires significant modification backed by manpower and experience and used in conjunction with automation tools such as Puppet or Chef.

In the other hand Mikyung Kang et al [28] design Run-Time Monitor (RTM) which is a system software to monitor the application behavior at runtime, analyze the collected information, and

optimize resources on cloud computing. RTM monitors application software through library instrumentation as well as underlying hardware through performance counter optimizing its computing configuration based on the analyzed data.

3.3. Private cloud monitoring

PCMONS is introduced by Shirlei Aparecida de Chaves et al [10] this monitoring tool is specifically designed for private cloud environment. The authors argue that since private cloud is used by a single company and the common goal is to take advantage of the existing facilities, the monitoring system should integrate seamlessly into organizations' existing management, infrastructure and operations. Leverage the installed software and hardware base as well as the skills and experience of IT administrators are also important things to consider during private cloud monitoring system architecture. The authors propose a three-layered architecture: infrastructure layer; integration layer and view Layer. Infrastructure layer consists of basic hardware, software, network and operating system. Integration layer is responsible for visualization environment and hypervisors to acquire infrastructural related information. The view layer is responsible for presenting the monitoring data appropriately to the type of user (here, a user represents actors such as developer, administrator or a manager, not an end-user). The authors also demonstrate the PCMONS tool in this paper using agent insertion-based monitoring methodology (for every new VM). This method creates additional overhead affecting VMs performance. The monitoring component in this paper is called as VM monitor, which injects scripts into the VMs that send useful data (for example, processor load and memory usage) from the VM to the monitoring system.

The system (PCMONS) is divided into the following modules listed below, Figure 4 shows the components.

- Node Information Gatherer: responsible for gathering local information on a cloud node and sends it to the Cluster Data Integrator.
- Cluster Data Integrator: prepares the data for the next layer and avoids unnecessary data transfers from each node to the Monitoring Data Integrator.
- Monitoring Data Integrator: Gathers and stores cloud data in the database for historical purposes and provides such data to the Configuration Generator.

- VM Monitor: This module injects scripts into the VMs that send useful data from the VM to the monitoring system.
- Configuration Generator: Retrieves information from the database, for example, to generate the necessary configuration files for visualization tools being used in the view layer.
- Monitoring Tool Server: This module is responsible for receiving monitoring data from different resources.
- User Interface: This module is helping the user to view the monitoring information and interact with other monitoring module.
- Database: Stores data needed by Configuration Generator and the Monitoring Data Integrator.

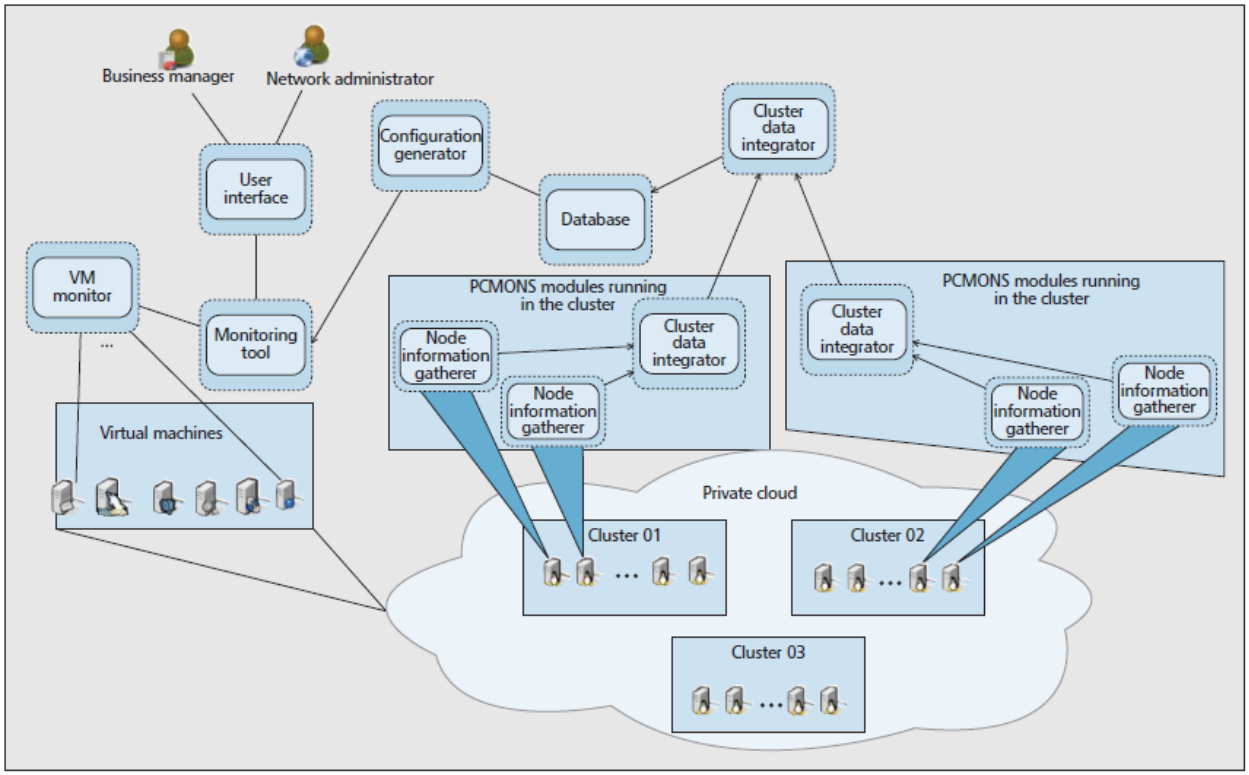


Figure 4: A typical deployment scenario for PCMONS

Papers which focus on design gives a good stand for the newly proposed design, as we can see on the first paper; they try to design a new monitoring system which specifically achieves a certain

set requirement and the way they handle the requirements by the design will help as guide for this research. private cloud monitoring (PCMONS) are motivates this research to see the private cloud monitoring in different perspective since the proposed solution by the papers misses the unique properties of private cloud which pretty much affect the monitoring and can enhance resource utilization efficiency and quality. Current monitoring solutions lack domain knowledge to monitor private cloud this is the gap what this proposal found in related papers.

3.4. Evaluation Trend

Two recent papers will be discussed here, these papers use bitbrain data (see Chapter 5 about bitbrain) to implement and evaluate their work. The first paper is "Heuristics and metaheuristics for dynamic management of computing and cooling energy in cloud data centers" [29]. In this paper, the authors try to optimize current cloud infrastructures energy consumption by using novel power and thermal-aware strategies and model. The authors used Bitbrain Workload and Power & thermal models to evaluate their work. Finally, they find out that combined awareness from both metaheuristic and best fit decreasing algorithms allow to describe the global energy into faster and lighter optimization strategies that may be used during runtime. The other paper is "An Efficient and Fair Multi-Resource Allocation Mechanism for Heterogeneous Servers" [30]. The authors argued that existing multi-resource fair allocation mechanisms, notably dominant resource fairness and its follow-up work. They try to address this issue by using server-based approach and evaluate using extensive trace-driven (Bitbrain) simulations. With this approach the authors can able to enhance resource utilization compared to the existing mechanisms.

3.5. Summary

This Chapter mainly focused on the papers which are highly related with this research work. Only one private cloud monitoring system is introduced so far, and there are two reasons for this. The first one is giving less attention for monitoring system and the second one is thinking that the already available monitoring solution are good enough for private clouds. However, suit-to-all monitoring solutions is not efficient enough for private clouds, target system (monitored environment) knowledge is very important. So, in this research work new monitoring framework is introduced which considered the environmental knowledge.

Chapter 4: Framework Design

4.1. Overview

This Chapter presents the design of the solution, the design is focused on the part which will add in this work, the design is not full-fledged monitoring solution. Rather its exclusively define and design the newly added components which able to fulfill the hypothesis which this research paper wants to proof.

4.2. Framework Design

The design part will have three sub parts; the high-level design, low level design and the algorithm data flow. Each part will discuss below.

4.2.1. High level design

According to their functionalities, we divide the nodes in cloud monitoring systems into three categories: collection agent, federation agent, and root agent [14]; Figure 5 shows the relationship between these three agents.

Root agent: A root agent is the control panel of a cloud monitoring system. It determines the scheduling algorithms and sends instructions to control the management strategies in the cloud monitoring system. Root agents locate on the top layer of the layered monitoring topology.

Federation agent: A federation agent is the actual executive for data storage and management. It receives network packets, extracts data from the packets, processes the data, and stores them persistently with the guidance of the instructions from the connecting root agent. Federation agents locate on the middle layer of the layered monitoring topology.

Collection agent: A collection agent is a program that locally collects run-time measurements from monitoring targets. Collection agents are the source of monitoring data. However, they do not persistently store the data but send them to federation agents. Collection agents are also called sensors. Collection agents locate on the bottom layer of the layered monitoring topology.

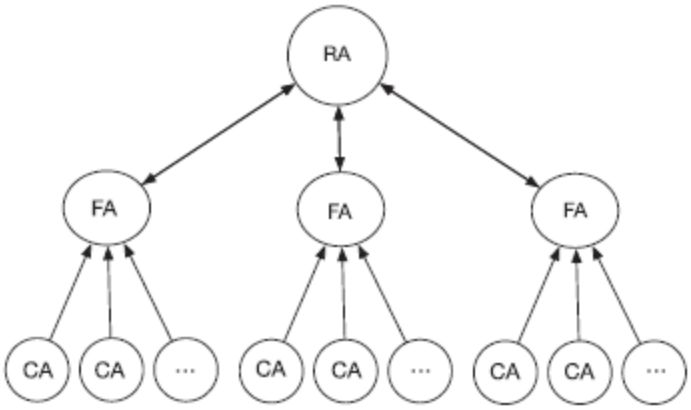


Figure 5: Topology of cloud monitoring systems: RA, FA, and CA represent root agent, federation agent, and collection agent, respectively [14].

Here, the kind of topology a monitoring system should have is discussed, but first it is good to see the kind of topologies we have in cloud monitoring: In cloud monitoring systems, there are four prominent topologies: centralized topology, layered topology, P2P topology, and hybrid topology [14].

Centralized topology: Employs a single centralized root agent to connect with and communicate with all the collection agents. There is no federation agent in a centralized topology. Only handle a limited number of collection agents for its physical and software capacity and has single-point-failure and scalability concerns.

Layered topology: several federation agents are employed to release data reception and management pressure on the root agent. Each federation agent oversees a certain bunch of collection agents. Can fulfill monitoring requirements in data centers with any scale if the number of federation agent is large enough but still only have one root agent.

P2P topology: manages all the collection agents in a totally decentralized way. All the collection agents are treated equally, and each collection agent is authorized to play all three kinds of roles (i.e., collection, federation, and root agent). This topology incurs security concerns and more additional resource cost also leads to inaccuracy on data transmission and reception.

Hybrid topology: Inherits the benefits from the structures of tree-based topology and P2P topology. The data collected by each collection agent are disseminated to all other collection agents which connected with the same federation agent.

For this research paper, a new topology is proposed by considering the above four kind of topology. A layered topology will adopt with some enhancement. Since the topology has one root agent, the root agent failures will halt the entire monitoring system. Therefore, redundant root agents and federated agent with some extent can solve the problem, Figure 6 shows the proposed topology. The solid line shows the actual connection and default active path while the broken line indicates possible connection between agents but not the default routes. In case the root agent, which the federation agent is connected through solid line fail, one of the other two root agents will take over the operation. The same can be done for each federation agents, it will consume more resources, but it will be more reliable.

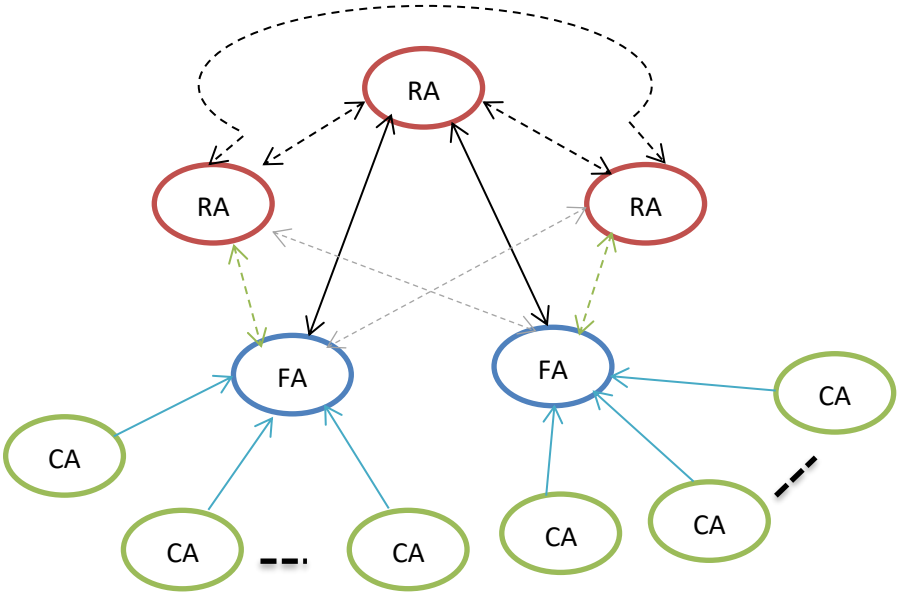


Figure 6: Proposed monitoring topology (extended layered topology)

According to the time sequences of the events, data monitoring task can separate into four stages [14]. Figure 7 shows the stages: collection agents locally collect real-time measurements, that is, data collection stage; collection agents transmit monitoring data to federation agents, that is, data transmission stage; federation agents preprocess and store the received data, that is, data storage stage; root agents manage control policies and send query requests to federation agents, that is, data management stage.

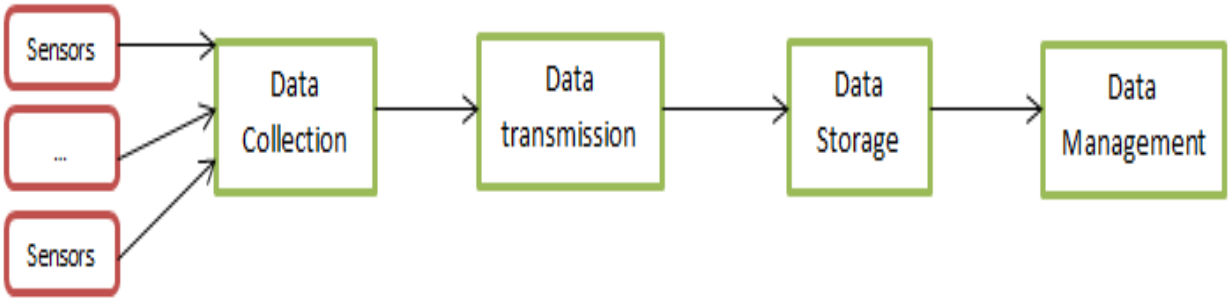


Figure 7: Four task stages in cloud monitoring [14]

It is a well-known practice to define a static threshold as the alert condition. In this practice, the alert function is triggered when the value of corresponding metric exceeds the threshold. The value of the threshold has an important impact on monitoring systems. It would generate unprecedented floods of false alarms if the alert threshold is set to a conservative value, while it would emit some important information if the alert threshold is set to a progressive value. Unfortunately, there is no versatile approach to defining an alert condition that suits to all the situations [14]. It requires expert knowledge on the target system to define a well-suited alert condition.

In order to increase resource utilization and QoS in a private cloud, we can take advantage on the threshold values we set for each monitored resource. As we saw earlier expertise knowledge on the target system will help to have a dynamic threshold that suits the situations. The threshold value we set might be affected by different parameters like the last one-hour utilization data (resent behavior), other service utilizations (because of dependency), and historical utilization data (daily/weekly/monthly data). Automatic resource provisioning can happen without human intervention and this will increase resource availability at the same time. Normally each agent (collection, federation and root) might have different approaches to do things in different monitoring schemes, and some agents can possibly affect by cloud deployment models. There is no written document/ paper found in this regard, but collection agent seems work similar in all models since its function is just collecting monitoring data from monitored nodes. In the other hand federation and root agent would affect by cloud deployment model settings. Federation agent is the one which process and stores the data which came from the collection agent. The processing is based on the collected monitoring data and the new threshold. In this research, the new threshold is calculated based on whether recent utilization data, or due to the change in other service data (because of dependency) or by daily weekly or monthly utilization data. Thus, based on the newly

calculated threshold the root agent will change the threshold for that specific service (monitored node) and allow the alarm module to have the new threshold set to take measures. Infrastructure users and applications are pretty much known in private cloud; hence we can vary the threshold depend on the application behavior. In the other hand applications might depend on one or more applications so we can determine application behavior from others as well (dependency). From daily, weekly and monthly resource utilization of a system, it's possible to determine and set a threshold. This all can work while we know the nature of the systems which we run on our infrastructure. Private cloud can give this all information, Figure 8 shows the proposed monitoring system high level design, light green shaded parts are the newly added components which help to increase QoS and resource utilization efficiency in private cloud computing.

This high-level design explains the architecture that would be used for developing this proposed framework (Figure 8). The diagram provides an overview of the entire framework, identifying the main components that would be developed for the solution and the data flows too. The new added components are the one which are highlighted in green, other components are exist in other monitoring architectures and the components and the layers (root, federation and collection agent) are directly adopted from literature [14].

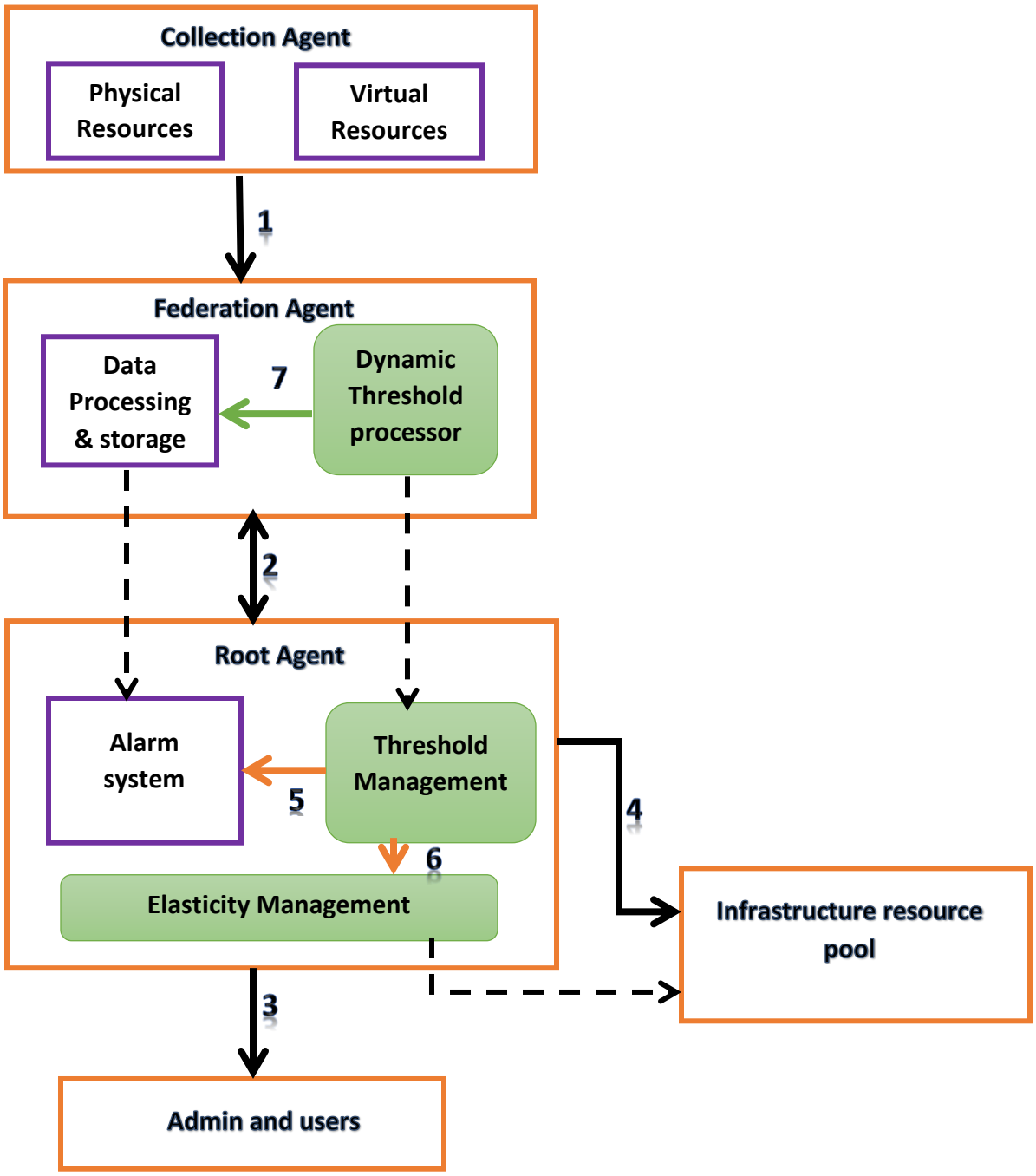


Figure 8: The proposed framework high level design

HLD description

Newly added components

Dynamic threshold processor: calculate a new threshold based on the collected utilization data and threshold calculation parameters: historical data (recent/daily/weekly/monthly), other services (dependency).

Threshold management: set the new calculated threshold to the respected service and send it to the alarm node. It also decides what kind of elasticity is needed and send elasticity request to elasticity management.

Elasticity management: If a service is using too much resource more than it needs, then the resource should release back to the resource pool or if a service is needed additional resource, then automatically resource provisioning should take; these all action would be taking care of by Elasticity management.

Data flow

1: Monitoring data flow from collection agent to federation agent

2: Processed monitoring data is passed to the alarm system, at the same time the newly calculated threshold value will also pass to threshold management node.

3: Alarms and other required monitoring information will pass to admins and users.

4: New resource provisioning request will send to the resource pool

5: New threshold value will pass to the alarm system to consider on the next alarm

6: The type of elasticity needed (upgrade/downgrade) will send to elasticity management for resource provisioning

7: New threshold value will send to data processing and storage unit

Figure 9 shows how the proposed design can be integrated with SUSE OpenStack Cloud monitoring architecture and components. SUSE OpenStack Cloud is powered by OpenStack, the leading community driven, open source, cloud infrastructure project, and packaged with SUSE Linux Enterprise Server [31].

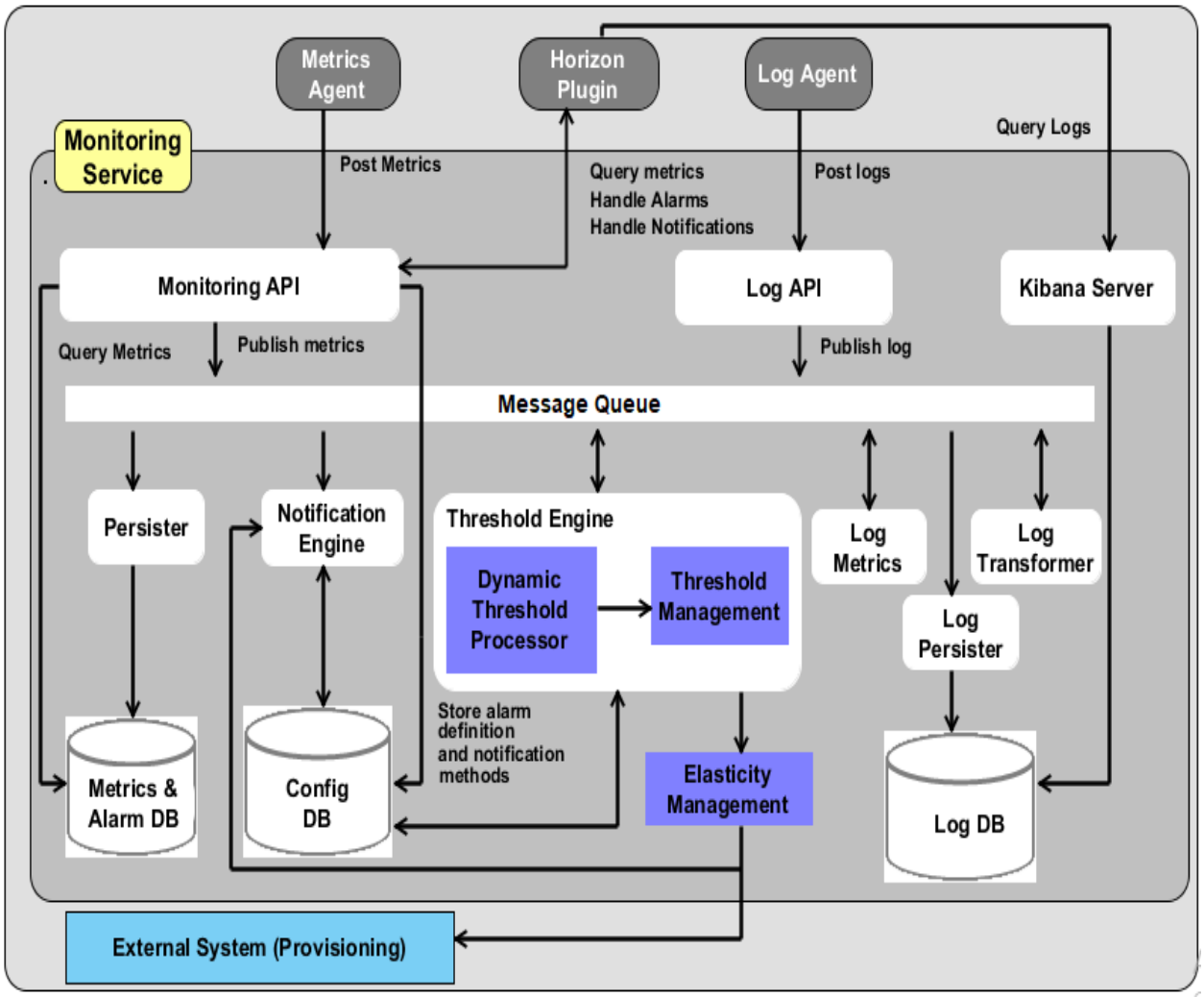


Figure 9: The new proposed solution with open SUSE monitoring architecture.

Color shaded components are the one which are proposed and added on the architecture by this research work as a solution for private cloud infrastructure monitoring. The other components of the architecture are defined below with some preliminary concept.

SUSE OpenStack

SUSE OpenStack [31] Cloud Monitoring relies on OpenStack as technology for building cloud computing platforms for public and private clouds. OpenStack consists of a series of interrelated projects delivering various components for a cloud infrastructure solution and allowing for the deployment and management of Infrastructure as a Service (IaaS) platforms.

Monitoring Service

The Monitoring Service is the central SUSE OpenStack Cloud Monitoring component. It is responsible for receiving, persisting, and processing monitoring and log data, as well as providing the data to the users. The Monitoring Service relies on Monasca, an open source Monitoring as a Service solution. It uses Monasca for high-speed metrics querying and integrates the Threshold Engine (streaming alarm engine) and the Notification Engine of Monasca.

The Monitoring Service consists of the following components:

Monitoring API: A RESTful API for monitoring. It is primarily focused on the following areas:

- Metrics: Store and query massive amounts of metrics in real time.
- Statistics: Provide statistics for metrics.
- Alarm Definitions: Create, update, query, and delete alarm definitions.
- Alarms: Query and delete the alarm history.
- Notification Methods: Create and delete notification methods and associate them with alarms. Users can be notified directly when alarms are triggered, for example, via email.

Message Queue: A component that primarily receives published metrics from the Monitoring API, alarm state transition messages from the Threshold Engine, and log data from the Log API. The data is consumed by other components, such as the Persister, the Notification Engine, and the Log Persister. The Message Queue is also used to publish and consume other events in the system. It is based on Kafka, a high-performance, distributed, fault-tolerant, and scalable message queue with durability built-in. For administrating the Message Queue, SUSE OpenStack Cloud Monitoring uses Zookeeper, a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

Persister: A Monasca component that consumes metrics and alarm state transitions from the Message Queue and stores them in the Metrics and Alarms Database (InfluxDB).

Notification Engine: A Monasca component that consumes alarm state transition messages from the Message Queue and sends notifications for alarms, such as emails.

Threshold Engine: A Monasca component that computes thresholds on metrics and publishes alarms to the Message Queue when they are triggered. The Threshold Engine is based on Apache Storm, a free and open distributed real-time computation system.

Metrics and Alarms Database: An InfluxDB database used for storing metrics and the alarm history.

Config Database: A MariaDB database used for storing configuration information, alarm definitions, and notification methods.

Log API: A RESTful API for log management. It gathers log data from the Log Agents and forwards it to the Message Queue. The SUSE OpenStack Cloud Monitoring log management is based on Logstash, a tool for receiving, processing, and publishing all kinds of logs. It provides a powerful pipeline for querying and analyzing logs. Elasticsearch is used as the back-end datastore, and Kibana as the front-end tool for retrieving and visualizing the log data.

Log Transformer: A Logstash component that consumes the log data from the Message Queue, performs transformation and aggregation operations on the data, and publishes the data that it creates back to the Message Queue.

Log Metrics: A Monasca component that consumes log data from the Message Queue, filters the data according to severity, and generates metrics for specific severities, for example, for errors or warnings. The generated metrics are published to the Message Queue and can be further processed by the Threshold Engine like any other metrics.

Log Persister: A Logstash component that consumes the transformed and aggregated log data from the Message Queue and stores it in the Log Database.

Kibana Server: A Web browser-based analytics and search interface to the Log Database.

Log Database: An Elasticsearch database for storing the log data.

Horizon Plugin: SUSE OpenStack Cloud Monitoring comes with a plugin for the OpenStack Horizon dashboard. The plugin extends the main dashboard in OpenStack with a view for monitoring. This enables SUSE OpenStack Cloud Monitoring users to access the monitoring functions from a central Web-based graphical user interface. For details, refer to the OpenStack

Horizon documentation [32]. Based on OpenStack Horizon, the monitoring data is visualized on a comfortable and easy-to-use dashboard which fully integrates with the following applications:

Kibana (for log data): An open source analytics and visualization platform designed to work with Elasticsearch.

Metrics Agent: A Metrics Agent is required for retrieving metrics data from the host on which it runs and sending the metrics data to the Monitoring Service. The agent supports metrics from a variety of sources as well as a number of built-in system and service checks.

Log Agent: A Log Agent is needed for collecting log data from the host on which it runs and forwarding the log data to the Monitoring Service for further processing. It can be installed on each virtual or physical server from which log data is to be retrieved.

4.2.2. Low level design

As we saw in the high-level design the newly added components have its own functionalities on the two monitoring layers; federation and root agents. Let us see what will happen in each component.

Dynamic threshold Processor

As discussed earlier the dynamic threshold is depend on three main things; the first one is the historical utilization of a given node could determine the future resource utilization. Historical data utilization refers four things: -

- The last one-hour utilization behavior might determine future utilization behavior.
- The last 24 hour/ one day utilization might determine future utilization behavior.
- Weekly utilization behavior can determine future utilization, or
- Monthly utilization behavior might be the one which affect the future utilization behavior.

The second one is, certain nodes utilization might be influenced by another node utilization. This is because applications could be dependent to each other in nature so, one node which run a certain application utilization may depend on others or others could depend on this node utilization. The third thing is that this solution considered internal rules and policies. Private cloud is owned and used by a single organization and this organization has many internal rules and policies and these

rules and policies can possibly affect utilization. Figure 10 demonstrates the low-level design of dynamic threshold processor.

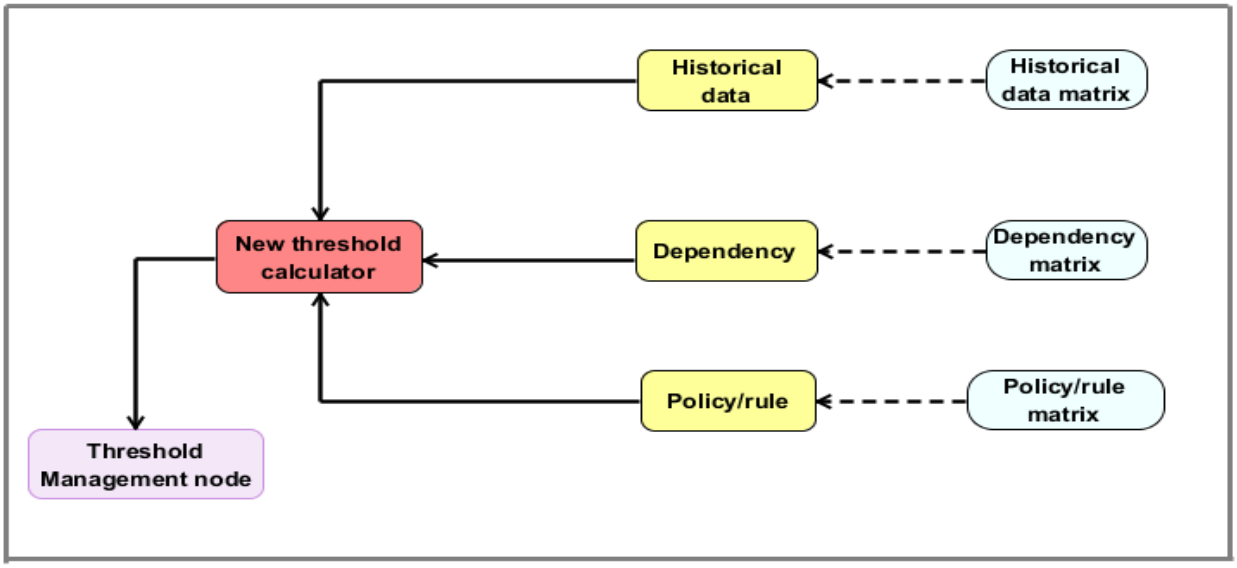


Figure 10: Dynamic threshold Processor LLD

These parameters (historical data, dependency and policy/rule) use their own matrixes: historical data matrix, dependency matrix and policy/rule matrix.

Historical data matrix

Node index is a unique index for each monitored node, as we saw earlier historical data could be the last one-hour utilization data, daily, weekly or monthly utilization data. Each monitored node could have calculated utilization data, for instance the Table 3 shows the node index and their corresponding data for each historical utilization data classification (recent/daily/weekly/monthly).

Table 3: Resource utilization data

Node Index	Recent(1hr)	Daily Avg	Weekly Avg	Monthly Avg
AX01	Increasing	67%	70%	72%
AX02	Decreeing	75%	62%	80%
----	**----**	**----**	**----**	**----**

If a certain node uses the last one-hour utilization data, the data might have increasing, decreasing or random behavior. If the data has increasing or decreasing behavior, the threshold would increase or decrees by a certain percent, else will take the average. For example, we will have Table 4 to calculate the new threshold based on historical utilization data.

Table 4: New threshold based on historical utilization data

Historical data	Average utilization	Utilization (threshold) will increase by
Recent (Increasing)	-	Last utilization value i.e. the highest
Recent (decreasing)	-	Last utilization value i.e. the lows
Recent (Random)	70<=Avg<=80	-7%
Daily	70<=Avg<=80	5%
Weekly	70<=Avg<=80	10%
Monthly	70<=Avg>=80	15%

Note: Threshold ranges and the numbers which specified in the matrixes are random numbers taken to show how the solution is work. These numbers are not dependent on the design or the algorithms; while implementing the solution in real environment, the numbers will replace with the company's real data.

Dependency matrix

This matrix defines two things regarding dependency-based threshold calculation, Table 5 shows which node is depend on which node/s and in what level of dependency. The level of dependence reference one node level of dependency on others, regarding this there are three dependency levels. The first dependency is the one which shows highly dependency between nodes (1st level dependency), the second one refers moderate dependency between nodes (2nd level dependency) and the last one is the one which refers low dependency (3rd level dependency).

Table 5: Dependency matrix

Node Index	1st level dependency	2nd level dependency	3rd level dependency
AX01	BC03-BC07	CZ11, CZ13-CZ16	CZ20
AX02	AX04-AX09	AX11, AX14, AX23	BC03, BC04
----	----	----	----

Table 6 defines how the threshold should look like in each dependency level. Node threshold will increase by 10% while one or more 1st level dependency node utilization is increased, and it will be 7% and 5% for 2nd and 3rd level dependency. First, the system checks 1st level dependency, and if nodes which list under 1st level dependency for a given node has a significant change then the threshold will be calculated accordingly and will not check 2nd and 3rd level dependency. If 1st level dependency has no significant change then the system will check 2nd level dependency and 3rd. For a given node, all three dependency levels will not apply as the same time to calculate the threshold.

Table 6: Dependency type

Dependency type	Utilization (threshold) increase by
1ST level dependency	10%
2nd level dependency	7%
3rd level dependency	5%

Policy/ Rules matrix

Regarding policy/rules it will depend on the company's interest, but here this paper considers two rules regard to resource utilization. If a new calculated resource utilization is increased or decreased by 20% of current resource utilization, then the node declared critical or normal regarding resource utilization. Based on the declaration (critical/normal) system admins might take different action accordingly. The node might not be on critical stage when we see it but increased resource utilization by 20% means critical for the company Table 7 shows the policy/rule matrix.

Table 7: Policy/rule matrix

Resource utilization	Declaration
New RU >= CRU + (CRU*20%)	Critical
New RU <= CRU - (CRU*20%)	Normal

Threshold management

Threshold management is responsible for taking next action after dynamic threshold processor calculate the dynamic threshold (Figure 11 shows the low-level design of threshold management). The alarm system will take the dynamic threshold as input to notify admin and users beside this,

the dynamic threshold tells us the resource utilization of a node; so, what should we do about it? Threshold management will answer this. Based on the current threshold value of the node, threshold management will decide whether additional resource is required or not. Elasticity type matrix will help to decide the elasticity type and the type of elasticity needed for each node will send to elasticity management for further action.

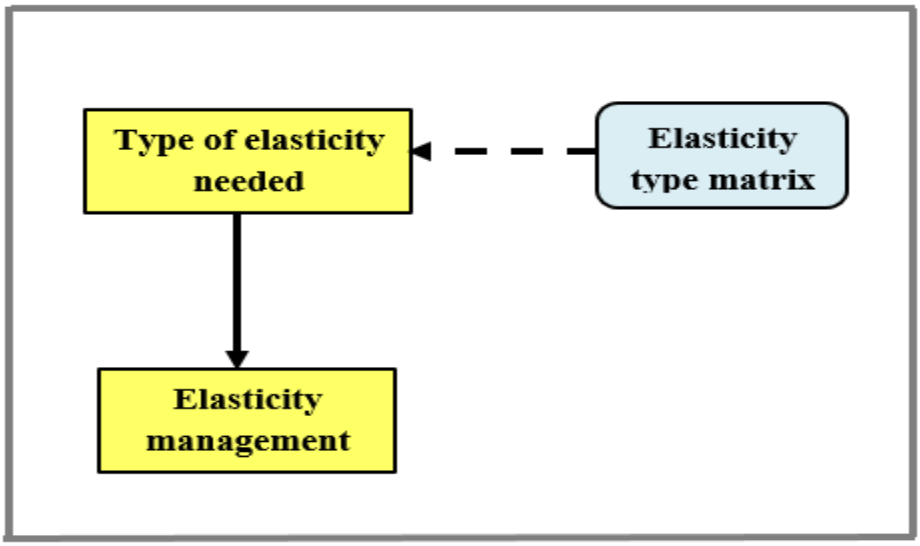


Figure 11: Threshold management LLD

elasticity matrix shows what type of elasticity is needed in what condition Table 8 is all about elasticity type, here for instance, if the threshold value is greater than 80% means resource utilization for that specific node is getting exhausted so resource upgrading is needed. In the other hand threshold value below or equal to 65 means downgrading should apply and between 65 and 80 is normal and no elasticity(upgrade/downgrade) is needed. In private cloud, resource provisioning is done by the end users and here in this solution the monitoring system itself can generate provision order as well.

Table 8: Elasticity type matrix

Threshold value	**Elasticity type**
Th>=80%	Upgrade
Th<=65%	Downgrade
65<Th<80	No elasticity

Elasticity management

Here after receiving elasticity type from threshold management next action is how much resource should be added or reduced from a given node, Figure 12 shows how elasticity management is designed. After the amount of resource needed is calculated provisioning order will send to provisioning system and notification will send to user and admins to make them know that provisioning order has been sent.

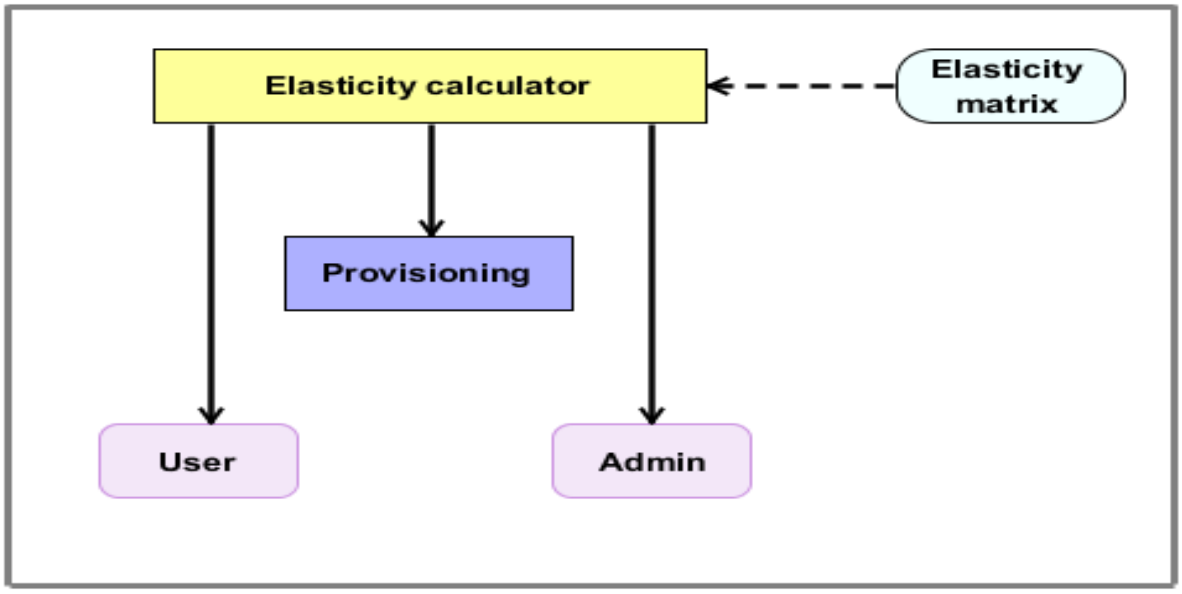


Figure 12: Elasticity management LLD

Elasticity management use elasticity matrix (Table 9) to calculate the resource needed to be added or reduced from a given node. The amount of resource needed to be added or reduced is depend on the available resource and the amount of added or reduced resources can change time to time as we needed.

Table 9: Elasticity matrix

Elasticity type	**New resource utilization will be**	**New provisioning will be done for**
Upgrade	Current resource + (Current resource *35%)	+ (Current resource *35%)
Downgrade	Current resource – (Current resource *15%)	- (Current resource *15%)

4.2.3. Data flow

Data flow in each function module/components will discuss here, starting with the data which the federation agent gets from collection agent till sending provisioning order to provisioning system and notifications to users and admins. This data flow shows clearly what this solution is trying to do. All three modules/components are discussed independently, and we will start from dynamic threshold processor which discuss on Figure 13, then threshold management and finally elasticity management.

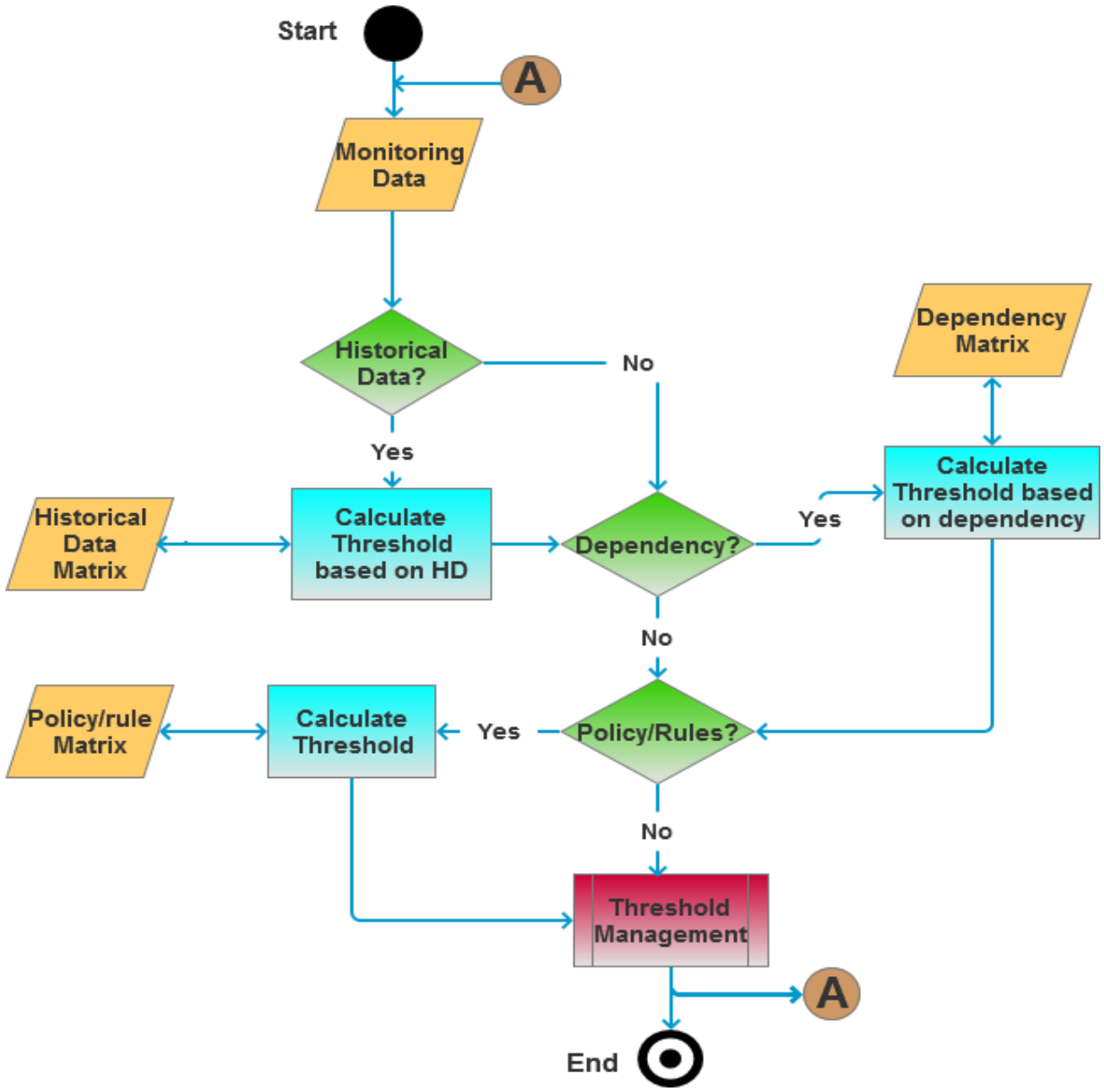


Figure 13: Dynamic threshold processor data flow

There are four alternatives to calculate the threshold based on historical utilization data: recent (1 hour), daily, weekly and monthly. To calculate the threshold based on recent resource utilization data of a given node first we should check the behavior of the data, it might be decreasing/ increasing or random. For decreasing/ increasing kind of resource utilization data, the last utilization value (the largest for increasing and the least for decreasing) will be current threshold value of that specific node else the average will take (for random) as current threshold value. But for daily, weekly and monthly based threshold calculation the average will take as current threshold value. Next the dependency of that specific node with other- monitored node will check (Table 5 shows the dependency between nodes). So previous threshold will hold the current threshold value and new current threshold value will calculated. For example, based on historical utilization data the threshold has been calculated and current threshold value is 7. So, current threshold = 7, and let us say a node which has 1st level dependency with this monitoring node has significant change in its utilization (increased by 20% from previous utilization) hence

previous threshold = current threshold (which is 7)

current threshold = current threshold + (current threshold*10%) = 7.7

If the node which has 1st level dependency with this monitored node has no significant change in its resource utilization, then 2nd level dependency nodes will check and if one of the 2nd level dependency node has significant change then

previous threshold = current threshold (which is 7)

current threshold = current threshold + (current threshold*7%) = 7.49.

for the case of 3rd level dependency the threshold will be

previous threshold = current threshold (which is 7)

current threshold = current threshold + (current threshold*5%) = 7.35.

then the policy/rule will check the utilization of the monitored node and if the utilization is increased by 20% then the node will be labeled as critical else its normal and no special monitoring is needed for that node.

The next data flow is for threshold management, as shown in Figure 14 the alarm system is not the focus of this solution.

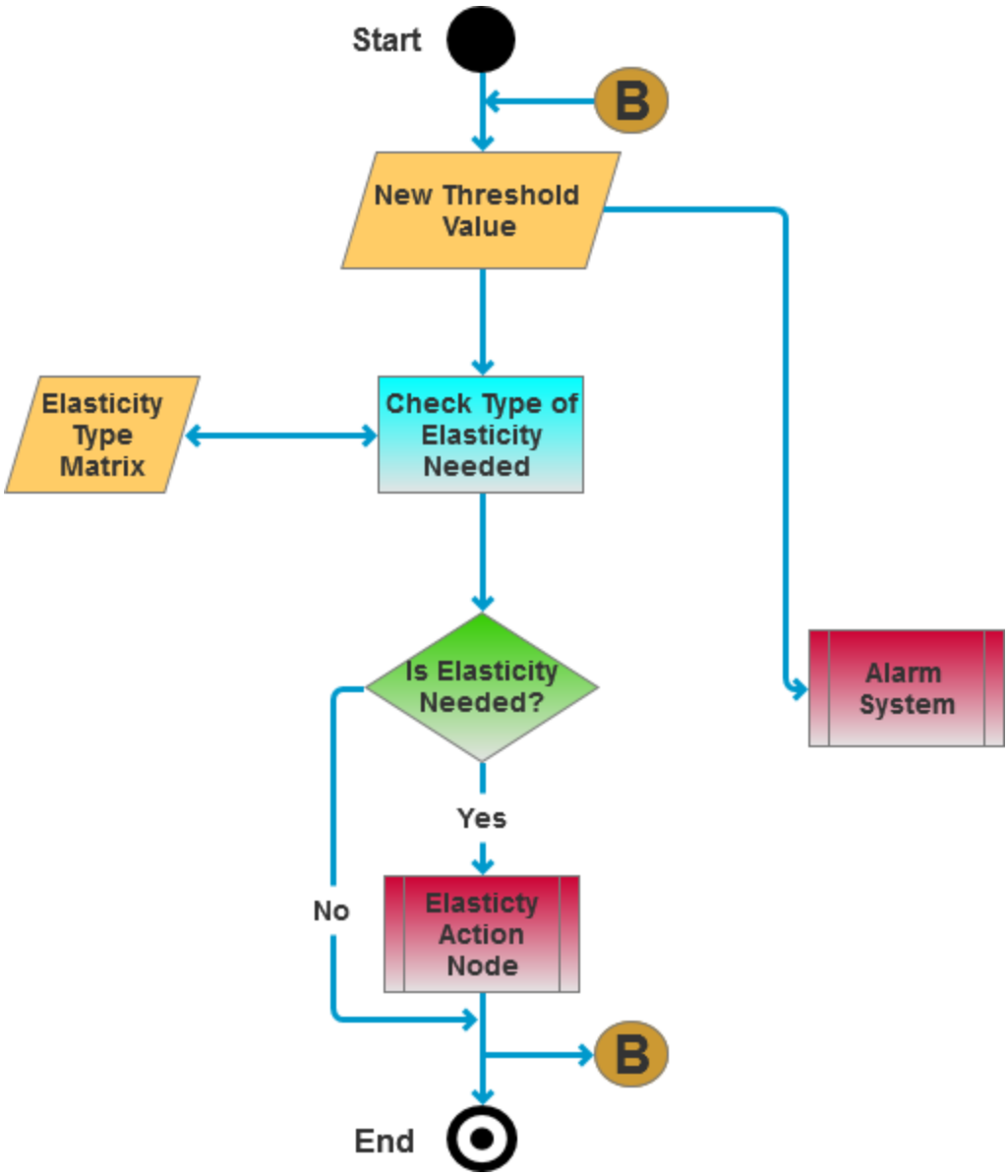


Figure 14: Threshold management data flow

Threshold management will receive the newly calculated threshold value for a given node and based on that it decides what kind of elasticity is needed. If downgrade or upgrading is required, the node index and type of elasticity are sent to elasticity management module.

The last one is elasticity management data flow (Figure 15), like threshold management here the provisioning modules are the one which is not covered in this work.

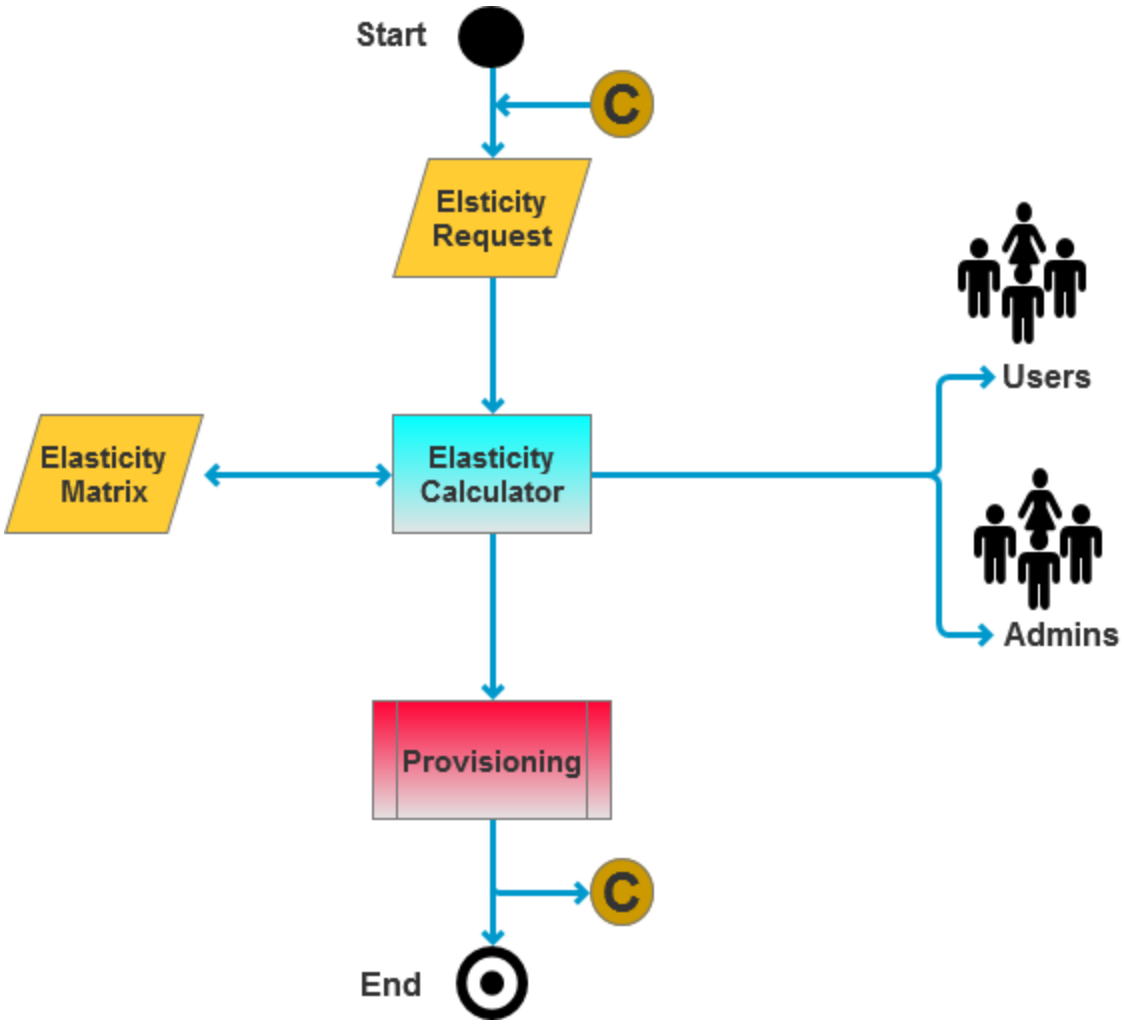


Figure 15: Elasticity management data flow

Based on the elasticity matrix which discussed earlier the amount of resources added/reduced will calculate and it will send to provisioning system for request execution. At the same time users and admins will notified that elasticity is needed and order has sent to provisioning system.

4.2.4. Pseudocode Algorithm

Algorithm related things will discuss in the upcoming Chapter, however here the pseudocode will discuss for all three newly added components/ modules.

Pseudocode for dynamic threshold management

1. Start
2. Previous utilization = current utilization
3. Current utilization = new monitored data from collection agent
4. If historical data is applied
    If the last one-hour utilization is increasing or decreasing
        Previous threshold = current threshold
        Current threshold = the last utilization value
    End if
    Else if last one-hour utilization is random
        Sum = sum of utilization data
        Previous threshold = current threshold
        Current threshold = Sum/12 (12 is number of utilization data collected for the last 1 hour).
    End if
    Else if daily, weekly or monthly historical data is applied
        Sum = sum of utilization data
        Previous threshold = current threshold
        Current threshold = Sum/288 or Sum/2016 or Sum/8064
    End if
  End if
5. If dependency is applied
    Else if the dependency is 1st level dependency
        Previous threshold = current threshold
        Current threshold = Current threshold + (Current threshold*10%)
    End if
    Else if the dependency is 2nd level dependency
        Previous threshold = current threshold
        Current threshold = Current threshold + (Current threshold*7%)
    End if
    Else if the dependency is 3rd level dependency

Previous threshold = current threshold

Current threshold = Current threshold + (Current threshold*5%)

End if

End if

6. If policy/rule is applied

   If Current utilization = Previous utilization + (Previous utilization*20%)

   Then the node labeled as “Critical”

   End if

   Else its normal

   End if

7. End

Pseudocode for threshold management

1. Start
2. Get current threshold value
3. Sent to the alarm system
4. If current threshold > = 80

   Elasticity-type = upgrade

   End if

   Else if 65<current threshold<80

   Elasticity-type = normal

   End if

   Else if current threshold < = 65

   Elasticity-type = downgrade

   End if

5. End

Pseudocode for elasticity management

1. Start

2. Get elasticity-type
3. If elasticity-type = upgrade

   Current utilization = Current utilization + (Current utilization *35%)

   End if

   If elasticity-type = downgrade

   Current utilization = Current utilization - (Current resource *15%)

   End if

4. Generate and send provisioning order to provisioning system
5. End

4.3. Summary

This Chapter showed the design of the new proposed framework, it exclusively shows the newly added components design. This research work believes that all kind of monitoring system can incorporate these components to take the advantage of the dynamic threshold mechanism for private cloud infrastructure monitoring.

Chapter 5: Evaluation

5.1. Overview

This Chapter discuss the evaluation step of the solution in detail; tools and technologies used in evaluation and the algorithm and test cases which can show the main functionalities of the solution are also discussed. In addition, the evaluation setup and considerations take during the evaluation are also discussed and finally the result and discussion part is presented.

5.2. Tools and Technologies

To evaluate the proposed solution, this research does not use any specific tool since there is no defined tool to evaluate such solution (monitoring). Java, the very known programing language is used to code, almost all related researches used this programing language to write their codes [9] [12] [14] [24]. Most researchers use actual cloud environment or lab to evaluate their works, however since we do not have such environment around, it is not possible evaluate the proposed solution in real cloud environment or cloud lab. Basically, what is needed for evaluation is monitored data from monitored node to do the dynamic threshold computation, and to do that there is no accessible cloud environment or such lab either. Though, there are number of papers which focus their work on tracing workloads in actual cloud environment. This work load trace data can be real cloud environment input for this evaluation. This research chooses bitbrain workload trace data to do the evaluation, next we will see some key concepts regarding what has been done on the workload trace and what is bitbrain.

Bit brain

Bitbrains is a cloud service provider that specializes in managed hosting and business computation for enterprises. In general, Bitbrains hosts three types of VMs: management servers, application servers, and compute nodes. Management servers are used for the daily operation of customer environments (e.g., firewalls). Examples of application servers are database servers, web servers, and head-nodes (for compute clusters). Compute nodes are mainly used to do simulation and other compute-intensive computation, such as Monte Carlo-based financial risk assessment. Bitbrain customers include many major banks (ING), credit card operators (ICS), insurers (Aegon) [17].

Bitbrains IT Services develop, implement, host and manage IT environments within the strictest boundaries of security. Bitbrains specializes in providing Solvency II compliance services for insurers, managed hosting, vCloud services, virtual private datacenter services and Infrastructure-as-a-Service [17]. Here is how they did the trace, from the distributed datacenter of Bitbrains, they collect two traces of the execution of business-critical workloads. For this they use the monitoring and management tools provided by VMware, such as vCloud suite. For each trace, the vCloud Operation tools record 7 performance metrics per VM, sampled every 5 minutes: the number of cores provisioned, the provisioned CPU frequency, the CPU usage (average usage of CPU over the sampling interval), the provisioned memory capacity, the actual memory usage (the amount of memory that is actively used), the disk I/O throughput, and the network I/O throughput. Thus, they obtain traces that cover both requested and actually-used resources, for four resource types (CPU, memory, disk, and network). They collect the data between August and September 2013, the traces accumulate data for 1,750 virtual machines, with over 5,000 cores and 20 TB of memory, and operationally accumulate over 5 million CPU hours. This trace data is available at bitbrains site [33] , and this research used this data as input utilization data to evaluate the proposed solution.

This research chooses this Bitbrain data with two reasons, the first one is the work load data collected from real cloud business-critical loads which is able to show that the proposed system can work for companies which run business critical system like ethio telecom. The second and the main one is the data is collected for a month long. This is actually very important because the proposed solution need monthly data of a monitored node for those which use monthly historical data to calculate the dynamic threshold. Hence, these two things are the reason why this research is going after bitbrains data and it helps to evaluate the solution with real cloud environment data.

5.3. Evaluation setup

The solution which is proposed in this thesis work is not a full-fledged cloud monitoring system. But the newly added components will help the already used monitoring system to enhance resource utilization and QoS in private cloud environment. As an example, upgraded version of open SUSE monitoring system is presented in Chapter 4 (Figure 9) which equipped with the newly introduced components. The same is true for other monitoring systems as well, we can easily integrate these new components with the existing ones. In this section two main points will discuss regarding the evaluation setup:

A. Why not real experiment?

B. Why not simulation tools & if not, what alternatives do we have?

These two questions will show the steps which this thesis work going through to evaluate the proposed solution; the steps will discuss in detail.

A. Why not real experiment?

The first choice this paper wants to use as validation step was to simulate the solution in real cloud environment or in cloud lab. However, since real cloud or such lab environment are not exist both alternatives shouldn't work as a validation step. As a second choice we may think, why not deploying cloud environment itself and run the solution on it? The solution needs to run on top of cloud computing environment and to do that beside monitoring knowledge, this approach needs comprehensive knowledge and professionals in cloud computing. Since the study focus was the monitoring part, and with the assumption of simulation tools; we could not able to acquire a knowledge which help to develop cloud tool or environment which can able to integrate with monitoring systems. There are cloud operating systems such as OpenStack, OpenNebula and eucalyptus these operating systems helps to create the cloud environment, and some are open sources too. The operating systems has their own minimum requirements to operate on and needs knowledge to integrate the cloud environment with the monitoring system. This research did not find any related document which might help with the integration process. As discussed in the limitation part in Chapter 1, this work lacks to validate using real cloud environment.

B. Why not simulation tools & if not, what alternatives do we have?

It was tried to use simulation tools as a second alternatives to validate this work; however, there is no specific cloud monitoring simulation tool. There are a few cloud simulation tools and in this research three of them are seen.

CloudSim [34]

The primary objective of this project is to provide a generalized, and extensible simulation framework that enables seamless modeling, simulation, and experimentation of emerging Cloud computing infrastructures and application services. By using CloudSim, researchers and industry-based developers can focus on specific system design issues that they want to investigate, without getting concerned about the low-level details related to Cloud-based infrastructures and services.

GreenCloud [35]

Greencloud is a sophisticated packet-level simulator for energy-aware cloud computing data centers with a focus on cloud communications. It offers a detailed fine-grained modeling of the energy consumed by the data center IT equipment, such as computing servers, network switches, and communication links.

iCanCloud [36]

iCanCloud is a simulation platform aimed to model and simulate cloud computing systems, which is targeted to those users who deal closely with cloud-based systems. The main objective of iCanCloud is to predict the trade-offs between cost and performance of a given set of applications executed in a specific hardware, and then provide to users' useful information about such costs.

From the three simulation tools this research work chose the first one (cloudsim) to study further for three reasons. The first one is the simulation tool objective i.e. experimentation on infrastructures and application services. The second one is the language used to implement the tool is java, while the other two uses C++ and the last reason is cloudsim is the widely used simulation toolkit. The main thing which require from this simulation tools are cloud environment which have nodes/resource and their runtime utilization data. The main thing is the utilization data, that is what this research solution needs to calculate the threshold and resource elasticity. This research work assumes that the simulation tool (cloudsim) can able to provide runtime utilization data of cloud resources/nodes. However, the tool can able to create the cloud environment but did not give utilization data of the resources. Utilization data also known as workload data has provided independently in cloudsim which are collected from cloud laboratory called planetlab. These workload data are collected from cloud laboratory from different VMs every 5 minutes for 24 hours. Thus, this paper realized that cloudsim is not good enough to simulate the proposed solution with two reasons. The first reason is the cloud simulation tools are not designed to give utilization data from the actual simulated cloud environment, so using the tool has no relevance in this regard. The second reason is that, the workload data (utilization data) provided by the tool is not enough to validate the solution because the solution needs monthly, weekly, daily and hourly utilization data. Because of the reasons discussed above cloud simulation tools have limitation with the extent what this research work is needed.

The next thing is if the simulation tools are not helping, what alternative way we have for validation? bitbrain is the answer this research gets at this point. Bitbrain has discussed in detail earlier in this Chapter, in Table 10, what kind of research works use bitbrain data and how the validation of the solution goes through using this data is discussed. Many studies use bitbrain data in their works but five of them are summarized and discuss below.

Table 10: Bitbrain based papers

Title	Problem	Approach	Evaluation	Finding
Heuristics and metaheuristics for dynamic management of computing and cooling energy in cloud data centers [29].	Optimizing the energy consumption of cloud infrastructures is a major challenge to place data centers on a more scalable scenario.	Novel power and thermal-aware strategies and model.	- Bitbrain Workload - Power & thermal models	Combined awareness from both metaheuristic and best fit decreasing algorithms allow to describe the global energy into faster and lighter optimization strategies that may be used during runtime.
An Efficient and Fair Multi-Resource Allocation	Existing multi-resource fair allocation mechanisms, notably Dominant	Server based approach	Extensive trace-driven (Bitbrain) simulations	Its amenable to distributed implementation, and it enhances resource

Mechanism for Heterogeneous Servers [30].	Resource Fairness and its follow-up work.			utilization compared to the existing mechanisms.
Energy-aware and multi-resource overload probability constraint-based virtual machine dynamic consolidation method [37].	An effective and efficient virtual machine consolidation method is required for guaranteeing service quality, reducing energy consumption and maximizing resource utilization.	energy-aware dynamic virtual machine consolidation method.	Extensive simulation is conducted to compare the proposed method with previous virtual machine consolidation methods using Bitbrain workload trace data.	The proposed model is highly effective in reducing VM migrations and energy consumption of data centers and in improving QoS.
Enhancing Energy-Efficient and QoS Dynamic Virtual Machine Consolidation Method in Cloud Environment [38].	Aggressive VM consolidation approaches lead to physical host over-utilization which affects energy efficiency, resource utilization and Qos.	New dynamic virtual machine consolidation method.	Experiment under different workload traces (Bitbrain) from a real system.	The new proposed approach can significantly outperform other traditional methods regarding energy consumption, QoS guarantees, and the number

				of VM migrations.
Which is the best algorithm for virtual machine placement optimization? [39]	finding the optimal allocation of virtual machines (VMs) on the physical machines available in the provider's data center.	Objective comparison of different VM allocation algorithms	Using CloudSim and Bitbrain workload trace data.	There is no clear winner between the algorithms, but generally “Guazzone" and “Shi-AC" gave the best results.

5.4. Implementation with java code

The evaluation has two set of java code implementations, the first code is based on random monitored data input and the second one is bitbrains workload trace data.

Java code with random input

As we can understand from the name itself, the code is based on a random data input. Monitored data is random in nature, there is no constant data since monitored node resource utilization varies from time to time. This implementation wants to show the solution while operating on random inputs; The code initially generates random data for recent (1 hour), daily, weekly and monthly utilization data, the utilization data is generated as if the data is collected every 5 minutes. For recent utilization 12 random data is generated for one hour, for daily utilization 288, for weekly 2016 and for monthly 8064 random data have generated. In addition, dependency matrix is produced, Table 11 show the dependency and each row shows the dependency matrix for a given node (A, B, C & D).

Table 11: Dependency matrix, 0=no dependency, 1=1st dependency type, 2=2nd dependency type 3=3rd dependency type, NA=not applicable.

	a	b	c	d
A	**NA**	**1**	**2**	**1**
B	**1**	**NA**	**1**	**0**
C	**1**	**0**	**NA**	**3**
D	**2**	**1**	**3**	**NA**

Since the threshold is based on the utilization data, the current utilization data should have copied to previous data before current utilization is changed. So, initialization is done with the above three steps; random data generation, dependency matrix and coping current to previous utilization then the actual computation will start as per the design and pseudo code which is defined in Chapter 4.

Java code with bitbrain workload trace data

Bitbrain workload trace data is more valuable data input with that of auto random generated data, because the data is collected from real cloud environment from business-critical workloads and the data is collected exactly every 5 minutes. For the implementation, from 1750 VMs one-month data 16 VMs data is chosen randomly (4 VMs data for each nodes A, B, C & D) and each data VMs data has 8614 to 8637 records however, there is no explanation for this record variations. since bitbrain data is a one-month data, first initialization should have done for recent, daily and weekly data for each monitored node (A, B, C & D). The last 12 records are for recent data, the last 288 for daily and the last 1026 for weekly data. Dependency matrix is the same as used in 'Java code with random input' implementation part.

5.5. Test cases

There are five test cases to verify the proposed solution, and all the test cases expected to be successful to say the solution is working as planned. The test cases result discussed on test case result section.

Test 01: Threshold calculation with historical utilization data.

Objective	
To verify that the solution can do by considering only historical utilization data.	
Prerequisites 1. The java code is compiled and has no error.	
Procedure	**Expected result**
1. Define four possible history-based utilization data (recent, daily, weekly and monthly) for each node (A, B, C & D).	1. Calculating threshold using history data is successful.
Remarks None	

Test 02: Threshold calculation with dependency matrix.

Objective	
To verify that the solution can do by considering only dependency to calculate the dynamic threshold.	
Prerequisites 1. The java code is compiled and has no error.	
Procedure	**Expected result**
1. The system calculates the threshold based on the dependency matrix.	1. Calculating threshold using dependency is successful
Remarks None	

Test 03: Threshold calculation with historical data, dependency and policy/rules all together.

Objective
To verify that the solution can do by considering historical data, dependency and policy/rules all together to calculate the dynamic threshold.

Prerequisites	
1. The java code is compiled & has no error. 2. The above two test cases are successful.	
Procedure	**Expected result**
1. To calculate the threshold, assign historical data, dependency and policy/rules to each node. 2. The system calculates the threshold based on the historical data, dependency and policy/rules.	1. Assigning historical data, dependency and policy/rules to each node is successful. 2. Calculating threshold using history, dependency and policy/rules is successful.
Remarks None	

Test 04: Threshold management.

Objective	
To verify that the threshold management is working.	
Prerequisites 1. The java code is compiled and has no error. 2. The above test cases are successful.	
Procedure	**Expected result**
1. The threshold management calculate the type of elasticity needed as per the calculated threshold.	1. Calculating the type of elasticity needed for each node is successful.
Remarks None	

Test 05: Elasticity management.

Objective
To verify that the elasticity management is working.
Prerequisites

1. The java code is compiled and has no error. 2. The above test cases are successful.	
Procedure	**Expected result**
1. The elasticity management calculates the elasticity and send notification to user and admin.	1. The elasticity is calculated, and notifications send successfully.
Remarks None	

5.6. Evaluation Considerations

The setup starts with configuring NetBeans, NetBeans is an open-source project dedicated to providing rock solid software development products (the NetBeans IDE and the NetBeans Platform) that address the needs of developers, users and the businesses who rely on NetBeans as a basis for their products; particularly, to enable them to develop these products quickly, efficiently and easily by leveraging the strengths of the Java platform and other relevant industry standards [16]. After configuring NetBeans, the widely-used IDE for java, then changing the design and the algorithms into java code was the next step. Preparing bitbrain data was done since the actual file has many fields which are separated by comma. Hence, converting the data to tabular form and extracting only CPU utilization (in %) is done. Then java with random data input and with bitbrain data is implemented accordingly.

There are a few assumptions that this research takes into consideration while evaluating the proposed framework solution:

- The evaluation is done with the assumption of four monitored nodes (however the number of monitored nodes does not affect at all).
- Only CPU utilization is take as monitored data.
- The input monitored data is taken from the nodes every 5 minutes.

5.7. Test Cases Result

Since the implementation has two version with random input and with bitbrain workload trace data; the result discussion will include both implementations. Test case 04 & 05 will discuss with other test cases because test case 04 and 05 are dependent on the calculated threshold and it's good to see all together for better understanding.

The policy which considered to do the evaluation gives us two outputs, critical and normal. Critical or normal does not refer the threshold, it gives us critical if the resource utilization of a given node is greater than the previous utilization by 20% else it will give us normal. Sometimes new calculated threshold has no change, or no elasticity is needed but the policy may level the node as critical. Critical implies that specific node utilization has significant change from time to time, so close monitoring is needed. For example, a node resource utilization was 5 then it became 6 (increased by 20%) with this data the policy labeled this node as critical but, elasticity might not apply for this node.

To clearly show the change of the threshold based on the utilization, each test case has four rounds. Current utilization and current threshold values will be previous utilization and previous threshold for the next round. This will not be always true for test case 02 and 03 and the reason will discuss later.

Test case 01 result

This test case is considering only historical data to calculate the dynamic threshold, and the test results are discussed below. The test has two cases one for random inputs and the other is for bitbrain data.

Case 1: With random input

Please refer Appendix A for actual java output.

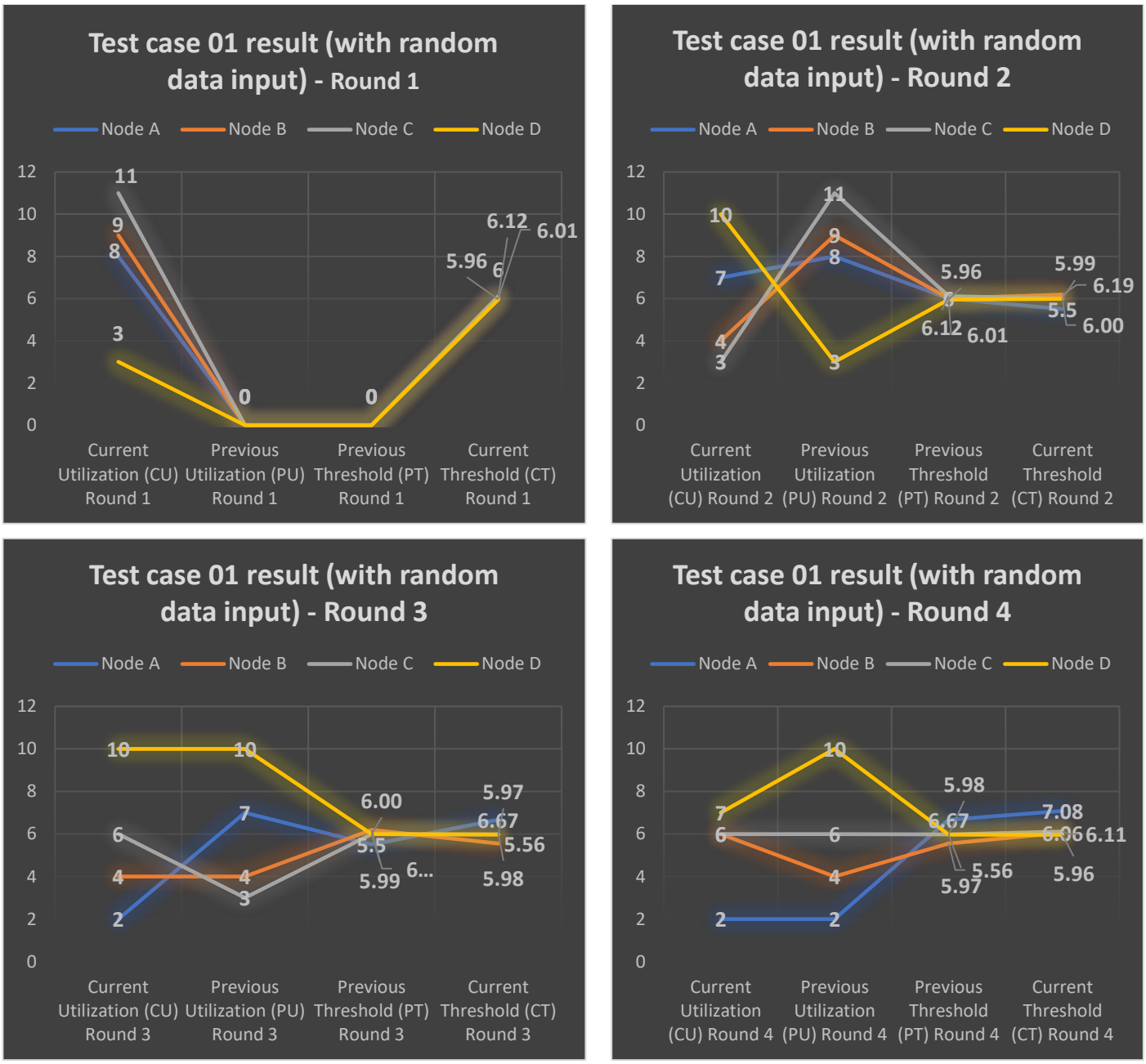


Figure 16: Test case one result with random data input

Threshold management & elasticity management

Table 12: Threshold management, elasticity management and policy/rule for history-based utilization data (Random input)

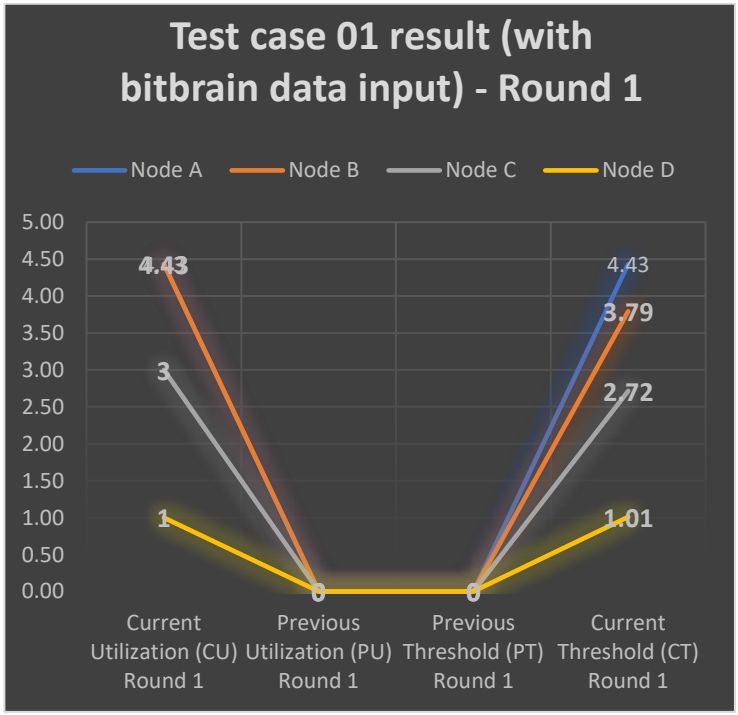
	Node A	Node B	Node C	Node D
Threshold Management	Upgrade	No elasticity	Upgrade	No elasticity
Elasticity Management	+350mips	No elasticity	+350mips	No elasticity

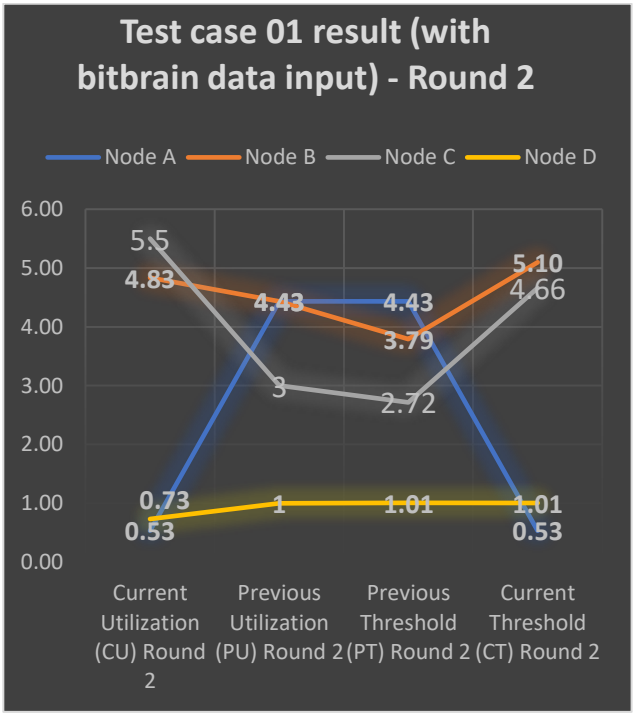
Policy/rule	Normal	Critical	Normal	Normal

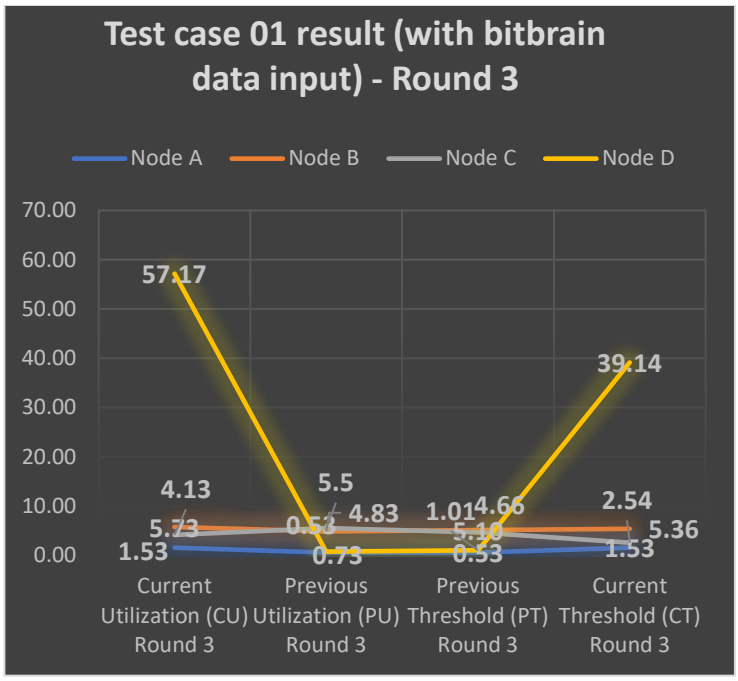
This test is done with random input based on historical data to calculate the threshold. Figure 16 shows the calculated dynamic threshold values based on utilization which is changed time to time. And based on the threshold, threshold management and elasticity management decide what kind of elasticity is needed and how much resource should add or deduct from each node. Table 12 shows us what threshold management; elasticity management and the policy/rule say about each node based on the utilization.

Case 2: With bitbrain input

Please refer Appendix B for actual java output.







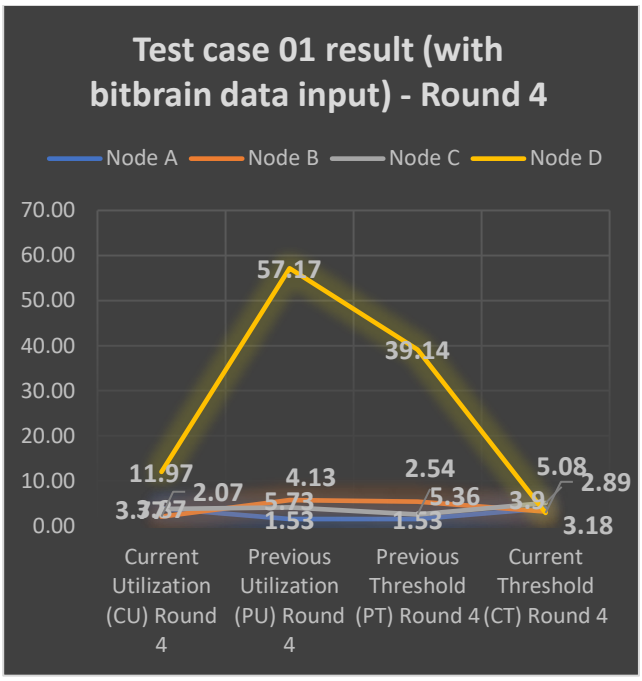


Figure 17: Test case one result with bitbrain data input

Threshold management & elasticity management

Table 13: Threshold management, elasticity management and policy/rule for history-based utilization data (bitbrain input)

	Node A	**Node B**	**Node C**	**Node D**
Threshold management	Downgrade	No elasticity	No elasticity	Upgrade
Elasticity management	-150mips	No elasticity	No elasticity	+350mips
Policy/Rule	Critical	Normal	Normal	Normal

This test is done with bitbrain input based on historical data to calculate the threshold. Figure 17 shows the calculated dynamic threshold values based on utilization which is changed time to time. And based on the threshold, threshold management and elasticity management decide what kind of elasticity is needed and how much resource should add or deduct from each node. Table 13 shows us what threshold management; elasticity management and the policy/rule say about each node based on the utilization.

Test case 02 result

This test is considering only dependency to calculate the threshold, like the previous test, this one also done for both random and bitbrain data inputs; the result is discussed below.

Case 1: With random input

Please refer Appendix C for actual java output.

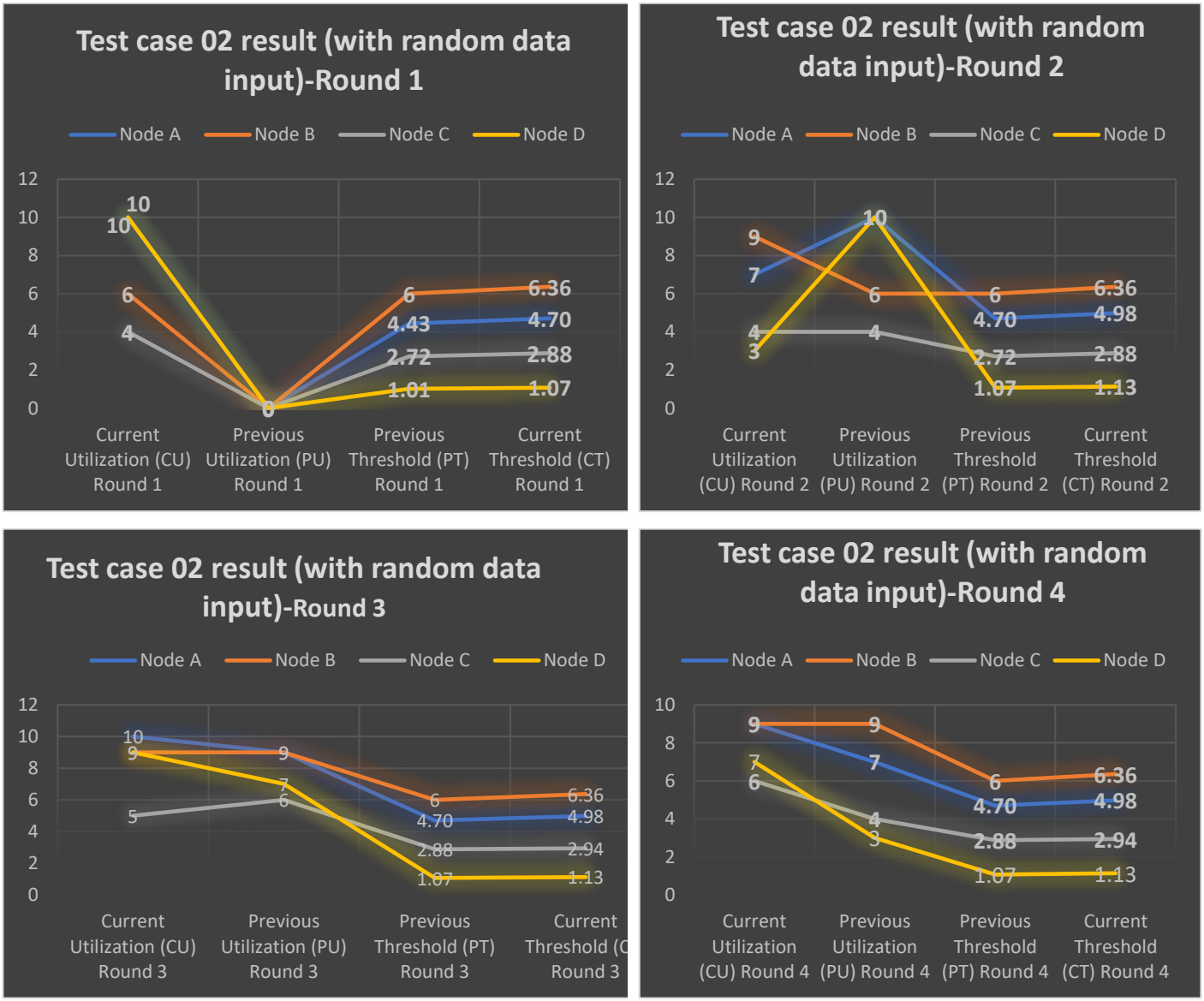


Figure 18: Test case two result with random data input

Threshold management & elasticity management

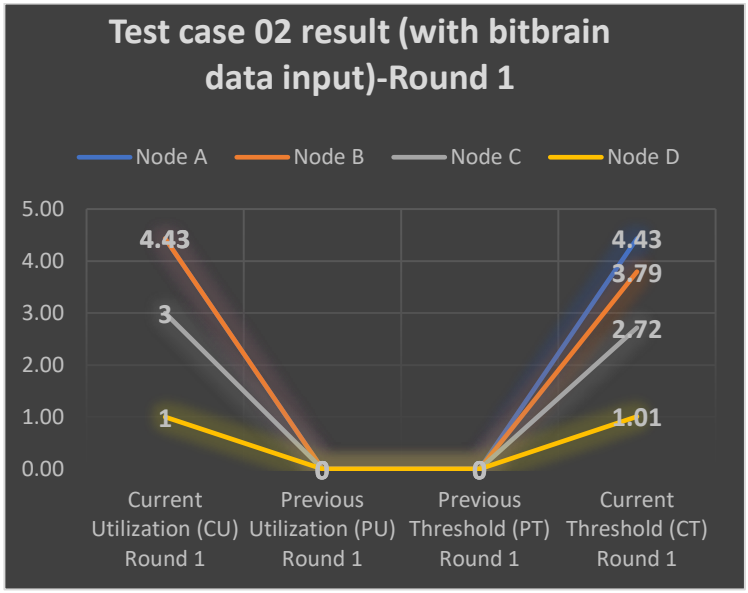
Table 14: Threshold management, elasticity management and policy/rule for dependency-based utilization data (random input)

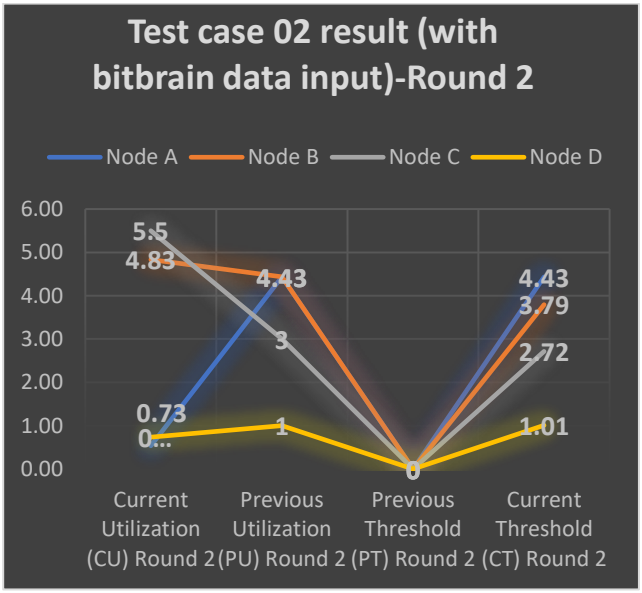
	Node A	Node B	Node C	Node D
Threshold management	No elasticity	Upgrade	Downgrade	Downgrade
Elasticity management	No elasticity	+350mips	-150mips	-150mips
Policy/Rule	Normal	Normal	Normal	Critical

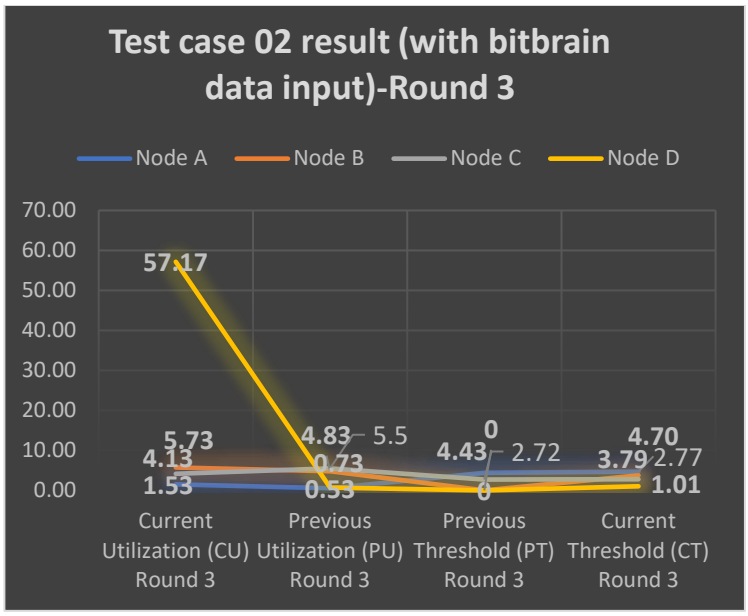
This test is done with random input based on dependency to calculate the threshold, the dependency matrix is shown on Table 11. Figure 18 shows the calculated dynamic threshold values based on utilization which is changed time to time. Based on the threshold, threshold management and elasticity management decide what kind of elasticity is needed and how much resource should add or deduct from each node. Table 14 shows us what threshold management; elasticity management and the policy/rule say about each node based on the utilization.

Case 2: With bitbrain input

Please refer Appendix D for actual java output.







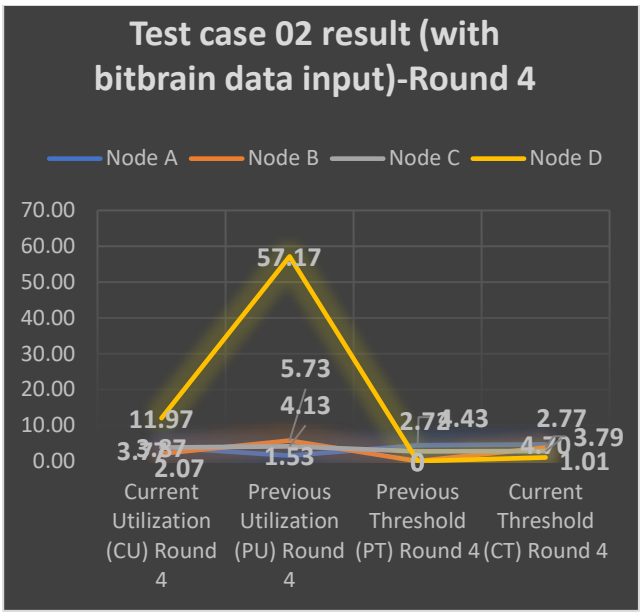


Figure 19: Test case two result with bitbrain data input

Threshold management & elasticity management

Table 15: Threshold management, elasticity management and policy/rule for dependency-based utilization data (bitbrain input)

	Node A	Node B	Node C	Node D
Threshold management	No elasticity	Downgrade	Downgrade	Downgrade
Elasticity management	No elasticity	-150mips	-150mips	-150mips
Policy/Rule	Critical	Normal	Normal	Normal

This test is done with bitbrain input based on dependency to calculate the threshold, the dependency matrix is shown on Table 11. Figure 19 shows the calculated dynamic threshold values based on utilization which is changed time to time. In addition, based on the threshold, threshold management and elasticity management decide what kind of elasticity is needed and how much resource should add or deduct from each node. Table 15 shows us what threshold management; elasticity management and the policy/rule say about each node based on the utilization.

Dependency based threshold calculation is different with that off history-based calculation, the result show that too. Previous threshold of a given round is not always holding the previous round. The threshold is calculate based on the dependency matrix (Table 11), if the node which has influence on a given node has no significant change then the new threshold will not be changed.

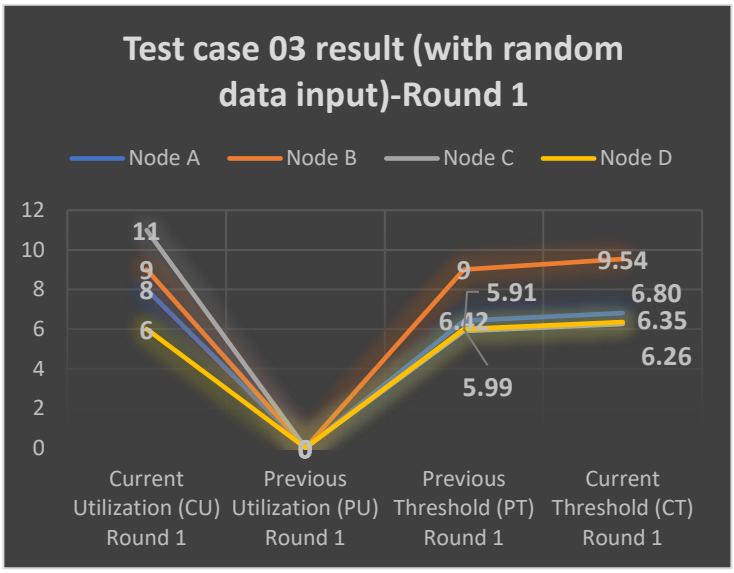
If a given node current threshold is increased by 5% from previous threshold then the node which is depend on this node will affect and the threshold will calculate based on dependency type (Table 6). For instance, node D has 2nd level dependency with node A, 1st level dependency with node B and 3rd level dependency with node C; hence the threshold of node D is changed if one of other nodes (A, B & C) utilizations is go beyond a certain level. If these nodes utilization did not have significant change, there is no change on the monitored node threshold.

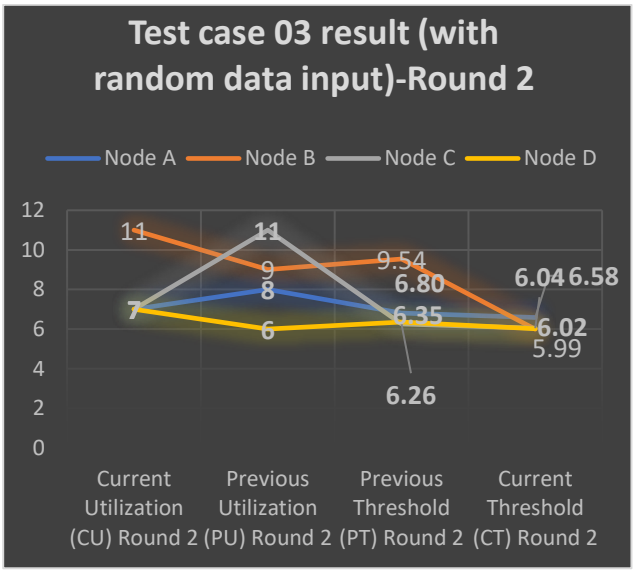
Test case 03 result

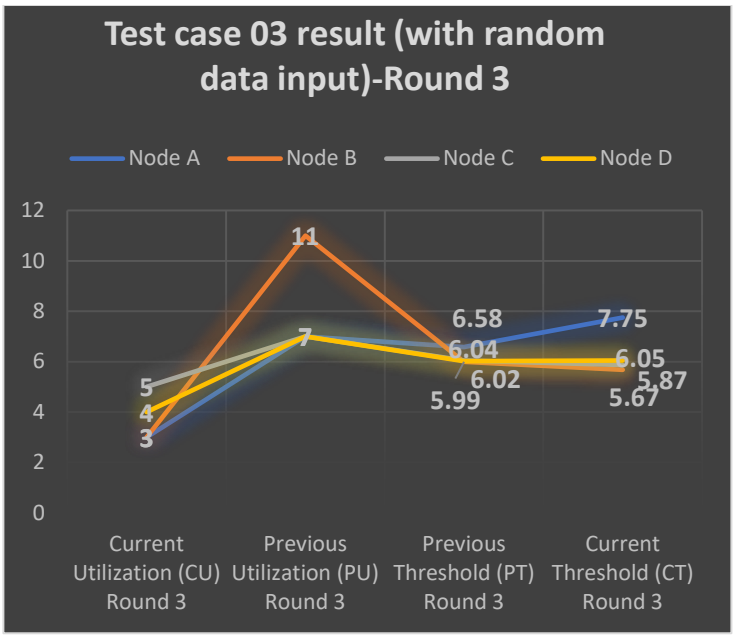
This test case considers all parameters (history data, dependency and policy/rule) to calculate the threshold, and the results for both random and bitbrain data are discussed below.

Case 1: With random input

Please refer Appendix E for actual java output.







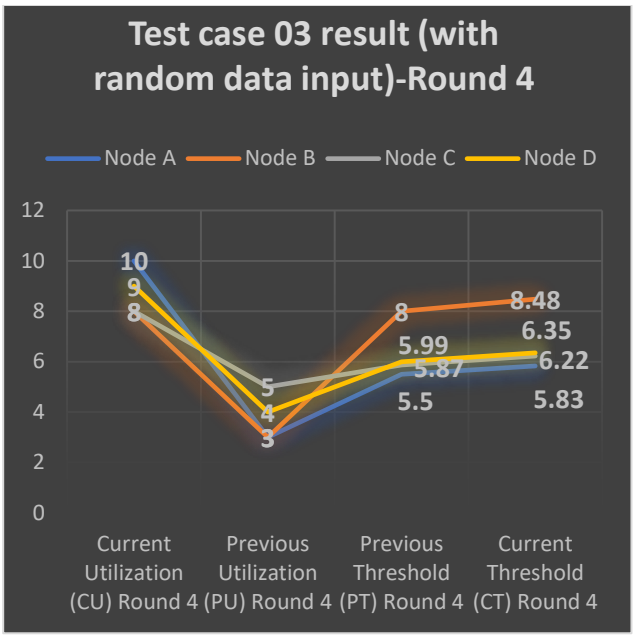


Figure 20: Test case three result with random data input

Threshold management & elasticity management

Table 16: Threshold management, elasticity management and policy/rule using all parameters (random input)

	Node A	Node B	Node C	Node D
Threshold management	No elasticity	Upgrade	Upgrade	Upgrade
Elasticity management	No elasticity	+350mips	+350mips	+350mips
Policy/Rule	Critical	Critical	Critical	Critical

This test is done with random input based on history based and dependency to calculate the threshold. Figure 20 shows the calculated dynamic threshold values based on utilization which is changed time to time. Based on the threshold, threshold management and elasticity management decide what kind of elasticity is needed and how much resource should add or deduct from each node. Table 16 shows us what threshold management; elasticity management and the policy/rule say about each node based on the utilization.

Case 2: With bitbrain input

Please refer Appendix F for actual java output.

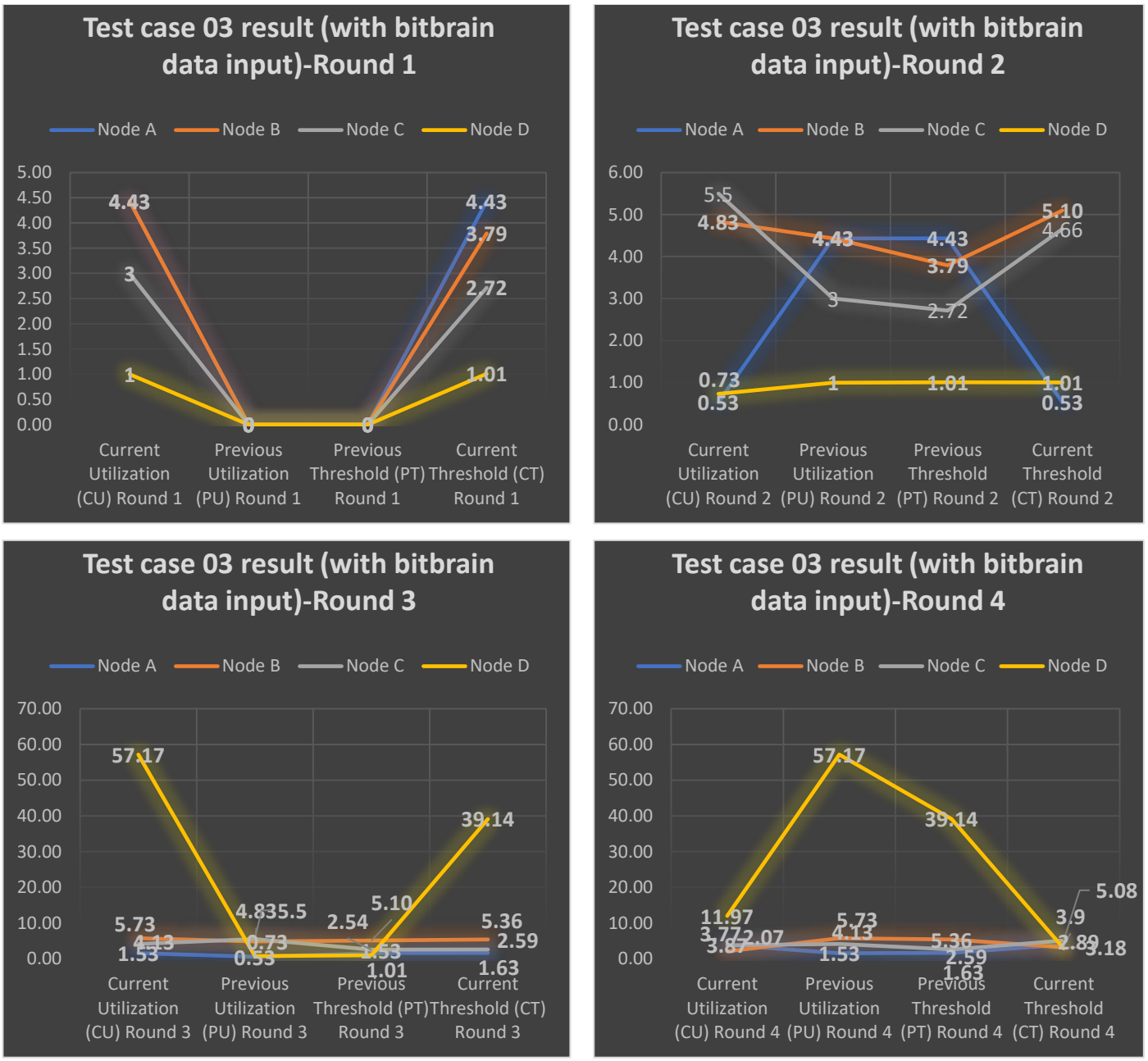


Figure 21: Test case three result with bitbrain data input

Threshold management & elasticity management

Table 17: Threshold management, elasticity management and policy/rule using all parameters (bitbrain input)

	Node A	**Node B**	**Node C**	**Node D**
Threshold management	Downgrade	No elasticity	No elasticity	Upgrade
Elasticity management	-150mips	No elasticity	No elasticity	+350mips
Policy/Rule	Critical	Normal	Normal	Normal

This test is done with bitbrain input based on history based and dependency to calculate the threshold. Figure 21 shows the calculated dynamic threshold values based on utilization which is changed time to time. Based on the threshold, threshold management and elasticity management decide what kind of elasticity is needed and how much resource should add or deduct from each node. Table 17 shows us what threshold management; elasticity management and the policy/rule say about each node based on the utilization.

These three test cases with random and bitbrain data inputs shows the whole figure of the proposed solution, according to the tests the solution is doing what it intends to do successfully.

The test results show the java implementation and its results, so what we should understand from the result is important. The data we got as a result shows the output of the proposed system. The threshold value is changed in each monitoring time based on each node resource utilization (in the result each monitoring time presented as monitoring round i.e. round 1 to 4). This demonstrated that the dynamic threshold is achieved in the proposed solution. The dynamic threshold is used as input for threshold management and elasticity management. After four consecutive monitoring time (four rounds) threshold management decided what type of elasticity is needed i.e. upgrade, downgrade or no elasticity. Then, elasticity management calculate the amount of resources added or removed from the nodes and automatically provisioning order is send to provisioning system. Here, it is good to understand that the frequency of elasticity processing can be changed according; we might make it one time in a 24-hour, 48-hour or whenever needed. Therefore, the data what we got in the result shows that the implementation of the solution is done based on the design by considering the two pillars (dynamic threshold and elasticity) successfully. The next question is,

how can we make sure that these two pillars can help to achieve what this framework propose to achieve i.e. increase QoS and increase resource utilization efficiency? The discussion part is specifically discussed about this question.

5.8. Discussion

One of the main properties of cloud monitoring is elasticity [13]. Elasticity is the competence to increase and decrease computational resources on demand, according to the goal of a specific application or system [8]. One research paper mention that it's possible to do elasticity automatically but it does not specify how [40]. Elasticity is done by cloud providers in public cloud scenario and by users in private cloud; but here in this proposed solution the elasticity is done by the monitoring system by itself automatically. To support elasticity, the monitoring system needs to track virtual resources created/destroyed by expanding/contracting a cloud and to correctly handle expansion/retraction of the system [8]. After the provisioning system provisioned the resource based on the calculated computational resources the index number of the resource (refer Chapter 4 for node index) will return to the monitoring system. The monitoring system will incorporate the new index on the normal monitoring process automatically by applying the right monitoring metrics. A paper which done a survey on monitoring tools compare 9 commercial and 8 opensource monitoring tools regarding monitoring properties [13]. Only 3 of commercial and 2 of opensource tools can track created/destroyed resource and monitor accordingly but not calculating the amount of created/destroyed resource.

Elasticity is a key point to add Quality of Service (QoS) into cloud services [40]. Management of cloud environments involves building strategies to make it elastic to ensure QoS and to use resources efficiently. At the same time elasticity increase resource utilization efficiency, because resources are allocated to nodes based on the actual resource demand. Elasticity helps to improve the use of resources, essential in a cloud computing environment, since it avoids the waste of resources, reducing costs [40].

The value of the threshold has an important impact on monitoring systems. It would generate unprecedented floods of false alarms if the alert threshold is set to a conservative value (static threshold), while it would emit some important information if the alert threshold is set to a

progressive value (dynamic threshold) [14]. Dynamic threshold is the base for this solution, it helps to decrease false alarm during monitoring and used to implement elasticity.

This research work intended two answer two research questions, below the result with respect to each research questions will discuss:

RQ1: How can we enhance existing resource utilization efficiency and QoS in private cloud monitoring by using dynamic threshold as system alert condition?

- Cloud environment is a very dynamic environment regarding resource utilization hence it's important to monitoring this environment in such way that the dynamic behaviors are fully considered. Monitoring thresholds play important role in monitoring, in this proposed solution the threshold will change over time based on the utilization of monitored nodes. These dynamic threshold helps to realize an important monitoring property, elasticity. The resource utilization of a given node is changed over time and based on the resource demand the resource will give. Through elasticity, this research work proofs that QoS and resource utilization efficiency has increased. Therefore, using dynamic threshold as alerting condition can enhance resource utilization efficiency and QoS of private clouds.

RQ2: How dynamic threshold in private cloud monitoring can be affected by private cloud owner's business behavior and policies/rules?

- Applications used in private cloud are used to achieve the business requirement of that specific company. For instance, financial institutions use applications which are related to the organization business behaviors. These applications and internal policy/rules related to resource utilization has their own impact on the dynamic threshold. In this proposed solution five elements have been identified which has influence in dynamic threshold processing. These are four historical utilization data: the last 1hour utilization data, 1day utilization data, weekly utilization data and monthly utilization data. These historical determine the resource utilization of a given node so it should consider while calculating the dynamic threshold. The other element which should considered is dependency, resource utilization of nodes might be inter dependent. Systems in one organization are interconnected and work integrally, so resource utilization of a given node might determine by another node utilization this also considered in dynamic threshold process. The other element is policy/rule, organizations resources

utilization policy/rule should also one ingredient to calculate the dynamic threshold. Figure 22 shows functional elements and features of the new proposed solution; all features are depending on the dynamic threshold.

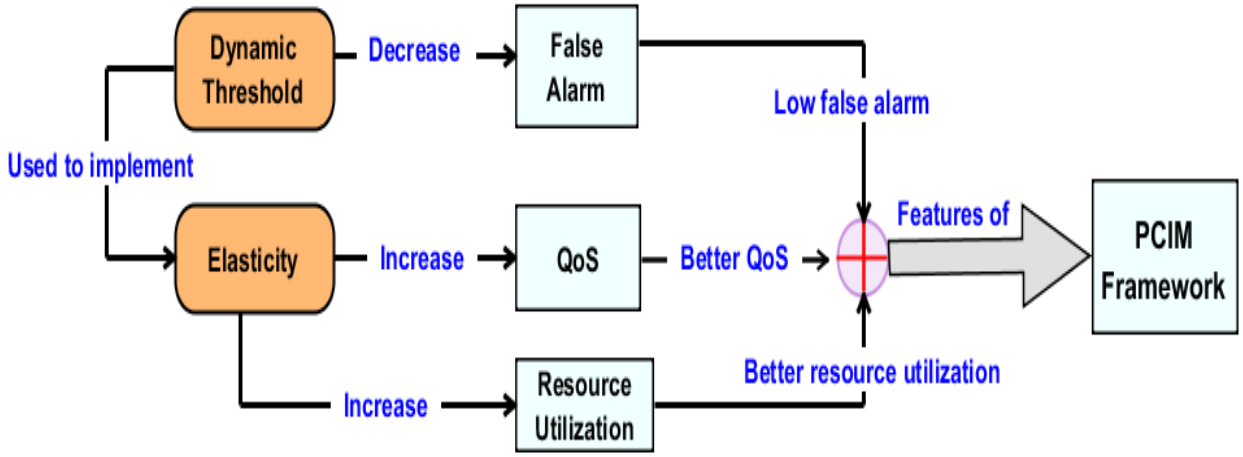


Figure 22: PCIM framework functional elements and its features

Dynamic threshold trigger elasticity and then elasticity help to increase QoS and resource utilization. Therefore, better QoS, better resource utilization and low false alarm are happened to be the newly added features of private cloud infrastructure monitoring framework.

5.9. Summary

This Chapter mainly focus on implementation and evaluation of the proposed framework, it also discusses about evaluation setup and considerations. One thing we should understand here is that, the considerations did not affect the solution to do in any set of cloud environment. For instance, this research work considers that there are four monitored nodes, it defines the dependency matrix and it also defines which historical data (recent, daily, weekly and monthly) belongs to which node. These all considerations are not the only considerations to evaluate the system, we could have whatever considerations we want to evaluate the proposed solution; and the considerations did not show the boundaries of this work.

Chapter 6: Conclusion and Recommendation

6.1. Conclusion

This thesis work is focused on introducing new private cloud infrastructure monitoring framework; the design, implementation and evaluation of the new proposed solution has been discussed on previous Chapters. Number of different research papers, white papers, books and websites has referred to understand state of the art and to point out research gaps which is not covered by other researches yet. This work is intended to proof the hypothesis: "it's possible to enhance resource utilization efficiency and QoS of private cloud by using dynamic threshold as system alert condition and automatic elasticity to manage resource utilization".

The hypothesis has two major points, the first one is enhancing resource utilization and the other point is QoS, and the new proposed system can grant these two points. The solution help to monitor the dynamic environment of the cloud with dynamic threshold, the threshold will go up and down based on the resource utilization of a monitored nodes. In addition, new resource might be provisioned or deduct from a node (automatic elasticity) based on the utilization; this dynamic based monitoring help to enhance resource utilization efficiency.

In the other hand QoS has direct correlation with elasticity, elasticity increases the QoS. To insure the QoS, this proposed solution introduces a module called 'elasticity management' this module is generating provisioning orders based on the dynamic threshold. The generated order will give additional resource for a node or will deduct the resource and make it back to the resource pool so that others can use it. This monitoring system determine resource demand of a given node ahead of time which can help to achieve two things; one it minimizes unused resources of a given node by taking back to the resource pool. The other thing is it makes sure that a monitored node is getting the required amount of resource by initiating and sending new provisioning order to provisioning system.

This research work has its own contribution in the area; the first contribution is, it introduces new insight regarding private cloud infrastructure monitoring. Normally in cloud environment, infrastructure is expensive to purchase and implement; hence, monitoring this environment in such a way that utilization efficiency can be increased is very important and this is the other contribution

of this work. The solution is specifically for private cloud environment owners, and this solution will help them to realize a better resource utilization and QoS.

6.2. Recommendation

This solution is design for private cloud environment, any private owned cloud environment may use this monitoring solution. Especially in our country actual cloud environment is not deployed yet but most companies may go for cloud in near future. Most companies will adopt private cloud because:

- Private cloud is a secure environment than other deployment models.
- There are no public cloud providers in Ethiopia, so companies should deploy their systems abroad and that raise trust issue (system and data may obtain by third parties since the environment is shared by many).

So, beside the above two reasons this solution will help them to realize better resource utilization efficiency and QoS. The same is true for ethio telecom, currently ethio telecom has many distributed data centers which have no communication between them and has no resource sharing among them. In the current situation, implementing cloud seems mandatory for ethio telecom and this solution will help to monitor the cloud. Ethio telecom has many systems which are dependent with each other and there are systems whose utilization is determined by historical utilization data. So, this research paper highly recommended ethio telecom to adopt this solution when the company deploys private cloud.

6.3. Future work

This research shows the integration of the proposed solution with open SUSE monitoring architecture. Hence, in the future the proposed framework should integrate with other widely used monitoring architectures like Nagios and other architecture. The integration makes this proposed system more usable and accessible. It is also good to test this proposed solution in real private cloud environment or cloud lab. The evaluation is done using actual cloud utilization data but it’s good to test the solution with real environment in the future.

References

[1] S. Naidila and K. Dilip, "Cluster, Grid and Cloud Computing: A Detailed Comparison," *The 6th International Conference on Computer Science & Education (ICCSE 2011),* 2011.

[2] H. Michael, L. Fang, S. Annie and T. Jin, "NIST Cloud Computing Standards Roadmap," *National Institute of Standards and Technology U. S. Department of Commerce,* July 2011.

[3] Y. Haibo and T. Mary, "A Descriptive Literature Review and Classification of Cloud Computing Research," *Communications of the Association for Information Systems,* vol. 31, pp. 35-60, July 2012.

[4] R. Sharma, "The Impact of Virtualization in Cloud Computing," *International Journal of Recent Development in Engineering and Technology,* vol. 3, no. 1, July 2014.

[5] L. Malhotra, D. Agarwal and A. Jaiswal, "Virtualization in Cloud Computing," *Journal of Information Technology & Software Engineering,* vol. 4, no. 2, 2014.

[6] K. Rajwinder and L. Pawan, "Load Balancing in Cloud Computing," *Association of Computer Electronics and Electrical Engineers,* 2014.

[7] stackify, "www.stackify.com," stackify, [Online]. Available: https://stackify.com/cloud-monitoring/. [Accessed 23 August 2018].

[8] R. Guilherme, C. Rodrigo, G. Vinicius, d. S. Gledersson, d. C. Márcio, G. Lisandro, R. Liane and B. Rajkumar, "Monitoring of Cloud Computing Environments: Concepts, Solutions, Trends, and Future Directions," *dl.acm.org,* April 2016.

[9] S. Hassan, G. Abdullah, A. Raja, K. Muhammad and A. Abdelmuttlib, "Cloud monitoring: A review, taxonomy, and open research issues," *Elsevier,* vol. 98, pp. 11-26, 2017.

[10] C. Shirlei, U. Rafael and W. Carlos, "Toward an Architecture for Monitoring Private Clouds," *IEEE,* December 2011.

[11] Alamela and Karthik, "www.quora.com," quora, June 2009. [Online]. Available: https://www.quora.com/What-is-cloud-infrastructure-monitoring#. [Accessed 23 August 2018].

[12] A. Juan, C. Jose and V. Wladimiro, "IaaSMon: Monitoring Architecture for Public Cloud Computing Data Centers," *Springer,* vol. 14, p. 283–297, 2016.

[13] A. Giuseppe, B. Alessio, d. D. Walter and P. Antonio, "Cloud monitoring: A survey," *Elsevier,* 2013.

[14] Z. Xinkui, Y. Jianwei, Z. Chen and C. Zuoning, "SimMon: a toolkit for simulation of monitoring mechanisms in cloud computing environment," *Wiley Online Library,* 2016.

[15] visual-paradigm, "www.visual-paradigm.com," visual-paradigm, [Online]. Available: https://www.visual-paradigm.com/support/faq.jsp. [Accessed 2017].

[16] Oracle, "www.netbeans.org," NetBeans, [Online]. Available: https://netbeans.org/about/.

[17] S. Siqi, B. Vincent and I. Alexandru, "Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters," *IEEE,* 2015.

[18] ciowhitepapers, "www.ciowhitepapersreview.com," [Online]. Available: https://whatis.ciowhitepapersreview.com/definition/cloud-infrastructure/.

[19] CDW, "Infrastructure as a service," *CDW LLC,* 2012.

[20] A. Giuseppe, B. Alessio, d. D. Walter and P. Antonio, "Cloud Monitoring: definitions, issues and future directions," *IEEE,* 2012.

[21] C. Jose and G. Juan, "Comparative analysis of architectures for monitoring cloud computing infrastructures," *Future Generation Computer Systems,* 2015.

[22] C. Simin, G. Barbara, N. Dag, S. Cristina and L. Alf, "Design of Cloud Monitoring Systems via DAGGTAX: a Case Study," *Elsevier,* pp. 424-431, 2017.

[23] K. Manoj, D.-N. Duc-Tien, H. Markus and B. Marija, "An Overview of User-level Usage Monitoring in Cloud Environment," *doras.dcu.ie,* 2018.

[24] L. Mingwei, Y. Zhiqiang and H. Tianqiang, "A hybrid push protocol for resource monitoring in cloud computing platforms," *Elsevier,* 2015.

[25] P. Dirk, "Monitoring of Private Clouds," *Paessler AG,* 2011.

[26] M. Francisco and d. Osvaldo, "Dynamic threshold (over dynamic load-balancing mechanism) on mimd architecture," *International Nuclear Atlantic Conference,* 2007.

[27] W. Jonathan and B. Adam, "Observing the clouds: a survey and taxonomy of cloud monitoring," *Journal of Cloud ComputingAdvances, Systems and Applications,* 2014.

[28] K. Mikyung, K. Dong-In, Y. Mira, P. Gyung-Leen and L. Junghoon, "Design for Run-Time Monitor on Cloud Computing," *Springer,* pp. 279-287, 2010.

[29] A. Patricia, R.-M. José, M. José and A. José, "Heuristics and metaheuristics for dynamic management of computing and cooling energy in cloud data centers," *wileyonlinelibrary,* pp. 1-30, 2018.

[30] A. Jalal, L. Ioannis, K. George, U. Bhuvan and Z. Yiqiang, "An Efficient and Fair Multi-Resource Allocation Mechanism for Heterogeneous Servers," *IEEE,* 2018.

[31] suse, "www.suse.com," suse, 2018. [Online]. Available: https://www.suse.com/documentation/suse-openstack-cloud-7/. [Accessed 4 June 2018].

[32] T. O. Dashboard, "www.openstack.com," OpenStack, [Online]. Available: https://docs.openstack.org/horizon/latest/. [Accessed 2018].

[33] D. U. o. Technology, The Grid Workloads Archive, [Online]. Available: http://gwa.ewi.tudelft.nl/datasets/Bitbrains. [Accessed 2018].

[34] C. Rodrigo, R. Rajiv, B. Anton, D. R. Cesar and B. Rajkumar, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Wiley Online Library,* pp. 23-50, 2011.

[35] greencloud, "www.greencloud.gforge.uni.ln," Journal of Supercomputing, 2012. [Online]. Available: http://greencloud.gforge.uni.lu/index.html. [Accessed July 2018].

[36] icancloud, "www.arcos.inf.uc3m.es," 16 February 2015. [Online]. Available: https://www.arcos.inf.uc3m.es/old/icancloud/Home.html. [Accessed July 2018].

[37] L. Zhihua, Y. Chengyu, Y. Lei and Y. Xinrong, "Energy-aware and multi-resource overload probability constraint-based virtual machine dynamic consolidation method," *Elsevier,* September 2017.

[38] L. YAQIU, S. XINYUE, W. WEI and J. WEIPENG, "Enhancing Energy-Efficient and QoS Dynamic Virtual Machine Consolidation Method in Cloud Environment," *IEEE,* June 2018.

[39] Z. Szabo, M. Adam and Mate, "Which is the best algorithm for virtual machine placement optimization?," *Wiley Online Library,* 2017.

[40] C. Emanuel, S. Flávio , R. Paulo, G. Danielo and S. José, "Elasticity in cloud computing: a survey," *Springer,* vol. 70, no. 7-8, p. 289–309, August 2015.

Appendix

Appendix A: Java implementation result with random input considering only history parameter

run:

Dynamic threshold processor is running ...

CU 1

=========

a	b	c	d
8.0	9.0	11.0	3.0

PU 1

=========

a	b	c	d
0.0	0.0	0.0	0.0

PT 1

=========

a	b	c	d
0.0	0.0	0.0	0.0

CT 1

=========

a	b	c	d
6.0	6.01	6.12	5.96

Dynamic threshold processor is running ...

CU 2

=========

a	b	c	d
7.0	4.0	3.0	10.0

PU 2

=========

a	b	c	d
8.0	9.0	11.0	3.0

PT 2

=========

a	b	c	d
6.0	6.01	6.12	5.96

CT 2

=========

a	b	c	d
5.5	6.19	6.00	5.99

Dynamic threshold processor is running ...

CU 3

=========

a	b	c	d
2.0	4.0	6.0	10.0

PU 3

=========

a	b	c	d
7.0	4.0	3.0	10.0

PT 3

=========

a	b	c	d
5.5	6.19	6.00	5.99

CT 3

=========

a	b	c	d
6.67	5.56	5.98	5.97

Dynamic threshold processor is running ...

CU 4

=========

a	b	c	d

2.0 6.0 6.0 7.0

PU 4

=========

a b c d

2.0 4.0 6.0 10.0

PT 4

=========

a b c d

6.67 5.56 5.98 5.97

CT 4

=========

a b c d

7.08 6.06 6.11 5.96

Threshold Management is running ...

Here is notification from threshold management to users and admins.

Node a needs upgrade

Node b do not need any elasticity

Node c needs upgrade

Node d do not need any elasticity

.....

Elasticity action node is running ...

Below is notification from elasticity action node.

Additionsl 350 mips cpu resources will added to node a

No elasticity is take for node b

Additionsl 350 mips cpu resources will added to node c

No elasticity is take for node d

The policy is checking for each node ...

This notification should take seriously !!!

==========

According to the policy node a is in normal stage regarding resource utilization.

WARNING: According to the policy node b is in critical stage regarding resource utilization.

According to the policy node c is in normal stage regarding resource utilization.

According to the policy node d is in normal stage regarding resource utilization.

BUILD SUCCESSFUL (total time: 0 seconds)

Appendix B: Java implementation result with bitbrain input considering only history parameter

run:

Dynamic threshold processor is running ...

CU 1

==========

a	b	c	d
4.43	4.43	3.0	1.0

PU 1

==========

a	b	c	d
0.0	0.0	0.0	0.0

PT 1

==========

a	b	c	d
0.0	0.0	0.0	0.0

CT 1

=========

a b c d

4.43 3.79 2.76 1.01

Dynamic threshold processor is running ...

CU 2

=========

a b c d

0.53 4.83 5.5 0.73

PU 2

=========

a b c d

4.43 4.43 3.0 1.0

PT 2

=========

a b c d

4.43 3.79 2.72 1.01

CT 2

=========

a b c d

0.53 5.10 4.66 1.01

Dynamic threshold processor is running ...

CU 3

=========

a b c d

1.53 5.73 4.13 57.17

PU 3

=========

a b c d

0.53 4.83 5.5 0.73

PT 3

=========

a b c d

0.53 5.10 4.66 1.01

CT 3

=========

a b c d

1.53 5.36 2.54 39.15

Dynamic threshold processor is running ...

CU 4

=========

a b c d

3.87 2.07 3.77 11.97

PU 4

=========

a b c d

1.53 5.73 4.13 57.17

PT 4

=========

a b c d

1.53 5.36 2.54 39.15

CT 4

=========

a b c d

3.9 3.18 5.09 2.89

Threshold Management is running ...

Here is notification from threshold management to users and admins.

```
     ====================
Node a needs downgrade
Node b do not need any elasticity
Node c do not need any elasticity
Node d needs upgrade
====================================

 Elasticity action node is running ...

Below is notification from elasticity action node.
     =================
150 mips cpu resources will remove from node a
No elasticity is take for node b
No elasticity is take for node c
 Additionsl 350 mips cpu resources will added for  node d

 The policy is checking for each node ...

This notification should take seriously !!!
   ================================

 WARNING: According to the policy node a is in critical stage regarding resource utilization.

  According to the policy node b is in normal stage regarding resource utilization.

  According to the policy node c is in normal stage regarding resource utilization.

  According to the policy node d is in normal stage regarding resource utilization.
BUILD SUCCESSFUL (total time: 3 seconds)
```

Appendix C: Java implementation result with random input considering only dependency parameter

```
run:

Dynamic threshold processor is running ...
    CU 1
  =========
a       b       c       d
10.0    6.0     4.0     10.0
    PU 1
  =========
a       b       c       d
0.0     0.0     0.0     0.0
    PT 1
  =========
a       b       c       d
4.43    6.0     2.72    1.01
    CT 1
  =========
a       b       c       d
4.70    6.36    2.88    1.07

Dynamic threshold processor is running ...
    CU 2
  =========
a       b       c       d
7.0     9.0     4.0     3.0
    PU 2
  =========
a       b       c       d
10.0    6.0     4.0     10.0
```

PT 2

=========

a b c d

4.70 6.0 2.72 1.07

CT 2

=========

a b c d

4.98 6.36 2.88 1.13

Dynamic threshold processor is running ...

CU 3

=========

a b c d

9.0 9.0 6.0 7.0

PU 3

=========

a b c d

7.0 9.0 4.0 3.0

PT 3

=========

a b c d

4.70 6.0 2.88 1.07

CT 3

=========

a b c d

4.98 6.36 2.94 1.13

Dynamic threshold processor is running ...

CU 4

=========

a b c d

10.0 9.0 5.0 9.0

PU 4

=========

a b c d

9.0 9.0 6.0 7.0

PT 4

=========

a b c d

4.70 6.0 2.88 1.07

CT 4

=========

a b c d

4.98 6.36 2.93 1.13

Threshold Management is running ...

Here is notification from threshold management to users and admins.

Node a do not need any elasticity

Node b needs upgrade

Node c needs downgrade

Node d needs downgrade

.....

Elasticity action node is running ...

Below is notification from elasticity action node.

No elasticity is take for node a

Additionsl 350 mips cpu resources will added to node b

150 mips cpu resources will remove from node c

150 mips cpu resources will remove from node d

The policy is checking for each node ...

This notification should take seriously !!!

==========

According to the policy node a is in normal stage regarding resource utilization.

According to the policy node b is in normal stage regarding resource utilization.

According to the policy node c is in normal stage regarding resource utilization.

WARNING: According to the policy node d is in critical stage regarding resource utilization.

BUILD SUCCESSFUL (total time: 0 seconds)

Appendix D: Java implementation result with bitbrain input considering only dependency parameter

run:

Dynamic threshold processor is running ...

CU 1

==========

a	b	c	d
4.43	4.43	3.0	1.0

PU 1

==========

a	b	c	d
0.0	0.0	0.0	0.0

PT 1

==========

a b c d

0.0 0.0 0.0 0.0

CT 1

=========

a b c d

4.43 3.79 2.72 1.01

Dynamic threshold processor is running ...

CU 2

=========

a b c d

0.53 4.83 5.5 0.73

PU 2

=========

a b c d

4.43 4.43 3.0 1.0

PT 2

=========

a b c d

0.0 0.0 0.0 0.0

CT 2

=========

a b c d

4.43 3.79 2.72 1.01

Dynamic threshold processor is running ...

CU 3

=========

a b c d

1.53 5.73 4.13 57.17

PU 3

=========

a b c d

0.53 4.83 5.5 0.73

PT 3

=========

a b c d

4.43 0.0 2.72 0.0

CT 3

=========

a b c d

4.70 3.79 2.77 1.01

Dynamic threshold processor is running ...

CU 4

=========

a b c d

3.87 2.07 3.77 11.97

PU 4

=========

a b c d

1.53 5.73 4.13 57.16

PT 4

=========

a b c d

4.43 0.0 2.72 0.0

CT 4

=========

a b c d

4.70 3.79 2.77 1.01

Threshold Management is running ...

Here is notification from threshold management to users and admins.

==========

Node a do not need any elasticity

Node b needs downgrade

Node c needs downgrade

Node d needs downgrade

==========

Elasticity action node is running ...

Below is notification from elasticity action node.

==========

No elasticity is take for node a

150 mips cpu resources will remove from node b

150 mips cpu resources will remove from node c

150 mips cpu resources will remove from node d

The policy is checking for each node ...

This notification should take seriously !!!

==========

WARNING: According to the policy node a is in critical stage regarding resource utilization.

According to the policy node b is in normal stage regarding resource utilization.

According to the policy node c is in normal stage regarding resource utilization.

According to the policy node d is in normal stage regarding resource utilization.

BUILD SUCCESSFUL (total time: 1 second)

Appendix E: Java implementation result with random input considering all parameter

run:

Dynamic threshold processor is running ...

CU 1

=========

a	b	c	d
8.0	9.0	11.0	6.0

PU 1

=========

a	b	c	d
0.0	0.0	0.0	0.0

PT 1

=========

a	b	c	d
6.42	9.0	5.91	5.99

CT 1

=========

a	b	c	d
6.80	9.54	6.26	6.35

Dynamic threshold processor is running ...

CU 2

=========

a	b	c	d
7.0	11.0	7.0	7.0

PU 2

=========

a	b	c	d
8.0	9.0	11.0	6.0

PT 2

=========

a b c d

6.80 9.54 6.26 6.35

CT 2

=========

a b c d

6.58 5.99 6.04 6.02

Dynamic threshold processor is running ...

CU 3

=========

a b c d

3.0 3.0 5.0 4.0

PU 3

=========

a b c d

7.0 11.0 7.0 7.0

PT 3

=========

a b c d

6.58 5.99 6.05 6.02

CT 3

=========

a b c d

7.75 5.67 5.87 6.05

Dynamic threshold processor is running ...

CU 4

=========

a b c d

10.0 8.0 8.0 9.0

PU 4

==========

a b c d

3.0 3.0 5.0 4.0

PT 4

==========

a b c d

5.5 8.0 5.87 5.99

CT 4

==========

a b c d

5.83 8.48 6.22 6.35

Threshold Management is running ...

Here is notification from threshold management to users and admins.

Node a do not need any elasticity

Node b needs upgrade

Node c needs upgrade

Node d needs upgrade

.....

Elasticity action node is running ...

Below is notification from elasticity action node.

No elasticity is take for node a

Additionsl 350 mips cpu resources will added to node b

Additionsl 350 mips cpu resources will added to node c

Additionsl 350 mips cpu resources will added to node d

The policy is checking for each node ...

This notification should take seriously !!!

==========

WARNING: According to the policy node a is in critical stage regarding resource utilization.

WARNING: According to the policy node b is in critical stage regarding resource utilization.

WARNING: According to the policy node c is in critical stage regarding resource utilization.

WARNING: According to the policy node d is in critical stage regarding resource utilization.

BUILD SUCCESSFUL (total time: 1 second)

Appendix F: Java implementation result with bitbrain input considering all parameter

run:

Dynamic threshold processor is running ...

CU 1

==========

a	b	c	d
4.43	4.43	3.0	1.0

PU 1

==========

a	b	c	d
0.0	0.0	0.0	0.0

PT 1

==========

a	b	c	d
0.0	0.0	0.0	0.0

CT 1

=========

a b c d

4.43 3.79 2.72 1.01

Dynamic threshold processor is running ...

CU 2

=========

a b c d

0.53 4.83 5.5 0.73

PU 2

=========

a b c d

4.43 4.43 3.0 1.0

PT 2

=========

a b c d

4.43 3.79 2.72 1.01

CT 2

=========

a b c d

0.53 5.10 4.66 1.01

Dynamic threshold processor is running ...

CU 3

=========

a b c d

1.53 5.73 4.13 57.17

PU 3

=========

a b c d

0.53 4.83 5.5 0.73

PT 3

=========

a b c d

1.53 5.10 2.54 1.01

CT 3

=========

a b c d

1.63 5.36 2.59 39.15

Dynamic threshold processor is running ...

CU 4

=========

a b c d

3.87 2.07 3.77 12.00

PU 4

=========

a b c d

1.53 5.73 4.13 57.17

PT 4

=========

a b c d

1.63 5.36 2.60 39.15

CT 4

=========

a b c d

3.9 3.18 5.09 2.89

Threshold Management is running ...

Here is notification from threshold management to users and admins.

==========

Node a needs downgrade

Node b do not need any elasticity

Node c do not need any elasticity

Node d needs upgrade

==========

Elasticity action node is running ...

Below is notification from elasticity action node.

==========

150 mips cpu resources will remove from node a

No elasticity is take for node b

No elasticity is take for node c

Additionsl 350 mips cpu resources will added for node d

The policy is checking for each node ...

This notification should take seriously !!!

==========

WARNING: According to the policy node a is in critical stage regarding resource utilization.

According to the policy node b is in normal stage regarding resource utilization.

According to the policy node c is in normal stage regarding resource utilization.

According to the policy node d is in normal stage regarding resource utilization.

BUILD SUCCESSFUL (total time: 2 seconds)

Appendix G: Comparison of cluster, grid and cloud computing paradigms

	Clusters	*Grids*	*Clouds*
SLA	Limited	Yes	Yes
Allocation	Centralized	Decentralized	Both
Resource Handling	Centralized	Distributed	Both
Loose coupling	No	Both	Yes
Protocols/API	MPI, Parallel Virtual	MPI,MPICH-G, GIS,GRAM	TCP/IP,SOAP, REST,AJAX
Reliability	No	Half	Full
Security	Yes	Half	No
User friendliness	No	half	Yes
Virtualization	Half	Half	Yes
Interoperability	Yes	Yes	Half
Standardized	Yes	Yes	No
Business Model	No	No	Yes
Task Size	Single large	Single large	Small & medium
SOA	No	Yes	Yes
Multitenancy	No	Yes	Yes
System Performance	Improves	Improves	Improves
Self service	No	Yes	Yes
Computation service	Computing	Max. Computing	On demand
Heterogeneity	No	Yes	Yes
Scalable	No	Half	Yes
Inexpensive	No	No	Yes
Data Locality Exploited	No	No	Yes
Application	HPC,HTC	HPC, HTC, Batch	SME interactive apps.

Figure 17: comparison of cluster vs grid vs cloud [1]

Declaration

The thesis entitled "A Framework for Private Cloud Infrastructure Monitoring" submitted to Addis Ababa Institute of Technology; in partial fulfilment of the requirement for the award of MSC degree in Telecommunication Engineering is conducted under the supervision of Dr. Mesfin Kifle.

I hereby declare that the information reported in the paper is the result of my own work, except where due to references is made. The thesis has not been accepted for any research work before and is not concurrently submitted by any candidates.

Selamawit Belete

October 2018