A UNIFIED FRAMEWORK FOR SERVICE AVAILABILITY AND WORKFLOW SCHEDULING IN EDGE COMPUTING ENVIRONMENT

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Engineering

in

Software Engineering

Submitted By

Tanya Dhand
(Roll No: 801531015)

Under the Supervision of:

Dr. Neeraj Kumar
Associate Professor
CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY,
PATIALA – 147004

JULY 2017

Certificate

I hereby certify that the work which is being presented in the thesis entitled, ***"A Unified Framework for Service Availability and Workflow Scheduling in Edge Computing Environment"***, in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Neeraj Kumar*. The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Tanya Dhand
801531015

Place: Patiala
Date:

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Neeraj Kumar)
Supervisor
Associate Professor
Department of CSE
Thapar University
Patiala (Punjab), India

Acknowledgment

I have been waiting long for this moment to acknowledge all those who contributed in building this work. It is my pleasure to thank all of them here. First of all, I offer my sincere gratitude to my supervisor *Dr. Neeraj Kumar* for accepting to be my supervisors. Without their help and encouraging advice, this work would have never begun. I am deeply indebted to them for providing me wonderful research atmosphere and platform to explore my research to the fullest.

I am also grateful to *Dr. Maninder Singh*, Head, CSED for providing me the opportunity to conduct my research work. I would also like to thank the Director of the institute, *Prof. Prakash Gopalan* for his continuous support.

My special thanks goes to my friends for discussing thoughts and sharing all ups and downs with me during the course of this work. At the same time I would also like to thank all my colleagues for their continuous support.

Last but not the least I would like to thank my parents and family members, who made me capable of reaching this point of life and for giving me their kind support and love. I dedicate my work to them.

Tanya Dhand
(801531015)

Abstract

Fog Computing is a new emerging promising paradigm, extending cloud technological resources near to fog edge devices. Fog computing works to give services in highly distributed fashion that has captured recent research and industry trend. Fog platform provides advantage over traditional cloud platform like providing location-based processing, additional intelligence (smart gateways), lower bandwidth consumption, much faster and cheaper to use, real time data delivery and low latency services. The goal of fog computing is not to prove itself better then cloud platform, it just provide real time task and workflow computation better then cloud computing at the network edge that is closer to devices, with that also to make system perform better.

The task scheduling and workflow scheduling are two major issues that gained enormous attention and need to be addressed over the fog platform. The task scheduling deals with allocating best fit resources (computation, storage and network resources), data processing and service availability at edge of network. To handle the task scheduling issues, we present a novel architecture for task selection and scheduling at the edge of the network using container as a service (CoaaS). Solving this issue by using cooperative game theory. For that we developed a multi-objective function in order to reduce the energy consumption and makespan by considering different constraints such as memory, CPU, and the users budget. For task selection and scheduling, we have used lightweight containers instead of the conventional virtual machines to reduce the overhead and response time as well as the overall energy consumption of fog devices, that is, nano data centers (nDCs).

On other hand scheduling of dynamic dependable multi-task jobs also known as workflow over fog network needs a reliable execution. In the traditional method over cloud network, workflow scheduling deals with various issues like high probability of communication and computing bottlenecks, large data transmission time and failure issues. In addition the smart end users always desire for location-based processing for real time workflows. Again and again accessing the centralized cloud platform to retrieve localized data leading to an inefficient act.

So there is need to investigate strategy for reliable and real time workflow scheduling and solving above issues. For that VM clustering strategy is investigated over edge network, that reduces the execution and scheduling overhead and providing QoS aware workflow

scheduling. We present the detailed architecture of workflow scheduling model over Fog-Cloud layer, that integrates the foglets and edge computing platform for better service scheme. We also design an algorithm based on dividing the workflow into sub tasks and finding the most optimal VM cluster for workflow scheduling.

Keywords: Fog computing, edge network, smart gateways, Quality of Service (QoS), Cloud network, container as a service (CoaaS).

Contents

List of Figures

List of Tables

List of Symbols

T	Task set
W	Workflow
η_i	Number of containers
α	Available number of containers
N	Available Broker
B_j	Current bandwidth
W_j	Load status
μ_j	Utility function of brokers
W_j^{max}	Maximum load capacity
M	Task requests
P	Containers
Q	VMs available at the nDC level
i^{th}	Task request
A_i	Maximum completion time
k^{th}	Task request of container
$F(x)$	The total energy consumption of the nDC
S	Feasible solutions
S	Available search space
$c_i(x)$	Constraint parameters
$m_i(x)$	Constraint parameters
$b_i(x)$	Constraint parameters
$\beta_{x,k}$	CPU limit
$\gamma_{x,k}$	Memory limit
$\Lambda_{x,k}$	Budget limit
μ_k	Utility function of the competing players
Δ	Task distribution matrix
β	CPU allocation matrix
E_{act}	Total energy consumption of nDC
E_{idle}	Total energy consumption of nDC
P_{act}	nDC's power consumption at active mode
P_{idle}	nDC's power consumption at idle mode
T_{act}	nDC's power consumption and related time slots
T_{idl}	nDC's power consumption and related time slots
δ_k	Total number of tasks mapped

θ_k	Number of processors assigned
d_k	Data size of tasks mapped
β_k	Processing rate
p_k	Task's processing time
c_k	Task's cost
$\mathscr{S}$	Set of stages in the game
VM_{Ava}	VM availability
V_T	Total work done by VMs
B_{VM}	Optimum busy time period of VMs.
μ_i	Numerical value
D_{ij}	Euclidean distance
V_x	Virtual nodes
fs	Fog System
SN_i	Storage node
CN_j	CPU node
U_{CPU}	Unused CPU capacity
$U_{storage}$	Unused storage capacity
A_{CPU}	Available CPU capacity
$A_{storage}$	Available storage capacity
A'_{CPU}	Allocated CPU capacity
$A'_{storage}$	Allocated storage capacity
α	Calculated available CPU capacity
β	Calculated available storage capacity
γ	Allocated CPU capacity calculated
δ	Allocated storage capacity calculated
U_{CPU}	Calculated unused CPU capacity
$U_{storage}$	Calculated unused Storage capacity
$nDC_{capacity}$	Nano data center formed by unused VM space
$C_{capacity}$	Cluster capacity
D_i	Input data set
σ_i	Centroid calculation

List of Abbreviations

ICT	Information and Communication Technology
IoT	Internet of Things
CRP	Cloud Resource Provider
QoS	Quality of Service
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
DC	Data Center
VM	Virtual Machine
SDK	Software Development Kit
API	Application Programming Interface
nDC	Nano Data Center
SLA	Service Level Agreement
RASA	Resource Aware Scheduling Algorithm
PSO	Particle Swarm Optimization
BSO	Best Resource Selection Algorithm
PCSO	Parallel Cat Swarm Optimization
AHP	Analytical Hierarchy Process
LIGO	Laser Interferometer Gravitational Wave Observatory
sRNAs	Small untranslated RNAs
SCEC	Southern California Earthquake Center
PSHA	Probabilistic Seismic Hazard Analysis
DVFS	Dynamic Voltage Frequency Scheduling Algorithm
EWSA	Efficient Workflow Scheduling Algorithm
RCL	Resource Consumer Layer
RCoL	Resource Collector Layer
CL	Controller Layer
BL	Broker Layer
CoL	Computation Layer
LAN	Local Area Network
TRM	Task Requirement Module
UTRM	User Task Requirement Module
URM	User Requirement Module

RRM	Resource Monitoring Module
CSM	Container Scheduling Module
OS	Operating System
WSS	Workflow Scheduling System
WDAG	Weighted Directed Acyclic Graph
CPU	Central Processing Unit
RAM	Random Access memory
RASA	Resource Aware Scheduling algorithm
RR	Round-robin algorithm
FCFS	First Come First Served
WSM	Workflow Scheduling Model

Chapter 1

Introduction

Rapid advancements in information and communications technology (ICT) along with an exponential increase in smart devices in the last few decades have led to the emergence of one of the most popular technologies called the Internet of Things (IoT). IoT consists of billions of interconnected devices for providing seamless services such as smart transportation, e-healthcare, smart education, and smart sensing to end users. Consequently, bulk data generation is inevitable from such a huge interconnection of heterogeneous and geographically distributed devices. It is also predict that almost 50 billion devices will be interconnected to the Internet by 2020 [2]. The current number of connected devices is already greater than the world population and is exponentially increasing, as shown in fig. 1.2. The huge amounts of data that will be produced by all these interconnected devices will need to be handled efficiently. Moreover, these large numbers of interconnections would also be associated with real-time network traffic in order to access different services such as smart transportation, e-health care, smart education, and smart sensing. This would also require seamless computation and processing of the data collected from various devices with heterogeneous capabilities. For that there is a new emerging technology [3] called as Cloud Computing.

1.1 Cloud Computing

Cloud computing is a popular virtualized computing platform [4] that give online services in the form of resources in pay per use manner. It has gained wide popularity due to its robust and reliable nature where user submit their application and get executable result without knowing inner implementation and hosting mechanism with minimal interaction. The resources that are supplied to user is in many form like network resources, storage resources and computation resources etc via web in consistent manner. The major re-

quirement for accessing the cloud services is the internet. The aim of cloud computing is to provide easy usable platform with flexible nature. The user submit its requirement of resources according to needs by specifying their QoS needed, availability, deadline and budget. Before giving services the agreement is signed between user and cloud resources provider (CRP), that must be followed to issue resources.

1.1.1 Service models of Cloud Computing

The cloud computing offers three service model [5] that are described below:

1. **Software as a service (SaaS)** : The capability offered by cloud computing to allow users to access application running on cloud platform. The applications are accessed from any remote devices by using internet services and pay how much you use. The consumer does not have to monitor underlined infrastructure and no issues of installing software. Examples are : Google Apps, Gmail.

2. **Platform as a service (PaaS)** : The capability offered by cloud computing to allow users (Software developer, web developer) to access environment and platform to develop their applications. PaaS platform is simple and easy to use. Example: Microsoft Windows Azure, Google App Engine.

3. **Infrastructure as a Service (IaaS)** : The capability offered by cloud computing to allow users to use computing resources like storage, network and servers. The consumer can built their own application by using any of platform over underlying infrastructure. Example: Amazon EC2, Google Computer Engine (GCE).

1.1.2 Deployment models of Cloud Computing

The cloud computing offers four deployment models [5] that are described below:

1. **Private cloud:** The cloud provided infrastructure is operated within an organization community. However it may be managed by third user or within organization users.

2. **Public cloud**: The cloud provided infrastructure is provided for general public by sharing common resources with huge general people.

3. **Community cloud**: The cloud provided infrastructure is shared among several users organizations having specific community. However it may be managed by third user or within organization users.

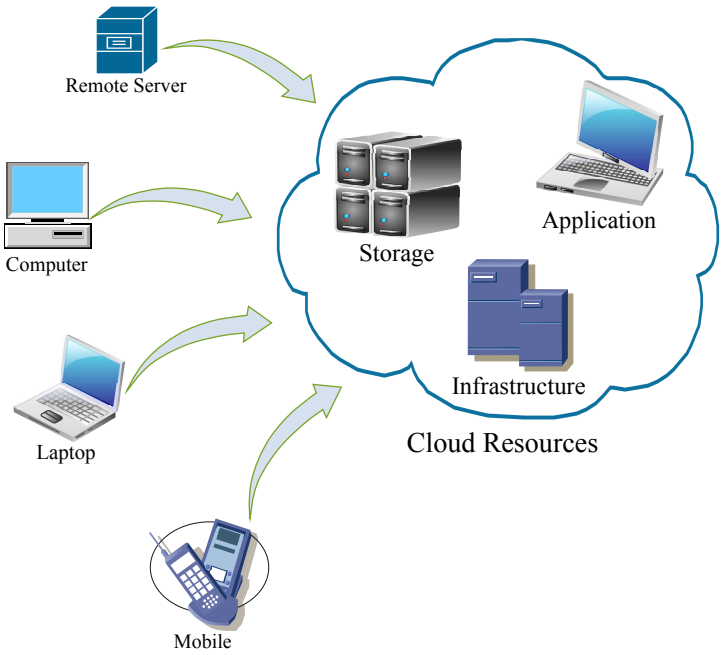

Figure 1.1: Cloud Computing platform.

4. **Hybrid cloud**: It is a composition of two cloud that is public cloud and private cloud. The use of these two clouds depends on data type either critical or non- critical data. The critical data to be assessed by private cloud and non-critical by public cloud.

1.2 Integrated challenges of Cloud Computing and IoT

The transmission of large amounts of data to the core cloud computing platform requires context-aware services [6] to be available to end users. For instance, owners of high-speed vehicles may be interested in knowing the best route (e.g., the one with the least traffic) to their destination. In contrast, other users might be interested in managing their home appliances remotely based on the current demand-response mechanism. In short, the integration of IoT with the cloud computing platform is necessary for efficient processing of stored data at the cloud repository. The cloud data centers are robust with respect to handling a diverse range of computations for various applications.

However, the transfer of large chunks of data to the remote cloud platform from the mobile nodes needs higher bandwidth, which in turn may cause higher latency [7] and network overhead. This in turn poses additional challenges such as achieving lower response time and high service availability. A recent survey conducted by Gartner, nearly a quarter billion vehicles will be connected to the Internet by 2020. These vehicles are expected

to become an important component of the future IoT-based infrastructure for providing in-vehicle services and automated driving capabilities. These statistics indicate that the number of consumers and the data they generate are expected to increase exponentially in the future.

The traditional cloud system perform all the computation, storage and processing within centralized data centers (DCs), leads to enormous amount of energy consumption leading to large CO_2 emission, massive data traffic generated form end IoT device layer causing network bottleneck, in turn, high service latency, security issue [8] and poor Quality of Service (QoS).

Therefore, there is a need for a new technology that can achieve fast response times for executing various jobs, even when the connected nodes/smart gadgets (vehicles, mobile sensors, moving objects, etc.) are highly mobile. In such an environment, fog computing [9] can provide support to real-time delay-sensitive applications to end users. The fog computing seems to be a promising technology to address the challenges we have identified earlier. Its significant advantages over existing technologies such as cloud computing also the response time, cost per service, infrastructure investment, and latency have gradually reduced with increasing technological advancements.

1.3 Evolution of Fog Computing

To enable seamless, real-time data processing, the concept of edge/fog computing was proposed, as highly virtualized platform that act as a center layer in between underlying IoT layer and cloud computing layer, which was proposed by Cisco in 2014 [10]. It aims to make ease of high quality data transfer to distributed end users connected in IoT network. It provides cloud services or resources at fog network edge. This concept extends the notion of cloud computing to the customers premises, wherein the computing, networking, and storage requirements would be meet at the edge of the network. It also deals with the processing of the workload on the fog devices (placed at the edge of the network) instead of relying on the core of the cloud platform to perform the required processing. Due to this reason, edge computing has emerged as a promising solution for enabling the seamless processing of data in the proximity of the user in real time. Edge computing is also often referred to as cloud close to the ground. As cloud providers provide various services in centralized manner in the same way the fog providers provide same services such as storage, computation and network in the distributed manner. Fog computing paradigm aims to provide real time computation, low latency services, improved quality of service (QoS), energy saving [11], and location awareness to end user applications. The user get

advantage of taking decision of what task to be moved on cloud and what task to move on Edge network. It gives real time optimal solution for workflow and task scheduling, with that to make performance better by exclusive mapping of task and vm over the fog nano data center (nDC). The fog nDC are same as cloud data center but the difference lies in lesser computational and storage capacity. However the smart gateways [12] present at fog edge helps in intelligent decision making in advance, before sending request to core fog-layer. These gateways are the intelligent gateways that are having full control over data, which takes the decision of which data to be passed and how much to sent, the time frame to send data over nano data centers. It performs little processing before passing data to the core fog layer for efficient resource utilization. Distinct from the other computing environment the fog environment has many unique features that makes it more efficient for workflow and task scheduling and improves the scheduling flexibility.

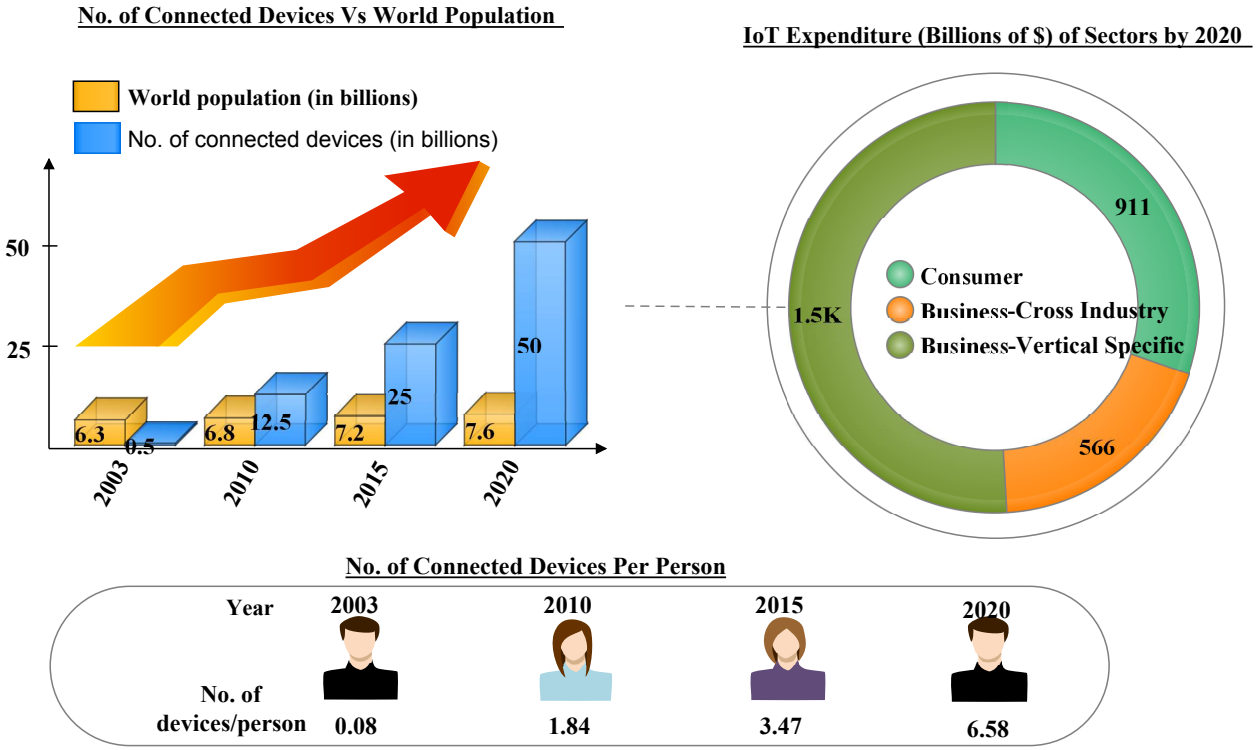

Figure 1.2: Edge computing revolution: a) number of connected devices vs. the world population

1.3.1 Architecture of Fog Computing

Fog computing follow distributed computing paradigm and follow multi-layer architecture. The detailed architecture of fog system is shows in fig. 1.3 showing concept of real time expected workflow and task scheduling in fog-cloud environment [13]. We analyze how

fog and cloud computing complement each other for some application and how the idea evolve, role and interrelation of cloud computing in edge computing. The hierarchy network consist of three layers.

1. **Front-end layer:** The bottom-most layer of fog architecture is front end layer. The front-end layer that is foglet layer consists of smart devices in user premises that passes the workflow and task scheduling request to the fog layer.

2. **Fog computing [14] layer:** The middle-most layer of fog architecture is fog computing layer. The fog layer consist of distributed fog nodes that pre-process and compute user request with additional intelligence called smart gateways and providing the priorities to workflow and task request. It schedule multi tasks and workflows in such a way that core layer faces least network traffic and allowing smart end users to experience with better quality of service and performance without deadline constraints.

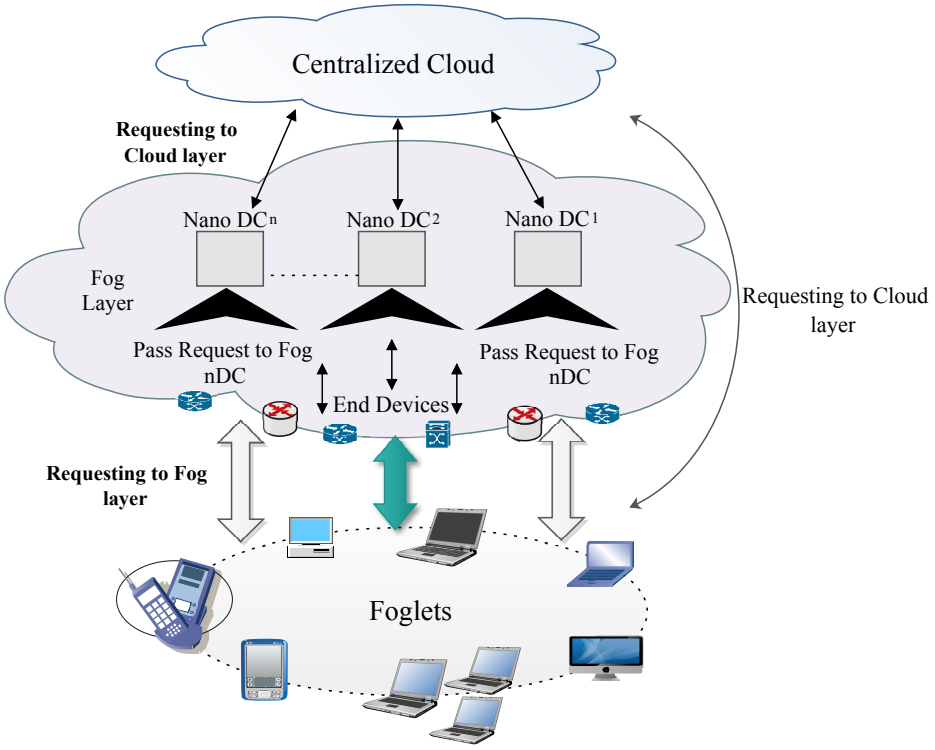

Figure 1.3: Overview of fog architecture

3. **Cloud computing [5] layer:** The uppermost layer of fog architecture is cloud computing layer. If fog nano data centers is not efficient to schedule the multiple task and workflow request then the request is passed to third core cloud layer that contains

trusted pool of resources available for use over the internet in pay how much you use manner. It aims to deliver resources on demand.

1.3.2 Essential features of Fog Computing

The important and beneficial features of fog computing [15] are listed below:

1. **Faster and Cheaper to use:** The fog nodes are much faster and cheaper to use by having additional intelligence, rather then waiting for longer response it gives the smart user to use open Software Development Kits (SDKs) and open Application Programming Interfaces (APIs).

2. **Trusted Platform:** The fog computing layer gives the trusted platform that supports security, networking and management lifecycle for many complex workflows and tasks and allowing scientist and other users to just focusing over worflow and task executions rather then interfering again and again.

3. **Resources Availability:** The fog computing provide continuum resources to the scientist and users as the resources are distributed over the endpoints and user get full advantage of the availability of resources and also increase the system efficiency and performance.

4. **Low Latency [16] and Real Time Processing:** As for scientific applications and set of tasks over fog computing layer, the major need is low latency and real time processing. The fog nodes have feature to significantly support time-critical applications which demand for latencies less then tens of milliseconds.

5. **Computing at Edge Location:** As some fog users desires for real time task and workflow execution, it facilitates services at edge of network and support location awareness to users, providing real time streaming to user applications.

6. **Faster Response Time:** The Fog computing nodes are available in distributed manner. Due to resource availability at network edge, the response time of fog nodes are much faster [17] then cloud nodes.

7. **On Demand Service:** Just like cloud computing the fog computing also facilitates large resource pool, use according to needs and charged accordingly how much you use.

8. **Enhance Quality of Service (QoS) to End Users** As compare to cloud environment, fog system provides better QoS by decreasing service latency and faster response time, providing better access to resources to fog end user.

9. **Avoid traffic to Enhance Efficiency:** Fog computing helps in reducing network traffic between fog end users and the cloud environment, playing middle intruder to reduce network traffic and servicing to user requirement efficiently.

1.3.3 Edge Computing vs. Cloud Computing [1]

Table 1.1: Edge Computing vs. Cloud Computing

Parameter	**Cloud Computing**	**Fog Computing**
Approach	Conventional cloud computing based on a centralized approach.	Fog computing is based on the concept of localization to enable fast processing and reduce latency.
Degree of security	Data are placed at remote single server in centralized manner, it can be target by hackers. So there is security gap.	In Fog environment we can replace data over distributed fog nodes and it become difficult for hackers to target number of servers.
Response Time	Due to user application request processed at central data centers, it takes long response time to send back the executable output.	As user get advantage of location based processing at the edge network, it takes lesser response time to send back the executable output.
User	The cloud computing target to provide service to general internet user.	The fog computing target to give real time service to mobile users.
Hardware	The cloud providers have large compute power and ample of storage space	The fog providers have a limited computation and lesser storage as compared to cloud providers.
User Reachability	The cloud resources are available at centralized manner, user can access it through IP networks.	The fog resources can be accessed through single-hop unwired connection.
Examples	Gmail, google talk, Google Docs, Microsoft 365, Social networking.	Shopping Center, inter-state Bus, restaurants, streets.

1.3.4 Example of Fog Computing

1. **Smart Transportation:** As fog nodes are distributed at different spaces. It give advantage to owners of high-speed vehicles by providing real time data streaming, that may be interested in knowing the best route (e.g., the one with the least traffic) to their destination.

2. **Smart Cities [18]:** The traffic light of the cities adjust various route light and patterns running on fog edge devices. Automatically adjusting light pattern using real time data streaming from fog nodes and making real time decision for controlling network traffic.

3. **Shopping Center:** We can deploy fog server at each of the floor of shopping mall that pre-collects the information about adds, sales and route navigation accessed through Wi-Fi by each user present on different floors of malls.

4. **Smart Building Control:** To control temperature of building, we put smart fog nodes at different floors of the building. Fog nodes catch the information and exchange with each other interconnected fog nodes and calculate total temperature of building. Sensors connected with them take a real time decision on collective information received and lower down the building temperature by turning on the air conditioners.

5. **Health Care Center:** Fog computing has a feature to give real services to improve health care services [19]. It helps to monitor real-time human health using health monitoring systems and provide time-sensitive medical applications. It has a ability to receive bio-signals from sensor nodes and sending data back through gateways.

1.4 Research issues of Fog Computing

1. **Energy Management [20]:** Since fog nodes are distributed over different areas, connected through each other with wireless networks. These fog nodes are equipped with limited range of computation, storage and processing power. Hence fog nodes have to process large volume of information generated in real time. Fog paradigm target to reduce both network and computing energy consumption.

2. **Maximum Resource Utilization:** It ensures maximum resources can be assigned to tasks and workflows to reduce ideal time of Virtual Machines (VMs). For that a new scheme is investigated to reduce ideal time and maximize VM computation [21].

3. **Reduce Scheduling Overhead:** The mobile users are sharing resources from anywhere, there is also a possibility of dynamic resource demand. Designing a new scheduling algorithm to distribute task and resources efficiently. Scheduling over fog nodes aim to reduce execution overhead and improving computational granularity of the system.

4. **Task Scheduling:** Task scheduling [22] scheme is needed to investigated to reduce overall operational cost, and overall system efficiency at fog platform. Therefore a task scheduling algorithm is investigated to minimize the computation time and improve system service. For that various schemes are available like VM clustering etc.

5. **Workflow Scheduling:** The workflow can be described as multifaceted computational tasks having high-level specification, depending upon each other inputs and outputs in order to fulfill a common goal. Workflow scheduling [23] is done with distributed fog workflow scheduling engines. Leading to large data transmission and require a real time service especially in the peak hours. So providing continuum resources for workflow scheduling is a big issue.

6. **VM Clustering:** Fog computing gives a way to run several virtual machines over fog nano data centers in a distributed fashion and allow users to perform processing closer to data source. The VM clustering over fog nano data center is a beneficial technique that allow to service large number of applications, run simultaneously at same time, achieve high productivity and higher profit level for the fog providers. For getting these benefits over fog layer the proper clustering strategy [24] must be used and investigated to avoid large VM migration, overload over fog nano data centers and large energy consumption etc.

1.5 Organization of Thesis:

The thesis here onwards is organized as follow way:

In Chapter 2, summarize existing research solutions on workflow and task scheduling, issues faced in task and workflow scheduling over cloud environment have been present. This literature survey is done using current and previous research papers.

In Chapter 3, description of research issues and challenges faced during task and workflow scheduling and problem formulation is illustrated.

In Chapter 4, overview of new proposed scheme over a fog environment is described. The layered architecture is also shown with diagrammatic representation.

In Chapter 5, summarize of proposed scheme performance and result evaluation of our proposed results. CloudSim and WorkflowSim tool is used to create the simulation environment.

Finally, In Chapter 6, the conclusion provides the overall contribution of the research work. The major issues for task and workflow scheduling that are solved during the course of M.Eng. work have been summarized in this chapter.

Chapter 2

Literature Survey

This chapter discusses the concept of virtualization, its benefits, problem of task and workflow scheduling, existing work done on task and workflow scheduling, various task and workflow scheduling algorithm with scheduling factors, parameters and findings and different scientific workflow structure.

2.1 Virtualization

Virtualization is the key technology in emergence of cloud and fog computing platform, aiming to provide resources to user with same hardware in different environment. The main idea of virtualization is to offer services that are not real or actual, creating virtual platform for resources like virtual storage, virtual servers, virtual desktop and virtual operating system. Virtualization allow distributed end users to share same resource instance or same application, example hard drive partitioning into two or more drives is known as virtualization because you are offering single hard drive into number of partitions for user ease. Virtualization provide ease of service both to service providers and service consumers. The virtualization give benefits to user by giving needed resources to user and charge accordingly in pay how much used manner. The virtualization in emerging technology play very significant role by improving efficiency, provide flexibility and improve utility of resources in various environment. It separates underlying hardware with guest operating system.

2.1.1 Virtualization in Fog Computing

Innovation of virtualization [1] has revolutionized the world with greater advantages by giving flexibility in every service domain. The benefits of these greatest technology widely acknowledged in fog computing. Virtualization in fog computing benefits by providing

following advantages shown in below table:

Table 2.1: Virtualization in IT organization

Before Virtualization	**After Virtualization**
1. Every machine has its own single operating system	1. Every machine has independence to crate any number of guest operating system
2. Software and hardware without virtualiztion are tightly coupled	2. We can add any number of virtual software over underlying hardware
3. Running number of applications over a single machine can cause a conflict	3. Independence is there in term of application usage
4. Non flexible infrastructure	4. Full flexibility is there
5. Poor IT resource utilization	5. Full IT resource utilization, able to consolidate various resources into virtual environment

2.2 Problem of Task Scheduling

Fog computing is a new upcoming paradigm providing flexible access to edge resources. Task scheduling in cloud environment is a major problem that reduces the performance of system. Task scheduling is process of allocating or mapping appropriate VMs to the user requested jobs. Task scheduling is always comes as a NP hard problem. There is need for efficient task scheduling algorithm that minimize the task scheduling issues. The existing algorithm mainly focused on execution time, execution cost and memory utilization. In some scenerious, there exists many task scheduling algorithm but failed to provide trade off between service availability with lesser service level agreement (SLA) violation and minimum energy consumption. Many computing parameters of the system like memory, bandwidth and processing power that directly effects the efficiency of system . The huge amount of data based upon task scheduling create challenge with respect to storage and analytics given the resource constraints for task scheduling jobs. The major goal of task scheduling in fog environment [25] is on energy consumption reduction while meeting user requirements. Mostly current task scheduling algorithms concern over overall task set completion time then single task completion time. Fog computing is a current research topic and lot of scope is available for system performance enhancement using new proposed scheme for effective task scheduling. In some issues there might be no resource availability, then there can be task migration problems or put task on wait list. So task scheduling itself is a big issue that need to solve by proposing new task scheduling scheme.

2.3 Existing Work on Task Scheduling

A lot of strategy and algorithm has been proposed, in this section we discuss some already proposed work for task scheduling.

2.3.1 Resource aware scheduling algorithm (RASA)

Saeed Parsa *et al.* [26] have formulated a Resource Aware Scheduling algorithm that is a combination of two other algorithm that is Min-min and Max-min. This algorithm calculates the expected completion time over available resources. This algorithm apply Min-min and Max-min algorithm in alternative way, like if first task is scheduled by Min-min algorithm then next task is assigned by Max-min algorithm and so on. This algorithm covers disadvantage of two traditional algorithm that is Min-min, Max-min. This algorithm does not focus on cost of executing task, deadline of task, cost of communication and load balancing. RASA performs better for large scale distributed system. This algorithm best fit for grid environment and follows batch mode scheduling method. The task that it execute is group based.

2.3.2 A particle swarm optimization-based heuristic for scheduling (PSO)

LinlinWu *et al.* [27] have formulated ths PSO algorithm. This algorithm is used to schedule application in cloud resource model and considering computation cost and data transmission cost. In this approach various scheduling applications is taken with different computation and communication cost. We compare the proposed PSO algorithm with Best Resource Selection (BSO) algorithm. In results it is shown that PSO will save 3 times cost as compared to BSO.

2.3.3 An optimistic differentiated job scheduling system for cloud computing

Shalmali *et al.* [28] have formulated an optimistic differentiated job scheduling system for cloud computing. This algorithm is proposed to schedule multiple resource request. To schedule multiple request there is non-preemptive priority algorithm. This algorithm aims to provide faster service to schedule large request. This algorithm also aims to provide QoS, and satisfy user goal. In this research one web based application is created like one for file uploading and other for file downloading. This algorithm achieve to give QoS to user and maximum profit to cloud providers.

2.3.4 Optimized-resource scheduling algorithm

Zhong *et al.* [29] have proposed this algorithm to solve scheduling problems in cloud computing and to get optimization in scheduling resources. This algorithm provides flexibility in allocating the cloud resources on request and make sure that resources are utilized to its fullest. The algorithm is based upon Improved Genetic Algorithm (IGA). It aims to increase utilization rate and speed. It is designed to make utilization better then the existing schemes. This algorithm solves task allocation problem. It works for multiple instances in cloud environment.

2.3.5 Autonomous synchronization-aware VM scheduling algorithm

Song Wu *et al.* [30] have proposed this algorithm. In VM scheduling performance degradation is a key issue. To solve this issue this algorithm is proposed. It schedules the VM by synchronizing VMs with that it is aware of VM monitoring status. This algorithm aims to make system perform efficiently means prevent performance degradation of parallel applications. It reduces allocation overhead, maximize resource utilization and perform balance scheduling and hybrid scheduling. This algorithm work best for group of tasks and work in a batch mode in a cloud environment.

2.3.6 QoS-aware algorithm for scientific workflow scheduling

Khadija *et al.* [31] have proposed QoS-aware algorithm for scientific workflow scheduling. Cloud computing allows to deal with complex applications like scientific workflow which contains set of task and require large manipulation and computation operations. Cloud computing gives dynamic provision of resources for workflow execution. But it contain various issues like allocation overhead issue, data placement constraints and high data transmission time. So this algorithm solve the critical challenge of workflow scheduling to optimize QoS. It consider various parameters like data transmission time, execution time, cost, resource availability and data placement. So this algorithm extends Parallel Cat Swarm Optimization (PCSO) algorithm to implement QoS aware algorithm.

2.3.7 Cost effective genetic algorithm for workflow scheduling in cloud under deadline constrain

Jasraj Meena *et al.* [32] propose Cost effective genetic algorithm for workflow scheduling in cloud Under deadline constrain. The cloud environment face various difficulties and one of them is in workflow scheduling in cloud environment. The existing research technique focus on minimization of cost, minimum finish time and QoS. However they does

not consider essential characteristics like VM performance and acquisition delay. To solve such issues this algorithm is proposed, that is cost effective genetic algorithm for workflow scheduling in cloud Under deadline constrain. A novel scheme is developed to encoding, crossover, population initialization and mutation operation. This algorithm also consider essential characteristics like VM performance and acquisition delay. It also evaluate performance of well known workflows like montage, LIGO and cybershake with different size.

2.3.8 A priority based job scheduling algorithm in cloud computing

Shamsollah Ghanbari *et al.* [33] propose this algorithm uses Analytical Hierarchy Process (AHP) theory. The effective scheduling approach is one of most challenging issue in cloud environment. Many researchers propose many algorithm in job scheduling but they leave some aspects uncover like job priority. To solve that issue there is a priority based job scheduling algorithm in cloud computing. The important issue in job scheduling is priority of jobs because some jobs have a longer execution time and some jobs have shorter. So shorter jobs cannot wait for long intervel of time. In this situation priority of job must be considered. There are very few algorithms that pay attention to multi-faced attributes. The proposed algorithm uses Analytical Hierarchy Process (AHP) theory and give best method on priority based problems.

2.3.9 Dynamic resource allocation algorithm

Jiayin Lia *et al.* [34] propose this dynamic resource allocation algorithm. The cloud users are distributed users and ask for the service simultaneously, also parallel processing makes system performance better. So proper mechanism is needed for allocation of resources to cloud platform making utilization of resources better. In this paper a dynamic resource allocation algorithm is proposed with preemptable tasks. This scheme make dynamic adjustment of resources based upon actual task execution information. This algorithm also proved in making system performance better.

2.3.10 A compromised time-cost scheduling algorithm

Ke Liu *et al.* [35] propose this algorithm, this algorithm deals with cost minimization under user defined deadline. So this algorithm mainly focus over cost and time. It provides just-in time graph during task execution related to time and cost. This algorithm work by checking uncompleted tasks and scheduling them in first round. Then it adds sub deadline for scheduling. After that estimating schedule time and cost in advance and make proper ser-

vice available for allocation. It provides just-in time cost time graph and after completing first round sleep until next round tasks arrived.

Task Scheduling Algorithms					
Algorithm	**Scheduling Factor**	**Findings**	**Scheduling Parameters**	**Scheduling Method**	**Environment**
Resource-aware scheduling algorithm [26] (RASA)	Group of tasks	It combines two traditional algorithm called as Max-min and Min-min. That is used for reduction of makespan	MakeSpan	Batch Mode	Grid Environment
A particle swarm optimization-based heuristic for scheduling [27] (PSO)	Group of task	1. Measure both data transmission cost and computation cost 2. Enable good resource distribution	Utilization Cost and Time	Dependency Mode	Cloud Environment
An Optimistic differentiated job scheduling system for cloud computing [28]	Single task	1. Efficient for giving QoS to user 2. Maximum profit to service providers	Maximum Profit and Quality of service	Dependency Mode	Cloud Environment
Optimized-resource scheduling algorithm [29]	Task allocation problem	1. Utilization of resource is high 2. Make use of Improved genetic algorithm for increasing resource utilization	Resource utilization and speed	Multiple Instance	Cloud Environment
Autonomous synchronization-aware VM scheduling algorithm [30]	Group of task	It is used to eliminate degradation of performance that runs parallel applications	Performance, balance Scheduling, hybrid scheduling	Batch Mode	Cloud Environment
QoS-aware [31] algorithm for scientific workflow scheduling	Group of tasks or workflow	1. Focus on QoS aware workflow scheduling 2. Focus on various parameters like data transmission time, execution time, cost, resource availability and data placement	data transmission time, execution time, cost, resource availability, data placement and QoS	Dependency Mode	Cloud Environment

Task Scheduling Algorithms					
Algorithm	**Scheduling Factor**	**Findings**	**Scheduling Parameters**	**Scheduling Method**	**Environment**
Cost effective genetic algorithm [32] for workflow scheduling in cloud under deadline constrain	Multiple workflow	A novel scheme is developed to encoding, crossover, population initialization and mutation operation	VM performance and acquisition delay	Dependency Mode	Cloud Environment
A Priority based job scheduling algorithm [33] in cloud computing	Array of jobs	1. Set priority according to job resource needed 2. Provide lesser finish time	Array of Job queue	Dependency Mode	Cloud Environment
Dynamic resource allocation algorithms [34]	Group of task	1. Preemptable task execution increase resource utilization 2. Based on update information of task it improves system performance	Response time, energy consumption, load balancing	Dependency Mode	Cloud Environment
A compromised time-cost scheduling algorithm [35]	Array of workflow instance	Reduce cost and time of task execution in cloud environment	Cost and Time	Batch Mode	Cloud Environment

2.4 Problem of Workflow Scheduling

The workflow can be described as multifaceted computational tasks having high-level specification, depending upon each other inputs and outputs in order to fulfill a common goal. It is represented as W$< T, E >$ is a directed acyclic graph where E is represented as set of edges among the interdependent task and T is represented as a set of task T=$\{T_1, T_2, T_3, ..., T_n\}$ of the workflow.

Today scheduling plays a very significant role for real time workflow execution for satisfying the smart end application requirements while effectively utilize the resources. Workflow scheduling is basically how we divide the task into subtasks and how we map the

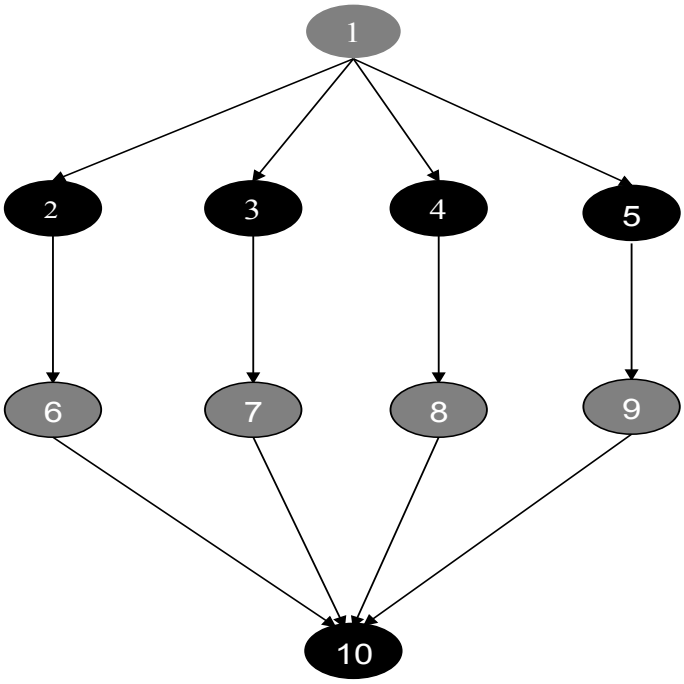

Figure 2.1: Workflow represented in graph form

sub-tasks over machine. However, scientists are facing many workflow scheduling issues over a cloud infrastructure like whenever they require a real time service especially in the peak hours it takes long computational time over centralized data center, managing multiple data centers by cloud provider in order to work cooperatively also demands continuous monitoring.

The scientist face problem of large transferring time from end premises to cloud. In order to reduce these workflow scheduling issues over cloud environment, many heuristics have been proposed like minimizing execution time, budget constrained, minimum completion time, min-min, max-min, suffrage and auction based optimization but all such algorithms just focus on few parameters unless other are ignored. We are focusing on uncover issues and major needs required by smart users.

2.5 Different Scientific Workflow Structure

2.5.1 Montage workflow

This scientific workflow [35,36] is created by NASA/IPAC Infrared Science, is a astronomy application. It is used for constructing custom mosaics of the sky by generating images.

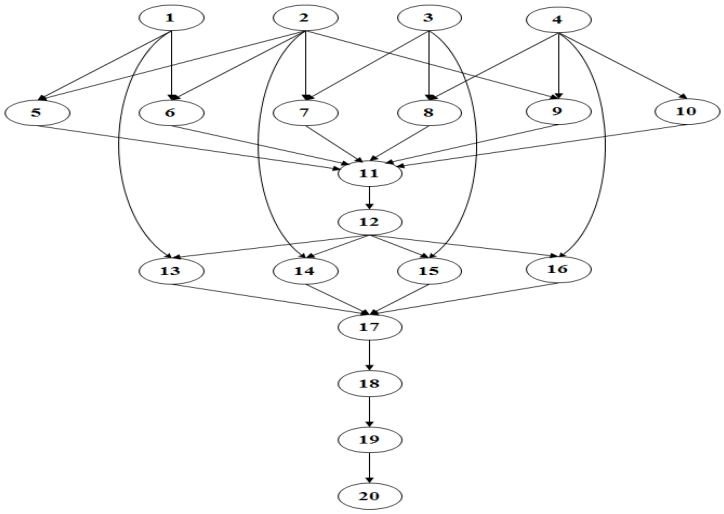

Figure 2.2: Montage workflow structure

2.5.2 Epigenomics workflow

This workflow [35,36] is used in biology. The USC Epigenome Center is presently involved to map epigenetic state of human cells on genome-wide scale. Tasks of Epigenomics are highly CPU intensive, less I/O intensive.

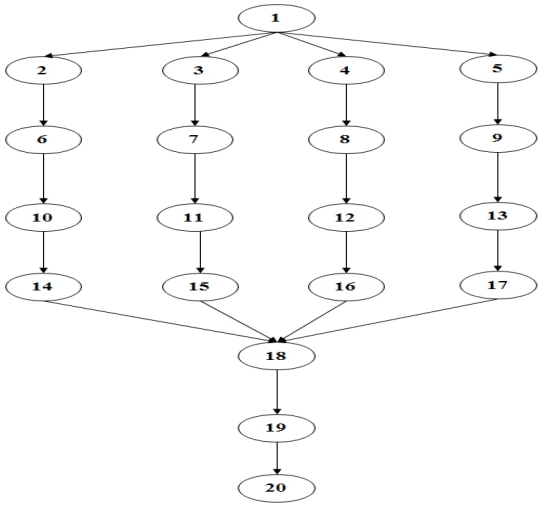

Figure 2.3: Epigenomics workflow structure

2.5.3 SIPHT workflow

The SIPHT workflow is used for search to small untranslated RNAs (sRNAs). It is useful in regulating various processes like secretion or virulence in bacterias.

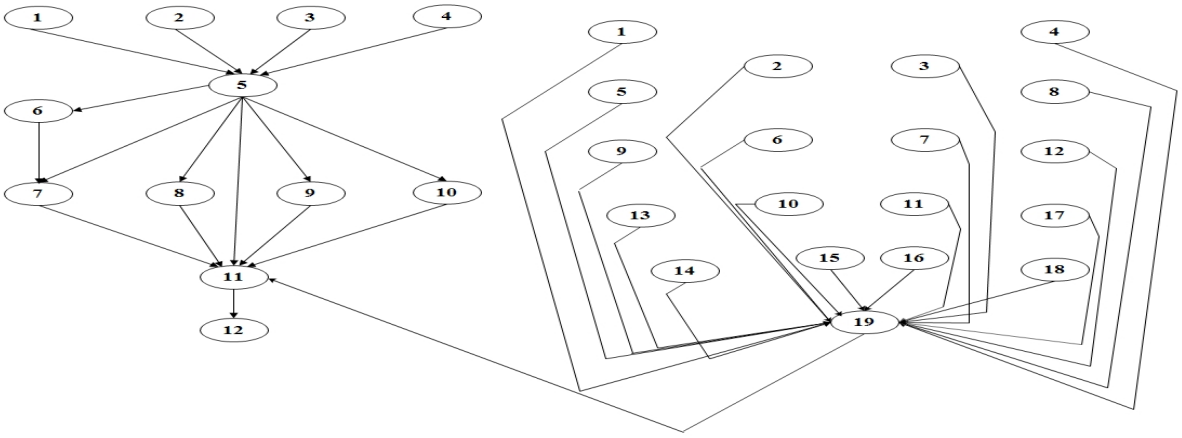

Figure 2.4: SIPHT workflow structure

2.5.4 LIGO workflow

Laser Interferometer Gravitational-wave Observator workflow [35] are useful for gravitational wave identification produced by various universe events.

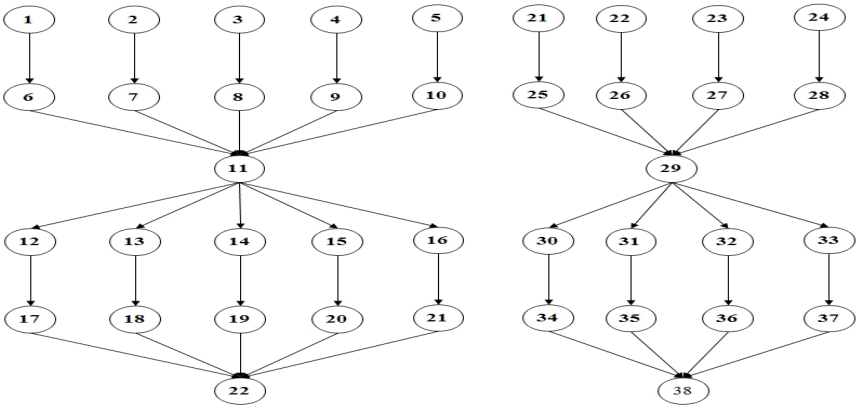

Figure 2.5: LIGO workflow structure

2.5.5 CyberShake workflow

Utilization of this workflow [35, 36] is done in Southern California Earthquake Center (SCEC). By using Probabilistic Seismic Hazard Analysis (PSHA) technique, it illustrate earthquake hazard. The task of CyberShake Workflow is mainly I/O intensive.

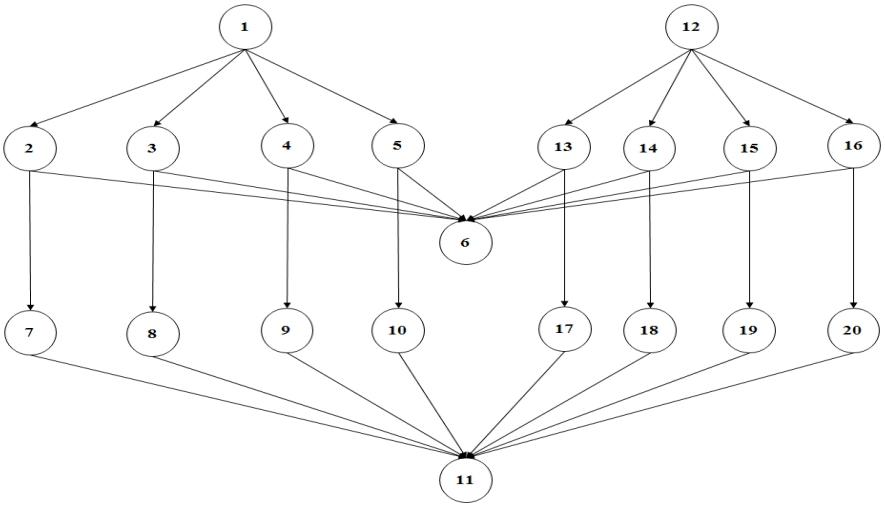

Figure 2.6: CyberShake workflow structure

2.6 Existing Work on Scientific Workflow Scheduling

Various authors present their work in area of workflow scheduling. These algorithms are applied over wide range of applications and problems. Some of workflow scheduling algorithms are discussed below.

2.6.1 Deadline and cost based workflow scheduling

Nitish Chopra *et al.* [37] have proposed this Deadline and cost based workflow scheduling algorithm. The proposed alogirthm perform cost optimization decides which resources should be used on lease for completetion of workflow applications. This algorithm based on level wise scheduling and execute level wise task. It uses sub-deadline concept helping in finding best resource also for cost saving and complete its execution within deadline. This algorithm is also compared with min-min approach.

2.6.2 Scheduling service workflows for cost optimization in hybrid cloud

Luiz *et al.* [38] propose this algorithm. In the cloud environment it is important concern that when and how services to be served to meet the deadline and get minimum execution time with minimum cost. So the proposed scheduling algorithm aims that which type of services to be executed in cloud environment in order to minimize monetary cost and time. This algorithm also shows through experiment that it decreases execution cost with maintaining executing time of services.

2.6.3 Upgrade fit heuristic algorithm

Long Wang *et al.* [39] proposed this algorithm. This algorithm extends existing workflow management capabilities from DAG to non-DAG processing. Upgrade algorithm determines number of VM and suitable type of VM for running iterative workflow tasks, and to compromise between execution speedup and budget expenditure.

2.6.4 Energy saving algorithms for workflow scheduling

Elaine N. Watanabe *et al.* [40] proposed this algorithm, present new task scheduling approach aiming to reduce energy utilization. Comparing with other scheduled algorithm, it shows 22.74% of energy saving and efficient scheduling is obtained and shows interconnection of tasks in the workflow.

2.6.5 Task duplication-based workflow scheduling algorithm

Indrajeet Gupta *et al.* [41] have proposed this algorithm. It proposes workflow scheduling algorithm based on task duplication for distributed cloud environment. This algorithm contain two face, in first phase it computes priority for all tasks and in second phase perform scheduling by calculating task arrival time from one task data to other task data. This algorithm aims to maintain minimum execution time and maximum service utilization. The performance of this algorithm is done through diffferent scientific workflow application with different size.

2.6.6 Enhanced weighted dynamic voltage frequency scheduling algorithm (DVFS)

Prathibha *et al.* [42] propose this algorithm that aims to minimize energy consumption at both server side and communication devices. This algorithm assign task to virtual machines aiming of minimizing energy consumption by servers. It also check data centers with renewable energy powered for job scheduling. In energy saving one important component

is networking devices like switches, routers which is part of communication system. For performance evaluation independent task and scientific workflow are considered.

2.6.7 Efficient workflow scheduling algorithm (EWSA)

Mainak Adhikari *et al.* [36] propose this algorithm. Executing large workflows within deadline in cloud computing is a big challenge. Many algorithm pre-compute scheduling task execution time, based on resource availability. On other hand handling, large number of resource utilization is a challenging task. Therefore, this proposed algorithm called Efficient workflow scheduling algorithm (EWSA) handles large application request simultaneously. This algorithm works on calculating execution time of tasks, aims to minimum deadline violation.

2.6.8 Static workflow scheduling algorithm

Arash Deldari *et al.* [43] have proposed this algorithm, that selects available resources over cloud and aims to minimize leasing cost of resources and user defined deadline should not be violated. This algorithm works by dividing workflow into number of clusters. Its simulation setup shows lesser cost in workflow execution and also meets user defined deadline.

Workflow Scheduling Algorithms			
Algorithm	**Scheduling Factor**	**Explanation**	**Environment**
Deadline and cost based workflow scheduling [37]	Dependable task	1. Perform cost optimization 2. Help in cost saving and complete its execution within deadline	Cloud Environment
Scheduling Service Workflows for Cost Optimization in Hybrid Cloud [38]	Multiple task	Decreases execution cost with maintaining executing time of services.	Cloud Environment
Upgrade Fit Heuristic Algorithm [39]	Group of task	1. This algorithm extends existing workflow management capabilities from DAG to non-DAG processing 2. Present new task scheduling approach aiming to reduce energy utilization	Cloud Environment

Workflow Scheduling Algorithms			
Algorithm	**Scheduling Factor**	**Explanation**	**Environment**
Energy saving algorithms for workflow scheduling [40]	Group of task	This algorithm present new task scheduling approach aiming to reduce energy utilization and maximum energy saving	Cloud Environment
Task duplication-based workflow scheduling algorithm [41]	Group of task	This algorithm aims to maintain minimum execution time and maximum service utilization	Cloud Environment
Enhanced weighted dynamic voltage frequency scheduling algorithm (DVFS) [42]	Group of task or independent task	1. It aims to minimize energy consumption at communication devices 2. It make minimizing energy consumption by servers.	Cloud Environment
Efficient workflow scheduling algorithm (EWSA) [36]	Multiple workflow	1. It is used for dynamic execution of workflow 2. Minimize execution time and cost	Cloud Environment
Static workflow scheduling algorithm [43]	Task group	1. This algorithm works by dividing workflow into number of clusters 2. It aims to minimize leasing cost of resources	Cloud Environment

Chapter 3

Research Motivation and Problem Statement

The observation from literature survey is that many researchers proposed their algorithms and models, but still number of research gaps need some further investigation. Some of them are as discussed below.

3.1 Research Issues or Challenges in Task Scheduling Algorithm

Task scheduling in cloud environment is a challenging task, some of the reasons are explained below.

1. The service resource pool is centralized repository so it is time consuming to predict which resources are available for execution of large task request.

2. To process the large volume of information generated from end user, the cloud-based infrastructure may lead to higher bandwidth consumption and high energy consumption.

3. The core computational units used in cloud computing [28] (i.e., data centers) are not as distributed and are often found at locations far from the user, resulting in relatively slower response time.

4. To support real-time data efficiently, there is a need to schedule the various tasks in an optimal manner in order to meet various SLA parameters such as response time, service availability, latency and cost.

5. The end user requests are moved to boundaries of cloud causing issue of data transfer and data placement.

6. The other research issue is maintaining data in cloud environment, it should not cause bottleneck issue to other services.

7. For load balancing [29] a VM migration technique can be used, but VMs are heavy weight and difficult to migrate as compared to containers (a research approach).

8. To achieve multi-objective criteria, that is difficult to imposed by certain applications. Different results are produced by different techniques, and become difficult in such situation to choose particular scheduling approach for generalized class of applications.

3.2 Research Issues and Challenges in Workflow Scheduling Algorithm

Despite from the advantages of workflow scheduling in cloud computing there are many obstacles to schedule and run scientific workflow in cloud environment some of them are listed below.

1. The huge data generated by large number of smart connected resources cause data management challenge in cloud environment. It has to deal with large in and out movement of data, large data storage and co-location of computation issues in cloud.

2. The smart end users always desire for location-based processing for real time workflows. But in cloud environment moving large data to distance CPU [38] is inefficient and expensive act.

3. Difficulty in handling dynamic workflows whose graph structure of workflows changes with time period.

4. It is difficult to have pre-estimated knowledge of resources matched with the required capacity needed for workflow application scheduling.

5. In cloud environment, because of centralized resource pool it is difficult to process real time workflow scheduling request.

6. Difficulties in scheduling overhead generated from multiple workflows because number of task of workflow are dependent over each other, so its difficult to co-relate its execution.

3.3 Problem Formulation

Theres are many research gaps in existing scheduling proposals for task and workflow in cloud computing, so there is a requirement of more formulized approach for the same. So keeping in view of the above, the problem formulation for the current proposal consists of following issues:

1. The task and set of task often require complex execution environment. In the traditional method, the execution of dynamic dependable multi-task jobs also known as workflow and large volume of information typically done by centralized cloud engines, leading to high probability of communication and computing bottleneck, large data transmission time and failure issues.

2. There is rapid increase in the number of end users accessing the cloud environment. Cloud is based on centralized data center, which have a power to serve large number of distributed users who can access information from anywhere. The day to day usage of data centers leading to large energy consumption. Thus the high energy consumption is a challenging issue, in cloud computing network.

3. Service Level Agreement (SLA) plays a vital role in guaranteeing Quality of service. The deadline in SLA is one of the most important attribute contribute in guaranteeing QoS for users. However this attribute is violated when there is real time tasks, with critical deadline-constrained. Therefore lesser number of SLA violation is a valuable research issue.

4. For the optimum utilization of computing resources, the VM migration technique is used from one server to another. But there is a issue that a large number of VM migration increases network congestion and increases overhead of tracking VM migration information.

5. The smart end users always desire for location-based processing for real time workflows. Again and again accessing the centralized cloud engine to retrieve localized data leading to an inefficient act, which motivates new virtualization technology and a promising platform called fog computing. The fog computing is used to cope up with these challenges. Its a powerful technology that supports data processing and service availability to end users at the edge of the network.

Chapter 4

The Proposed Scheme

This Chapter discusses about how the problem stated in previous research papers, the solution to the problem and the proposed algorithm.

4.1 Task Scheduling over Fog-Nano Data Center

4.1.1 Container-based edge service management system (CoESMS)

This section presents our proposed architecture and its layers. Figure shows the architecture of CoESMS consisting of three main entities: resource consumers (C), utility provider (P), and suppliers (S). It also consists of five different layers. The layer 1 is a consumer layer, layer 2, 3 and 4 are provider layer and 5 is supplier layer which we describe as follows.

Layer 1: Resource consumer layer (RCL)

Layer 2: Requirement collector layer (RCoL)

Layer 3: Controller layer (CL)

Layer 4: Broker layer (BL)

Layer 5: Computation layer (CoL)

We describe each layer below.

4.1.2 Layer 1: Resource Consumer Layer (RCL)

This layer comprises of the fog resource consumers such as IT clients and end users. These consumers initially access the smart devices through a local area network (LAN) and submit their service requests via the edge gateway. The gateway provides a channel to handle data communication between external network and the core network. Hence, the edge gateways relay the heterogeneous data and user requests to the second layer of the system for

further processing. In short, this layer is only responsible for aggregating and redirecting the user task requests to the CoESMS.

4.1.3 Layer 2: Requirement Collector Layer (RCoL)

This is the second layer of the proposed system and comprises of two modules namely, the task requirement module (TRM) and the user task request module (UTRM). Using these two modules, the tasks that would be executed at the edge are selected while rest of the tasks are forwarded to the core of the network. The main functionality of the TRM is to evaluate the task's request requirements with respect to the various parameters for further processing. These parameters include task type, task size, budget and time. The parameter "task type" can either be computation-intensive, storage-intensive or network-intensive task. Hence, the proposed fog platform provides services to the user only if the task type does not fall within this classification. Otherwise, tasks are redirected to the core of the network.

The next parameter is the "task size" and is classified into four categories. The first category represents tasks that have a low requirement for CPU time and memory, the second category denotes tasks with intermediate CPU and low memory requirement, the third category comprises of tasks with intermediate CPU and high memory requirement, and the fourth category represents tasks which require high CPU usage and high memory requirement. The other two parameters, i.e., budget and time are also considered as dynamic parameters. Their values vary in predefined ranges according to the SLA agreed between the user and the utility provider. Based on the above classification, each task request is associated with the above specified parameters. The corresponding configuration information is then stored using a heap data-structure in the form of an application form. The proposed work is done by the user requirement module (URM). Additionally, URM associates the user defined task requests with the required number of containers based on each of their configuration information. The module also assigns each user request with a unique request *id*. This identifier helps to schedule the task on the required containers and helps to track the process. After this, the generated application form is forwarded to the next layer.

4.1.4 Layer 3: Controller Layer (CL)

This layer is an important part of the proposed scheme and primarily controls the task scheduling process. We describe this layer in detail in the next section. This layer comprises of two modules: the resource monitoring module (RMM) and the container scheduling module (CSM). RM continuously keeps track of the idle containers that are available

for task scheduling. Based on the information received from this module, CSM schedules the tasks on the containers in an energy optimal manner using the proposed game theoretical framework which we discuss in the next section.

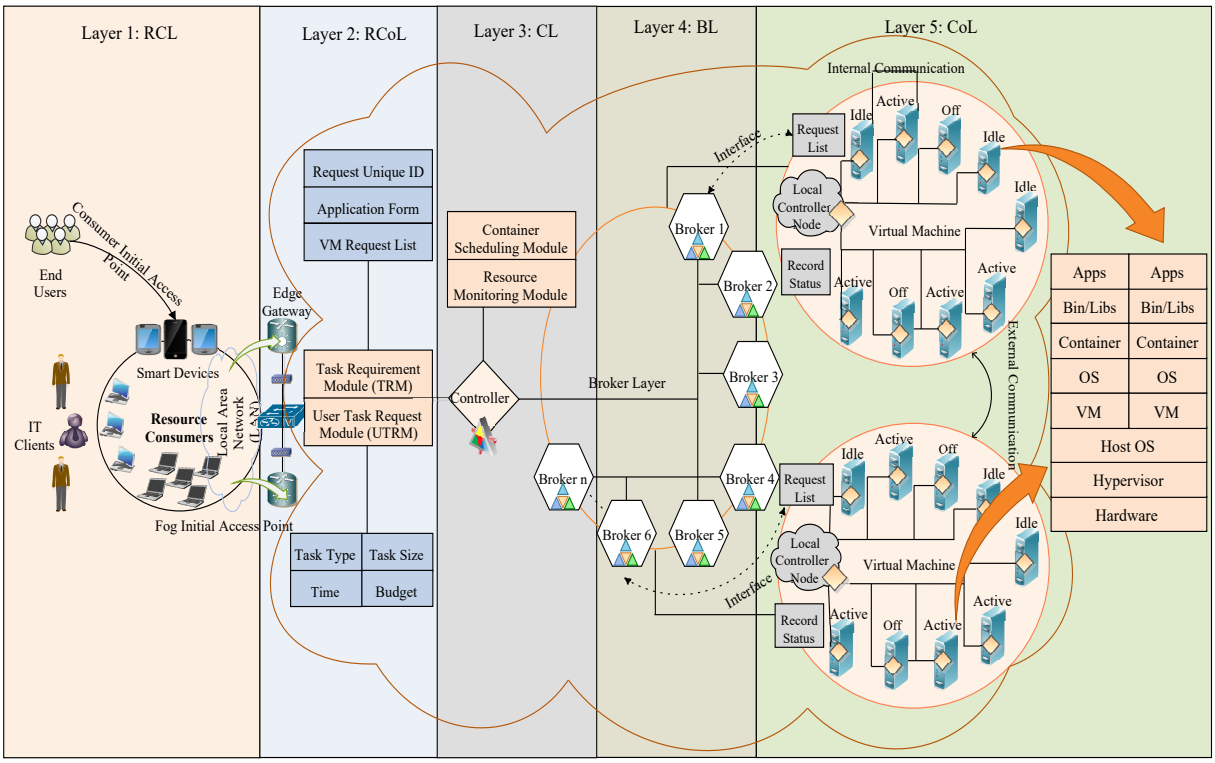

Figure 4.1: Proposed architectural diagram of the conatiner-based edge service management system (CoESMS).

4.1.5 Layer 4: Broker Layer (BL)

This layer acts as an intermediate entity between layer 3 and layer 5. The interaction between these layers is extensively used in the task scheduling process. This layer is designed to select, rank and reserve resource combinations in the form of containers. This is achieved with the help of the broker that belongs to the fog computing platform. The primary responsibility of this fog broker is to handle the user services, keep track of available VMs and containers and delegate the requests to the containers of VMs.

4.1.6 Layer 5: Computation Layer (CL)

The last layer comprises of the nDCs which actually execute the tasks using the lightweight containers [44]. It also incorporates a node-level local controller which keeps track of all containers at the nDC level and their current status. Depending upon their current status,

the controller selects the containers that can be allocated to different tasks. Additionally, it also performs container migrations within and between the VMs. These migrations are carried out for reducing the overall energy consumption of the nDCs during unexpected failures

4.2 CoaaS for Energy Efficient Task Scheduling at the Edge

CoaaS is an emerging technique that allows task scheduling on lightweight containers [44, 45] and supports their migration whenever required. It can also be viewed as an alternative solution to hypervisor-based virtualization, which is inadequate to execute time-critical tasks. The latter also lacks the capability to reallocate run-time resources. At present, four major virtualization technologies are available in the literature and these include: para, full, operating system (OS) level and native virtualization. Para-virtualization was first introduced by the Xen Project team and it is considered as a lightweight virtualization technique. It enables hardware virtualization and does not rely on the virtual extensions of the host machine. Full-virtualization provides a VM environment which fully simulates the host's underlying hardware. Native-virtualization partially simulates the hardware to run the full-fledged OS. The operating system level virtualization also known as container-based virtualization and is the most popular one among all the virtualization techniques.

In this work, we focus on container-based virtualization [45] because it helps to deploy the tasks over the containers. Moreover, it also supports lightweight migration within and among VMs. This in turn, helps in live system updates and eliminates the need for hardware emulation by using the underlying host OS. It should be noted that all the containers inside VMs share a single OS kernel. The main advantages of using containers over VMs include: lightweight, increased performance and higher efficiency. In short, containers can be executed at a higher speed than VMs and can be used for real-time data collection from the end users. Containers also share kernel instances, OS and network connections. The VMs, on the other hand, have their own virtualized network, CPU and OS. Additionally, containers enable higher degree of application execution using least number of OSs in comparison with VMs. Table 4.1 highlights the relative differences between containers and VMs.

Container scheduling and migration as proposed in this work can be best understood by using fig. 4.2 which shows that the container scheduling and migration are managed automatically by the docker (a software framework built around the container engine). Docker also contains all the components which are required for runtime task execution such as configuration files and databases. It enables an additional layer of abstraction and virtual-

Table 4.1: Comparison between containers and virtual machines (VMs)

Parameter	**Containers**	**Virtual Machines (VMs)**
Type of virtualization	Based on OS-level virtualization where the virtual layer acts as an application above the OS.	Part of completely virtualized environment which abstracts only the underlying physical hardware.
Degree of security & quality assurance	High risks involved due to direct deployment of containers over OS.	Relatively low risks are involved if the containers are deployed inside VMs.
Type of resource sharing	Containers share kernel instances, OS, bare file system and network connection.	VMs have their own virtualized network, BIOS, CPU, OS and disk storage.
Performance	Enable maximum degree of application execution using the least number of servers.	VMs can execute a higher number of applications with a variety of OS.
Type of user space	Separate user space to run each application instance.	VMs contain common user space for each application instance.
Kernel state	Require booting for running OS and application loading.	Require application loading only since the kernel is already operational.
Start and stop time	Start and stop time of docker container is less than 50 ms.	Start and stop time of VM is 30-40 and 5-10 secs respectively.
Degree of overhead	Low startup overhead for launching containers.	Relatively high overhead for launching VM.
Memory and CPU usage	Lesser CPU and memory usage per container instance.	Large memory and CPU usage per VM instance.
Migration and management	High migration speed and management as they support lightweight package.	Low migration speed and slower management because they are done on a large scale.
Cost	Cost overheads are lower because of lower operational and hardware investment.	VMs are typically heavyweight which incur high cost overheads.
Need of guest OS	No need to have a separate guest OS because each instance of application shares a common kernel and host OS.	VMs allow to share single hardware with multiple guest OS.
Tools	Various container virtualization tools are OpenVZ, docker, Sandboxie, Warden/Garden.	Various hyper-virtualization tools are Virtual PC, Virtual Box, WMWare, Xen, KVM.
Standardization	Not well standardized and involves complexity in OS and kernel specification.	Well standardized system having the same capabilities same as the bare-metal system.

ization. It is particularly used for Linux OS and leverages the resource isolation feature of the underlying kernel. This in turn permits containers to be independently executed within the kernel instance, thereby eliminating the need to initiate and maintain VMs [45].

Fig. 4.2 illustrates task scheduling using containers. In the proposed scheme, only L number of containers have been considered on two VMs respectively. The docker engine helps to schedule the tasks over the portable containers. The scheduled containers then execute the tasks. After task execution, the containers are freed. The docker also handles container-to-container migration for various scenarios such as task failure, downtime, and energy management. For instance, the simplest approach to trigger the container migration is on the basis of energy usage of the servers. Hence, the migration would be initiated whenever the utilization of the server exceeds or falls behind the predefined upper and lower threshold limits respectively. These migrations can be either within (internal) or between the VMs (external) as shown in fig. 4.2.

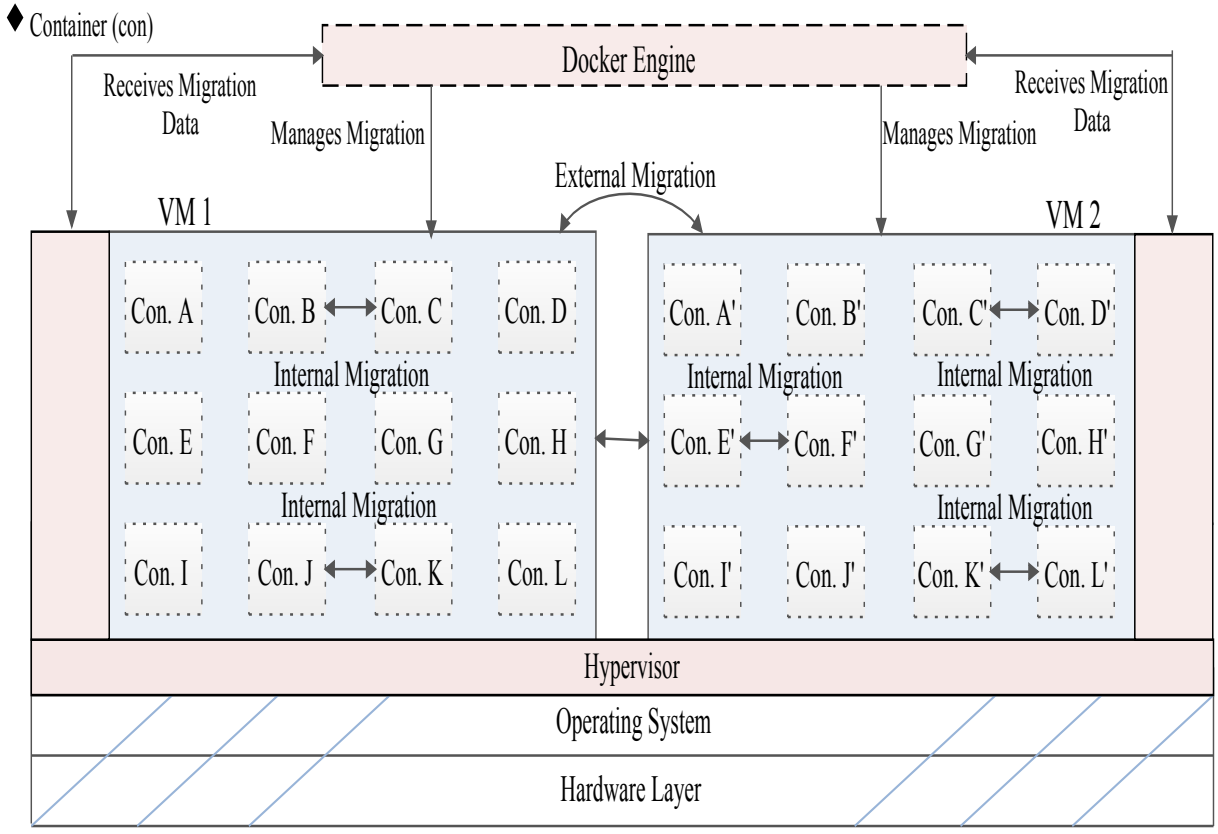

Figure 4.2: Task scheduling at container level

4.3 Operation of CoESMS: Task Selection and Task Scheduling

This section describes the operation of the proposed CoESMS. The major emphasis of CoESMS is on energy consumption reduction while meeting the user's requirements for accessing various services such as traffic information, parking information and infotainment services. It aims to provide an efficient task scheduling management system while satisfying the increasing user demand for data access and addressing the constraints of the traditional cloud computing platform.

The overall operation of CoESMS is shown fig. 4.3. The TRM module of layer 2 validates the different tasks according to their respective task types as specified above. Based on this evaluation, the tasks are either processed at the edge or at the core of the network. Moreover, layer 2 generates the configuration list of the requested tasks to be executed at the edge using the heap data structure.

This user's requests application form is sent to the controller of the Layer 3. The CSM module then validates the number of containers (η_i) required by each i^{th} task. This η_i is then compared with the available number of containers (α) as reported by RRM. If the number of containers required by the task is less than or equal to the number of available containers, then the task is scheduled for container allocation and is passed on to the next layer. Otherwise, the task is not configured for allocation at the edge and in this case the task is sent to the core for further processing.

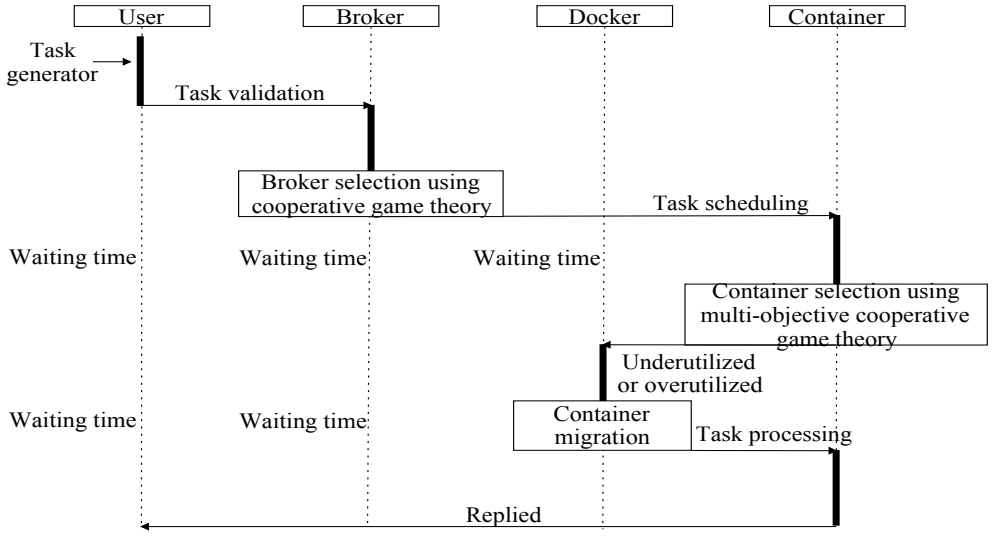

Figure 4.3: Sequence diagram for task selection and scheduling using CoESMS

In the next phase, the CSM selects the broker j amongst N available brokers as depicted

in fig. 4.1. This selection of the broker for further task processing is formulated as a cooperative game theory [46–48]. In the proposed scheme, we have considered an n-player game, where each player, i.e., broker attempts to reduce the overall communication cost based on the current bandwidth (B_j) and the load status (W_j). The utility function of brokers (μ_j) is formulated using weighted contributions of these two parameters, i.e., B_j and W_j as follows.

$$\mu_j = \frac{1}{N} \times \frac{\alpha.B_j}{\beta.W_j/W_j^{max}} \quad s.t.\ \alpha,\beta \in [0,1] \tag{4.1}$$

where, W_j^{max} denotes the maximum load capacity of the j^{th} broker and the parameters α and β refer to weights for B_j and W_j respectively. The higher the value of μ_j, the higher are the chances of selecting j^{th} broker and vice-versa. Moreover, the selected broker sends the configuration details of the task i under consideration to the local controller of the layer 5 for final task scheduling. The local controller then maps the tasks to the available containers across different VMs based on various configuration parameters (such as task type, CPU time, budget and memory). The mapping of the tasks to the appropriate containers can be viewed as a multi-objective cooperative game and is defined as follows.

Definition 1: Suppose we have a total of M task requests which need be to mapped to P containers across Q VMs available at the nDC level. The makespan of the i^{th} task request is represented as $A_i, i \in [1,M]$ and refers to its maximum completion time. The primary objective of the multi-objective cooperative game theoretical model is to schedule the set of task requests to the k^{th} container(s) such that the total energy consumption of the nDC and makespan of the tasks ($F(x)$) are minimized while satisfying the following set of constraints:

$$\begin{aligned} Minimize\ \ F(x) = f(E(x),A(x)); \\ s.t\ \ c_i(x) \leq \beta_{x,k}, \forall i,k, \\ m_i(x) \leq \gamma_{x,k}, \forall i,k, \\ b_i(x) \leq \Lambda_{x,k}, \forall i,k, \\ x \in S \end{aligned} \tag{4.2}$$

where F(x) is the objective function as per the solution x amongst the available set of feasible solutions (S). Here, the parameter S denotes the available search space which com-

prises of the feasible solution according to the set of constraints taken. $E(x)$ and $A(x)$ represent the minimization functions corresponding to nDC's energy consumption and task's execution time respectively. The constraint parameters $c_i(x)$, $m_i(x)$ and $b_i(x)$ correspond to i^{th} task's CPU, memory and budget requirements respectively. In addition, $\beta_{x,k}$, $\gamma_{x,k}$ and $\Lambda_{x,k}$ represent the limits on CPU, memory and budget with respect to resource usage function on the k^{th} container for a given task assignment respectively.

The multi-objective scheduling scheme discussed above is formulated using the cooperative game theory. However, due to the complexity of this problem, it is further formulated as a sequential cooperation game. Hence, the major parameters of this game are elaborated as follows:

1. The set P containers across the nDC act as players in the cooperation game.

2. The utility function of the competing players is computed as follows:

$$\mu_k = arg\left(min\left(E(x), A(x)\right)\right) \tag{4.3}$$

3. The set of strategies (Δ and β) adopted by the players is determined by the following set of constraints:

$$\begin{aligned} c_i(x) &\leq \beta_{x,k}, \forall i,k, \\ m_i(x) &\leq \gamma_{x,k}, \forall i,k, \\ b_i(x) &\leq \Lambda_{x,k}, \forall i,k \end{aligned} \tag{4.4}$$

Here, the strategy Δ refers to the task distribution matrix with respect to different VMs and containers. The strategy β represents the CPU allocation matrix. The objectives and the above mentioned constraints are defined accordingly. The parameters E_{act} and E_{idle} refer to the total energy consumption of nDC when it is active and idle respectively. Similarly, $P_{act}, P_{idle}, T_{act}$ and T_{idl} refer to the nDC's power consumption and related time slots.

$$\begin{aligned} E(\Delta) &= E_{act} + E_{idle} \\ &= (P_{act} - P_{idle})T_{act} \sum_{k=1}^{P} \delta_k \theta_k + P_{idle} T_{idl}; \\ A_i(\Delta, \beta) &= \delta_k / \beta_k; \end{aligned} \tag{4.5}$$

4. Each player aims to minimize the multi-objective function $F(x)$ in the multi-constraint environment.

In cooperative games, the players in a group coordinate their moves and actions while leveraging the advantages of some transferable utility. This means that the players with a greater utility function can coordinate with the players with a lower utility function to achieve optimal results. We express it as a sequential cooperation game in which the participating entities called the players initially start by selecting a predefined strategy and observe the actions of the preceding players.

4.4 The Proposed Scheme for Workflow Scheduling over Fog-Nano Data Center

4.4.1 Overview of Workflow Scheduling System (WSS)

The workflow scheduling for real time service demand is the greatest challenge for many scientist. Thats why, gaining attention of many researchers over the scheduling problem. This work section represent the proposed architecture of workflow scheduling over fog-cloud environment as shown in fig. 4.4 and showing proposed scheduling algorithm scheme in detail.

4.4.2 Architecture

The overall system architecture works in 3 major environment that is end-user environment, fog environment and cloud environment. At the starting of scheduling procedure, the initial input is provided by smart end-users. The each end user is attached to smart devices like smart computers, smart phones, this layer is also known as foglet that submit workflow scheduling request by accessing smart medium to the next environment called fog layer. The smart users can be a part of any domain like bio-informatics, weather forecasting, e-science or any other. The smart user submit their workflow request and with that also describe its requirements like deadline of workflow execution, budget and QoS requirement.

Before passing the request to core nano data centers, the data must be passed through edge gateways. These perform intelligent pre-processing over the data and calculate the preferences of the user workflow request type like the deadline, the size of CPU required, the time needed to execute and type of workflow. After intelligent processing the user submit the request to the supervise engine in first come first serve basis. The supervise

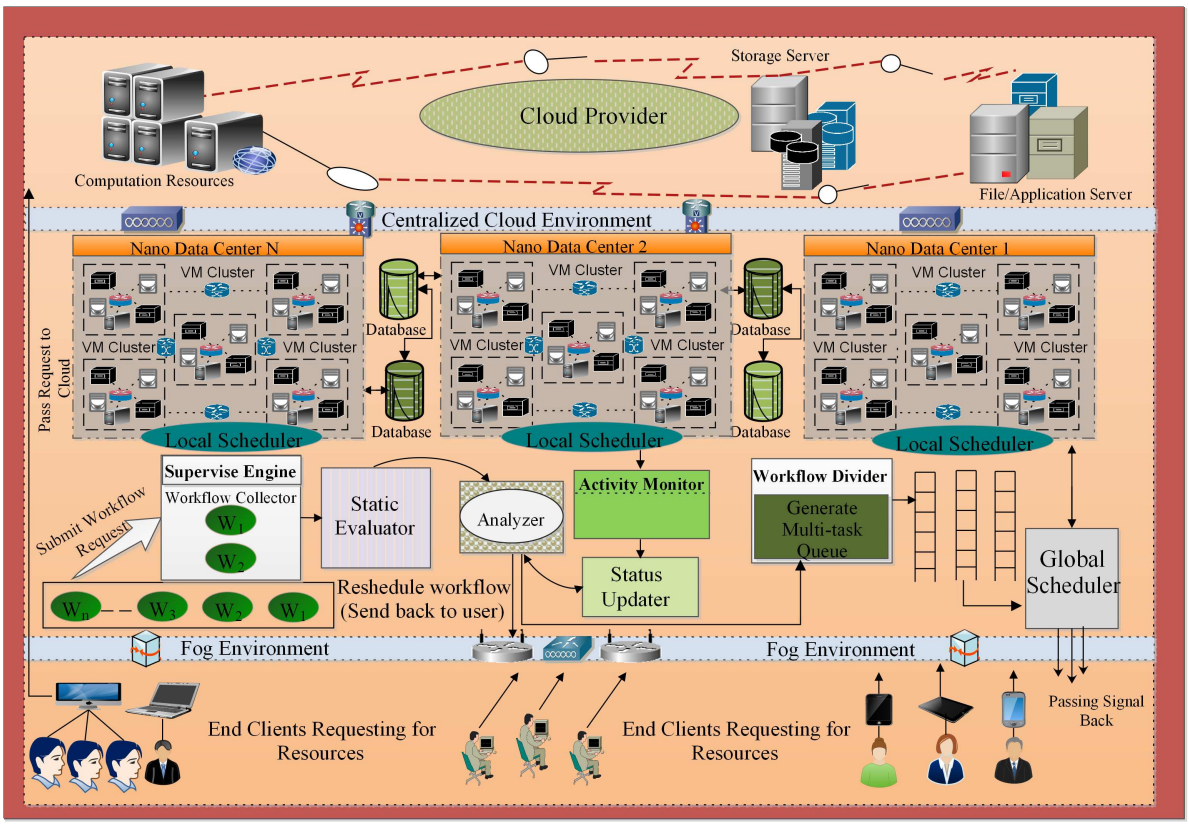

Figure 4.4: Architecture of workflow scheduling over fog-cloud layer

engine is like workflow collector that collects the incoming workflow request and pass request for scheduling to futher modules. At this point the end users has benefit of changing the requirements of workflow only upto then, when the workflow is in Supervise engine. The user get advantage of not submitting the changed workflow request again and again, they make the required changes when the workflow is still in supervise engine by just using the id's that supervise engine is providing. After the request being passed to static evaluator, no future changes is allowed.

The next module of the system is static evaluator. This mainly convert dynamic nature of workflow into static form. The static evaluator stores all the request and does not allow end users to make any futher changes to the workflow request. It means that the behavior of the workflow that is noted here, the system starts observing the requirement of the submitted workflow for scheduling. The basic requirements that this system is considering is budget, execution time, QoS and deadline of the workflow. After that the request enters the next module that is analyzer.

The analyzer plays a most important role in scheduling. It computes that the attributes that are required for workflow scheduling over fog nano data centers are available or not.

The analyzer is futher connected to 2 sub-modules that is activity monitor and status updater module. The activity monitor is attached to local schedulers available in all the nano data centers and keep the track of on-going activities in fog nano data centers. It keeps the record of VMs activities (the available VMs) and update all the information related to availability of the VMs to the next connected module that is status updater. The status updater is always in update form so that analyzer can decide whether the requested workflow can be scheduled over fog nano data centers or pass un-scheduled request back to user for resubmitting again. Now the analyzer works for 2 cases:

1. **Request scheduled successfully:** The request is scheduled means that the available VM clusters $(Unusedspace)$ has greater or equal space then required space.

2. **Re-submit request:** The resubmitted signals will be passed to user because the current available VM cluster space is lesser then the space needed to schedule the workflow.

After having availability of the VMs, the workflow scheduling request get submitted to workflow divider engine. Generally, the two most important workflow types are scientific workflow and business process workflow. But in this workflow scheduling scheme, we only focus on scientific workflow that follows data flow-driven structure. The scientific workflow is represented as a weighted directed acyclic graph (WDAG) that contains a set a tasks and dependencies between these set of tasks. The WDAG consist of vertices and egdes, the vertices of graph representing task set and the edges of graph showing data dependencies and the weight over the edges of graph is used to determine the communication cost.

Now this workflow divider works into two phases. In first phase, it divides the task of the workflow into n queues in order to minimize the scheduling and execution overhead. In second phase these task queues are passed to global scheduler for scheduling and execution of task queues over fog nano data center.

The last module that makes the scheduling possible and schedule the task queues over fog nano data centers containing VM cluster is Global scheduler. It allocates the task over fog nano data centers containing VM cluster based on locality and availability of the VM cluster but keeping the constraints defined by user as first priority. The global scheduler incorporates with local scheduler that resides in each nano data center. If there are n number of nano data centers then there is n local scheduler. There is no common local scheduler in between nano data centers each has a separate local scheduler. We also know that in some cases one local scheduler wants to communicate with local scheduler of some other nano data center then for that there is database repository attached to each nano data centers. So,

the data required can be fetched from database only, in such case no involvement of local scheduler is needed.

4.5 VM Clustering Over the Fog Nano Data Center

4.5.1 Reducing execution overhead by clustering

Fog computing gives a way to run several virtual machine over fog nano data centers in a distributed fashion and allow users to perform processing closer to data source. The VM clustering [24] over fog nano data center is a beneficial technique that allow to service large number of applications, run simultaneously at same time, achieve high productivity and higher profit level for the fog providers. For getting these benefits over fog layer the proper clustering strategy must be used and investigated to avoid large VM migration, overload over fog nano data centers and large energy consumption etc. Now, one of the service that get advantage in execution over VM cluster is scientific workflows. Let a workflow W$< T, E >$ is a directed acyclic graph where E is represented as set of edges among the interdependent task and T is represented as a set of task T=$\{T_1, T_2, T_3, ..., T_n\}$ of the workflow. It is assumed that all of resources are in distributed fashion and currently executing some tasks over it.

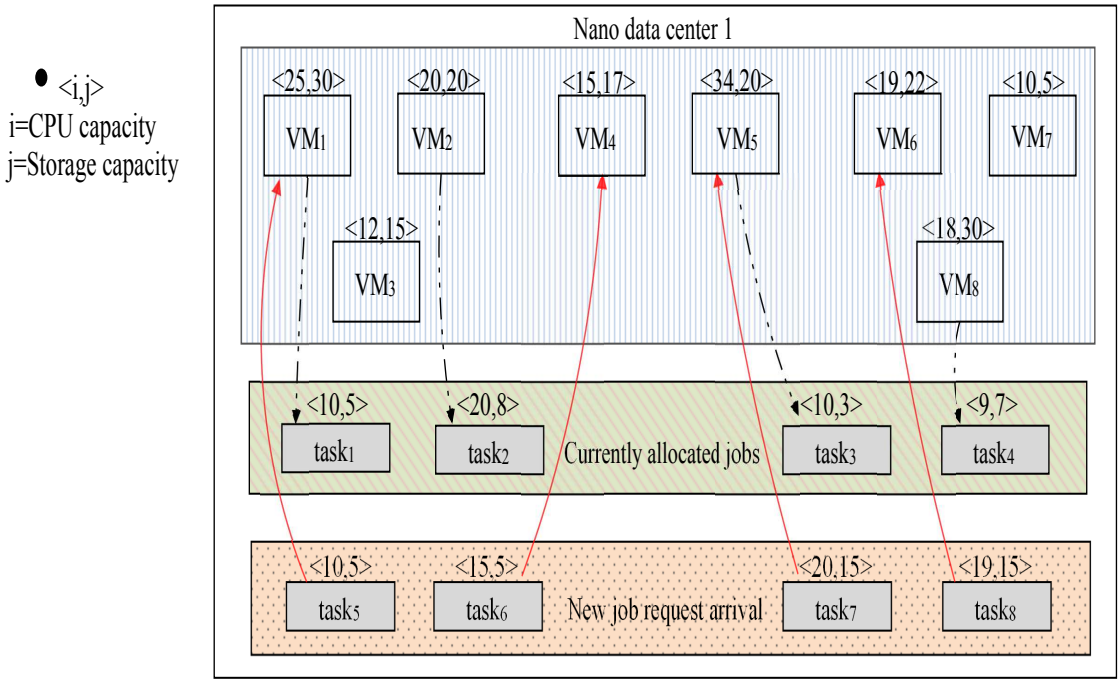

Figure 4.5: VMs clustering based on unused space

The scheduler handles the new arrival request of resource allocation and allocate request over the VM cluster having enough resources to execute user request. But VM allocation is not a simple task as some resources get overloaded while others get underloaded and to solve these issues VM migration technique is used, involving pausing and again starting VMs from one host to other host, leads to degrading the performance of whole system. So it is better to check and cluster the VM according to capacities and grouping the VM with same features into one cluster and even the VM clusters over fog nano data center are easy to monitor, manage and configure rather than single VM. It also give advantage over traditional method that instead of checking individual VM capacities, allowing to check single clusters helping easy allocation and time saving advantage.

We follow the technique for finding the unused space over VMs based on current resource usage and needed space required by coming resource request. By doing same we also determine the underloaded and overloaded VMs, comparing with the resource request, helpful in VM selection. Now to achieve a goal we try to sort the VMs according to their CPU utilization, Storage utilization and depending on their location for that we use K mean algorithm. Fig. 4.5 describe process for the selection of VMs.

4.5.2 VM selection

The example in fig. 4.5 shows the workload scheduling prediction idea, in particular on the basis of future request the load over VM is estimated, based on available capacity. In other words the empty VMs are estimated to schedule future tasks. It contain two steps, each VM total capacity is calculated used for prediction of cluster capacity and next predicting which VM can execute the future load of coming tasks. This technique will help in saving the time by pre-estimating of load. The fig. 4.5 shows the prediction of unused resources that satisfy the coming resource demand of tasks. There are number of distributed VMs, we will choose the VMs that have maximum remaining capacity to meet the task resource demand with the aim of fully resource utilization. We will calculate these prediction over each VM for every task request. To find the best match for the VM clustering, we introduce this concept by calculating unused VMs space.

4.5.3 Checking availability

In checking VM availability we calculate unused space of VMs and then comparing with the future task requests. We calculate availability as shown in following given equation:

$$VM_{Ava} = \sum_{vm=1}^{n} (nV_T - B_{vm}) \tag{4.6}$$

where, VM_{Ava} describe the total available unused space of VMs, n is the total no of VMs whose availability is checked, V_T is total service provided by VMs in given T time period and B_{vm} is optimum busy time period of VMs.

Suppose our proposed model provide two type of resources that is CPU and storage resource in system. We use i to denote the capacity of CPU in the system and j to denote capacity of storage in the system. In fig. 4.5 the $(task_1, task_2, task_3, task_4)$ are currently allocated tasks over distributed VMs and we consider $(task_5, task_6, task_7, task_8)$ are tasks request submitted, that should scheduled in future. We also assume the time slots for scheduling tasks. Let T be total time period where T=$(t_1, t_2, t_3, ..., t_n)$. The currently allocated tasks are scheduled in time t_1 and new task request arrive at time slot t_2. The vector $< i, j >$ describing capacity for each VM of the system. Now $task_1$ is executing over VM_1, $task_2$ over VM_2, $task_3$ over VM_5 and $task_4$ over VM_8 at time slot t_1. Now by using equation 1, the future task execution space can be decided that will help in VM clustering. Now we compare each task capacity required with the distributed VMs and find the most suitable VMs satisfying both CPU capacity required and storage capacity needed. Now the resource demand by $task_5$ is 10 for CPU and 5 for storage. Here the calculation for needed capacity is done in 2 steps.

1. **Calculating needed capacity over VM that is already serving tasks:** It is essential to meet the resource demand of the tasks. For that we will first match the required space needed over the VMs that are serving some other task request, whose available space matches with required resource capacity. It helps in efficient utilization of resources and selecting VMs for VM clustering. If this condition is not matching with the task request then we will follow the next condition.

2. **Calculating needed capacity over the unallocated VMs:** For this condition we will map the needed resource capacity over the used VMs. It means that VMs that are not serving to some other tasks but whose capacity meeting new task request and whose total capacity is totally free. This will also helps in VM clustering.

Following the above two conditions, the required space for $task_5$ (CPU and storage) is available at 1 VM that is meeting resource demand. So comparing their capacity by using equation 1, total unallocated space is 25-10=10 for CPU and 30-5=25 for storage. The required resource space by $task_5$ is $< 10, 5 >$. So, comparing with unused space for CPU $10 \geq 10$ and for storage $25 \geq 5$. So $task_5$ can be scheduled over VM_1. Same case is repeated for other new arrival task $\{6, 7, 8\}$. Repeating above same steps, using equation 1, $task_6$ will be scheduled on VM_4, $task_7$ on VM_5 and $task_8$ on VM_6. So all these VMs help in VM

clustering by sharing unused resource capacity. The above whole process will help in VM clustering for job scheduling.

4.5.4 VM clustering

Clustering is the process of decomposing large data set into smaller data set having same character. In this work, clustering process is done by K-Means [49] algorithm which was introduced by J. Mac Queen in 1967. K-Means clustering algorithm is useful and capable of partition a set of objects in dataset to cluster different type of data quickly and dividing K clusters on the basis of sample data to be tested. For K-Means algorithm the important thing is, knowing midpoint for the creation of clusters through that K partitioning of data set is performed. The basic pseudocode for K-mean clustering [50] which will be related to over proposed system is as follows.

Algorithm 1 Process for K-means algorithm

Step 1:Selecting K cluster.
Step 2: Initializing cluster centroid, μ_i**= numerical value (cluster centeroid)** where $i = (1,2,3,.....,K)$
Step 3: Using euclidean distance, creating distance matrix from x_l to cluster centroid $D_{ij} = \sqrt{\sum(x_l - \mu_i)^2}$ $D_{ij} \leftarrow$ **Euclidean distance between l^{th} data point to i^{th} cluster center.**
Step 4:Reallocate x_j to closet cluster.
Step 5: Again calculating centroid of each clusters.
Step 6:Repeat 3, 4 and 5 until centroid does not change.
END

Let us consider a fog system having n nano DCs(DC_n)=$\{dc_1, dc_2, dc_3, ..., dc_n\}$ and every nano data center have some virtual resources or virtual nodes V_x=$\{v_1, v_2, v_3, ..., v_x\}$. The fog system consist of (fs)=$< DC_n, V_x >$. Now considering each virtual node providing some resource availability that is, storage node (SN_i)=$\{s_1, s_2, ..., s_i\}$ and CPU node (CN_j)=$\{c_1, c_2, ..., c_j\}$. Let task list that need to be executed over VM clusters is T_l=$\{t_1, t_2, t_3, ..., t_l\}$.

Now we have to cluster the VM according to availability and build the nano DCs by knowing cluster capacity. For making VM clusters, first thing that we have to determine unused CPU capacity and unused storage capacity. Let U_{CPU} be unused CPU capacity and $U_{storage}$ be unused storage capacity, A_{CPU} be available CPU capacity and $A_{storage}$ be available storage capacity , A'_{CPU} be allocated CPU capacity and $A'_{storage}$ be allocated storage capacity that need to be calculated for VM clustering.

1. For identifying unused VMs, the first step is to find available CPU capacity and

available storage capacity. Let us assume α be the available CPU capacity and β be the available storage capacity. The available VM capacity is the total capacity available over distributed VMs. Like in fig.4.5 the available VM capacity of VM_1 is 25. so the available VM capacity denoted by α is given by formula:

$$\alpha = max\{V_n^{A_{CPU}} : n = \{1,2,.....,j\} \tag{4.7}$$

Same step is performed to calculate available storage capacity, denoted by β. So β is defined as:

$$\beta = max\{V_n^{A_{storage}} : n = \{1,2,.....,i\} \tag{4.8}$$

2. Now next step to make VM cluster is to find allocated VM capacity and allocated storage capacity. Let γ be the allocated CPU capacity and δ be the allocated storage capacity. These are those space that is currently executing some jobs. Like in fig. 4.5 the allocated VM space for VM_1 is 5 that is executing job_1 over it. So the allocated CPU capacity is:

$$\gamma = max\{V_n^{A'_{CPU}} : n = \{1,2,.....,j\} \tag{4.9}$$

Same step is performed to calculate allocated storage capacity, denoted by δ. So δ is defined as:

$$\delta = max\{V_n^{A'_{storage}} : n = \{1,2,.....,i\} \tag{4.10}$$

3. Now from above equations the unused VM space for VM clustering can be calculated using equation 4.6. From the unused CPU and storage, we can determine the capacity of the clusters. The unused CPU capacity, denoted as U_{CPU} is given as:

$$U_{CPU} = \{\alpha - \gamma\} \tag{4.11}$$

Now same above step is performed to calculate unused storage capacity denoted as $U_{storage}$, that helps in forming VM cluster.

$$U_{storage} = \{\beta - \delta\} \tag{4.12}$$

By following above process, the unused CPU space and unused storage has calculated that is now used to form VM cluster. So the nano data center that is formed by these unused VM space is described as:

$$nDC_{capacity} =< U_{CPU}, U_{storage} > \tag{4.13}$$

4. To make the VM cluster, now the cluster capacity can be determined by using nano data center capacity and making K clusters. The cluster capacity is denoted as $C_{capacity}$, here K is the number of VM available in every cluster and calculated below:

$$C_{capacity} =< nDC_{capacity} * K > \tag{4.14}$$

4.5.5 Calculating centroid

Now for K-means algorithm the centroid calculation is very crucial step that helps in VM clustering [51].

1. The first step in centroid calculation is to input the data set. Let the data set D_i= =$\{d_1, d_2, ..., d_i\}$ that is input for centroid calculation.

2. For i objects calculate the value that is average of n-objects. We can also use the below given equation for centroid calculation that is denoted by σ_i

$$\sigma_i = \frac{1}{d\,i} \sum_{k \in d_i} x_k \tag{4.15}$$

3. After centroid calculation, now compare the dataset values to it, assign values to the closest centroid.

4. After changing data set, again calculate centroid and repeat step 3 until the centroid do not change.

4.5.6 Pseudo code

The VM clustering technique [52] helps in efficient utilization of resources. It also prove as efficient method by reducing execution overhead of task scheduling over distributed VMs. For our proposed approach we have described the pseudo code, that describe the process for task division, VM clustering and resource scheduling.

4.5.7 Scheduling by cluster formation

We propose a clustering algorithm that checks the availability of VMs and select the VMs that match the user resource request and find the best match, also providing QoS. After selecting VM we cluster the VM according to their capabilities and finding centroid by using K-Means algorithm and then schedule request over VM clusters [24]. Fig. 4.6 show the scheduling of workflow task by formating clusters. First submit workflow to the workflow divider, also describing workflow specifications with the request. The workflow divider divides the workflow into task set. These task execute independently and share the result of each task to get a final output for a workflow. The example of montage workflow is taken, we divide the workflow into N-task steps. These sub-task set is passed to scheduler for scheduling over VM clusters. Here in fig. 4.6 we consider the system having 2 nano data center and we schedule request over VM clusters for executing user request in real time. Thus the system perform best and provider better scheduling in real time over fog layer.

Algorithm 2 Algorithm for VM Clustering based on unused VMs

Input: $VM^{req}, , VM_x^{avl}, S_i^{list}\ CPU_j^{list}\ T_l^{list}$

Output: $V^{cluster}, CPU^{unuesd}, S^{unused}$

1: Using the VM capacity required list VM^{req} by users check VM availability VM_x^{avl}
2: **for** (x=1; x $\leq$ n; j++)**do**
$n \leftarrow$ Number of DCs
3: Check for available VM capacity $VM_{capaciity}^{avl}$
4:index=0
5:Check unused storage and unused CPU in S_i^{list} and CPU_j^{list}
6:Calculate unused capacity using Eq. (4.6)
7:Check **if** VM^{req} match with capacity available, $VM_{capaciity}^{avl}$
8: **if** ($VM^{req} < VM_{capaciity}^{avl}$) **then**
9:index=index+1
10: **else**
11:Add request in waiting queue (Q)
12: **endif**
13:**endfor**
14: **for** each ($VM_{capaciity}^{avl}$) **d**o
15: Calculate centroid for VM clustering using Eq. (10)
16:Calculate nearest centroid for each index
17: Remove VM from index
18: Add VM to nearest cluster
19: Repeat until centroid do not change
20: **EndFor**

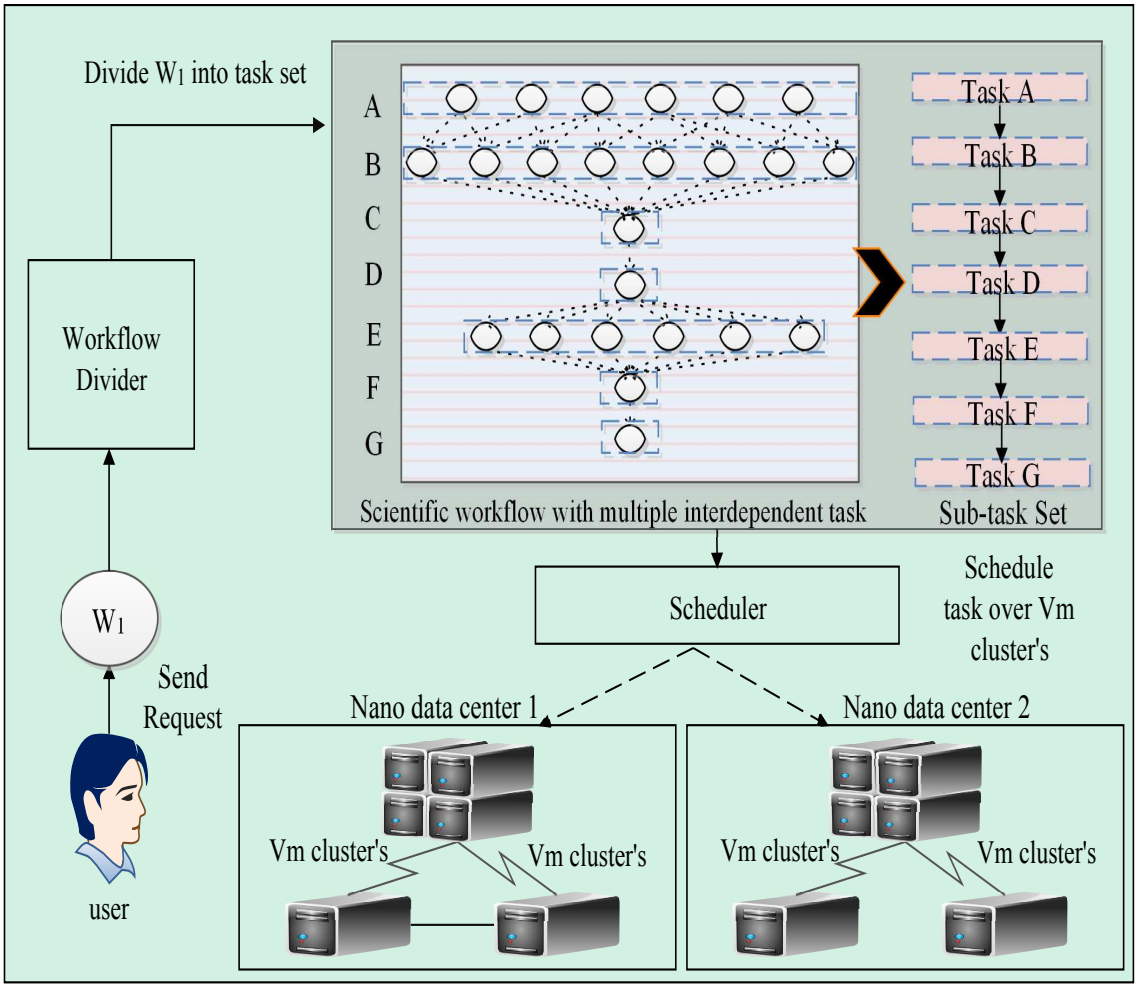

Figure 4.6: Architecture of workflow scheduling over fog-cloud layer

Chapter 5

Performance Evaluation

This chapter focuses on the proposed scheme performance and result evaluation of our proposed results. CloudSim and WorkflowSim tool is used to create the simulation environment.

5.1 Container-Based Edge Service Management System (CoESMS)

This section investigates the impact of the proposed CoESMS on the overall energy utilization at the edge of the network, while ensuring adherence to the SLA. It is implemented using an extended version of the popular cloud simulator CloudSim, Container CloudSim [53]. All this simulation is done in NetBeans. It provides a containerized environment with respect to cloud computing for modeling and simulation. This is done by comparing the proposed scheme with an existing scheme. This operation of the existing scheme can be summarized as follows.
It selects and schedules the tasks on available containers on a first-come first-serve basis. Additionally, the broker selection is relatively simple: a particular broker is selected based on its availability and its current workload, in contrast to our proposed scheme, which is based on the game theoretical concept.

The experimental simulation tests were repeated 25 times across a 24-hour simulation period. The related simulation parameters with respect to nDCs, VMs, and containers are summarized in Table 5.1. A total of 200 nDCs, 450 VMs, and 750 containers were used to perform extensive simulations. Additionally, four different types of VMs and three types of containers were also considered, wherein each VM has different CPU and memory config-

Table 5.1: Simulation setup with respect to the configurations of nDCs, VMs and containers

Nano Data center configuration and power characteristics							
nDC type		CPU [3GHz] cores	Memory (GB)	P_{idle}(Watt)	P_{max} (Watt)	Number of nDCs	
1		8	128	93	135	200	
Container and virtual machine configurations							
Container Type	CPU MIPS (1 core)	Memory (GB)	Number of containers	VM Type	CPU cores	Memory (GB)	Number of VMs
1	4658	128	250	1	2 cores	1	113
2	9320	256	250	2	4 cores	2	113
3	18636	512	250	3	1 cores	4	112
				4	8 cores	8	112

urations, as shown in 5.1. To evaluate the two schemes based on the above simulation setup, workload traces from PlanetLab were used [54]. The performance of the two schemes were compared using the following four performance metrics.

1. **Total number of container migrations:** This metric evaluates the total number of internal and external container migrations so as to minimize the total energy utilization of the servers available at the edge of the network. A simple threshold-based mechanism is considered for this purpose where the utilization levels of servers is above 80 percent or below 50 percent (if utilization is below 50 percent, migration is also done in the proposed scheme).

2. **Container overhead analysis:** Although containers are lightweight entities, they do have associated overheads, especially during their migrations. This overhead is due to the container startup time and number of migrations done.

3. **Average number of SLA violations:**The SLA metric is defined according to [53,55]. It is considered to be violated when a VM hosting container(s) does not get its requested CPU share. Alternatively, it can also be defined as the difference between the required and allocated CPU share of each VM.

4. **Total energy consumption (kWh):** This metric represents the amount of energy actually utilized by the individual servers to provide various services to the end users. The total energy consumption of the nDC is calculated by aggregating the individual energy consumption of its servers. Here, CPU utilization is chosen as an important

parameter to estimate energy consumption using the linear model as defined in [53, 56].

5.2 Workflow Scheduling Heuristics

In this section, the simulation results for proposed workflow scheduling heuristic is shown by using specific parameters. For evaluating the performance of workflow, some of very popular real world scientific workflow [35, 36] is used like Montage, CyberShake, Inspiral and Epigenomics. We use different task nodes for each workflow type to evaluate the performance better and showing our approach results.

5.2.1 Experimental parameters

The experiment is carried out using Cloudsim framework version 3.0 and workflow simulation tool version 1.1.0 on an Intel Celeron 29572U processor, 2 GB RAM and CPU 1.40 GHz running on Microsoft Window 7 Ultimate platform. For this simulation we have created 20 VMs and allocating these VMs over three nano DCs. By using k-means procedure all VMs with same computational capacity will be allocated in 1 cluster. Few matrix are used to evaluate the performance that are elaborated below.

1. **Completion time:** It is the calculated time that describes each task of the workflow is executed or completed in given specific time.

2. **RAM size:** It describes the needed capacity to create VM over nano DC.

3. **Depth of workflow:** This parameter describe the level of workflow containing various task to be executed.

The main scheduling approach that this implementation follows is to look for the needed RAM for the implementation of VM, needed by required workflow task nodes for execution.

5.3 Results of Proposed CoESMS

Containers are virtual entities that are more lightweight than VMs. Containers also support seamless internal and external migrations without imposing any significant overheads on the underlying systems CPU and memory utilization and network bandwidth. Thus, the proposed CoESMS efficiently schedules the tasks in an energy-efficient manner using

containers. Fig. 5.1 shows that the average number of container migrations is higher with CoESMS when compared to the other scheme by 8.42 percent. This directly implies that the proposed scheme minimizes the total energy consumption of the servers by triggering container migrations whenever required.

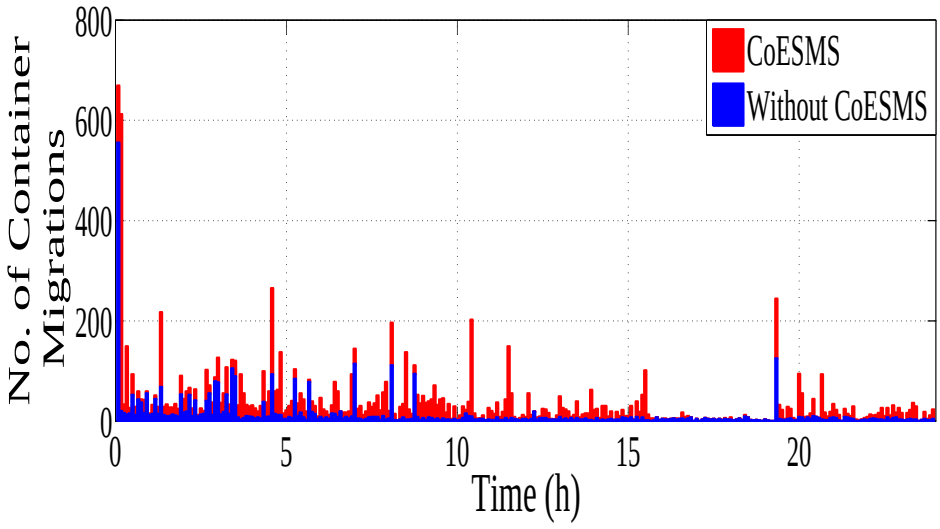

Figure 5.1: Number of container migrations with respect to time.

In the proposed scheme, container migrations are initiated based on the servers utilization level as mentioned above. As containers are relatively lightweight entities in comparison to VMs, they generate negligible computational and network overhead. However, the main overhead with containers is caused by the containers startup time. Hence, the overhead analysis of both schemes has been done with respect to the average startup time per container migration. Our results are depicted in fig. 5.2, which shows that the average startup time of the containers in both schemes is comparable. This is because the containers share the kernel instances and support quick startup times compared to VMs.

In addition, the experimental results shown in fig. 5.3 further show that the proposed CoESMS also reduces the number of SLA violations compared to the existing scheme. This can be attributed to the fact that the proposed scheme maps the tasks to containers based on the multi-objective game theoretical approach, which formulates the game among the containers and selects the most optimal task for execution. As a result, the proposed scheme achieves an 11.82 percent decrease in the number of SLA violations compared to the existing scheme.

Finally, the overall energy utilization (24 hours) of simulation is shown in fig. 5.4.

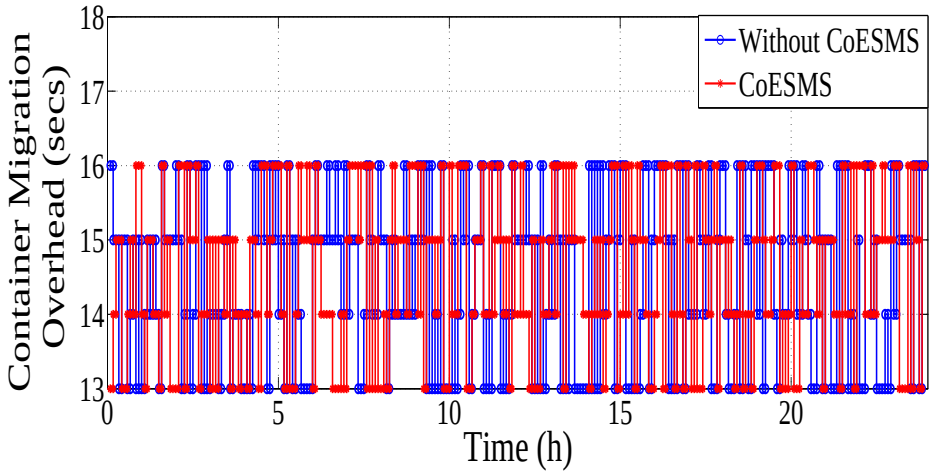

Figure 5.2: Overhead analysis of the schemes with respect to number of container migrations per unit time.

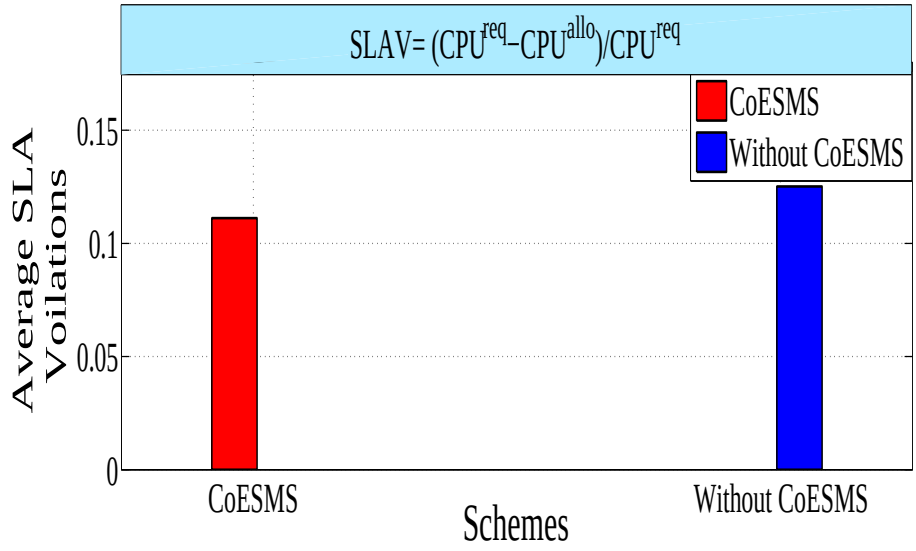

Figure 5.3: Average number of SLA violations observed during task scheduling.

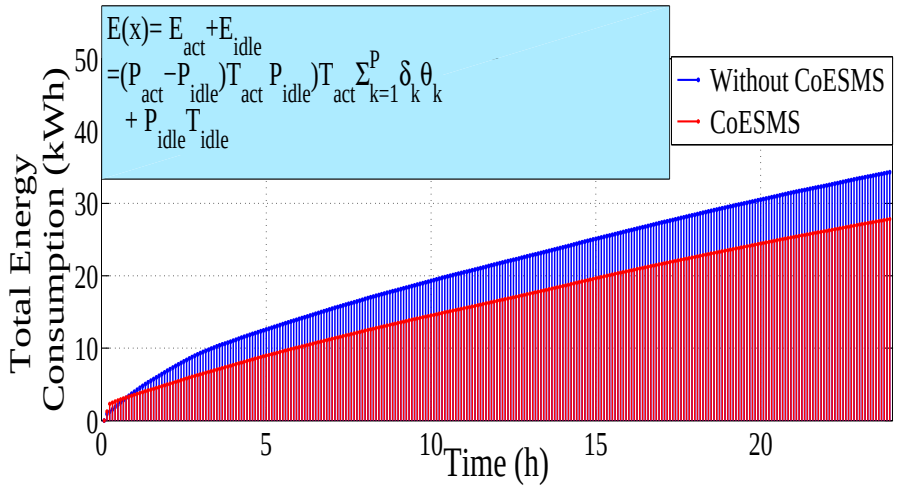

Figure 5.4: Total energy consumption of nDCs with respect to time.

5.4 Results of Workflow Heuristic

Table 5.2: Description for used scientific workflows

Scientific Workflow Characteristics			
Workflow Name	Total Number of Task	Workflow Type	Required Parameters(Memory, CPU Time)
Montage	15-50	Small	Low, Low
Montage	51-100	Medium	Low, Low
Montage	100-600	Large	Medium, Low
Inspiral	15-50	Small	Low Low
Inspiral	51-100	Medium	Low, Low
Inspiral	100-600	Large	Medium, Low
Sipht	15-50	Small	Low, Low
Sipht	51-100	Medium	Low, Low
Sipht	100-600	Large	Medium, Low
Epigenomics	15-50	Small	Low, Low
Epigenomics	51-100	Medium	Low, Low
Epigenomics	100-600	Large	Medium, Low

For the implementation of our proposed algorithm we use four scientific workflows

shown in table 1 having different task capacity and different dependencies among tasks. For scheduling of workflows over VMs we have used different parameter for each task node of workflows that are $< jobid, runtime, sizeofworkflow, categoryofworkflow >$. The host that we create having RAM 2048mb and Storage 1000000 having different host id's.

In table 5.2 it is shown that we divide our workflow into three categories that is small (15-50 task nodes), medium (51-100) and large ($\leq$600).

We have shown the result of proposed scheme and existing scheme by running each workflow over workflowsim tool. Executing large task nodes of workflow in few seconds and also describes the procedure of VM scheduling by knowing available RAM of host , total time of execution and status of workflow task.

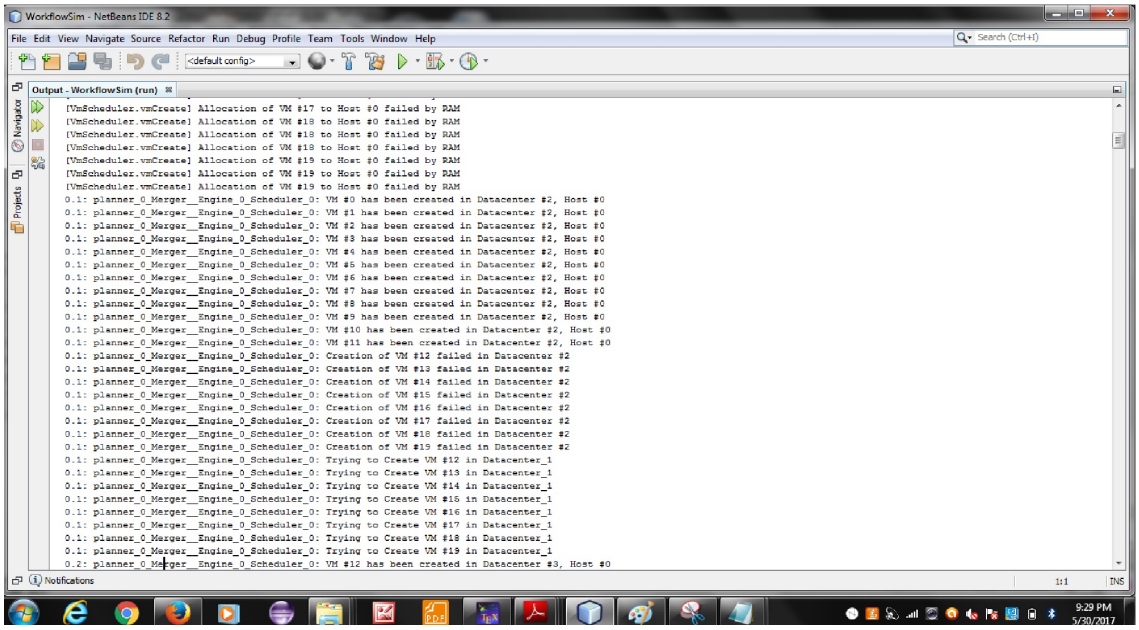

Figure 5.5: Screenshot of VM creation by checking RAM

In fig. 5.5 the VM creation is shown by checking RAM availability for scheduling user tasks.

In fig. 5.6 it is shown that the VMs are created over different nano data center by checking available RAM space. By checking the space availability in advance and creating VMs according to the space helps in faster execution of workflow and also helps in reducing schedule time.

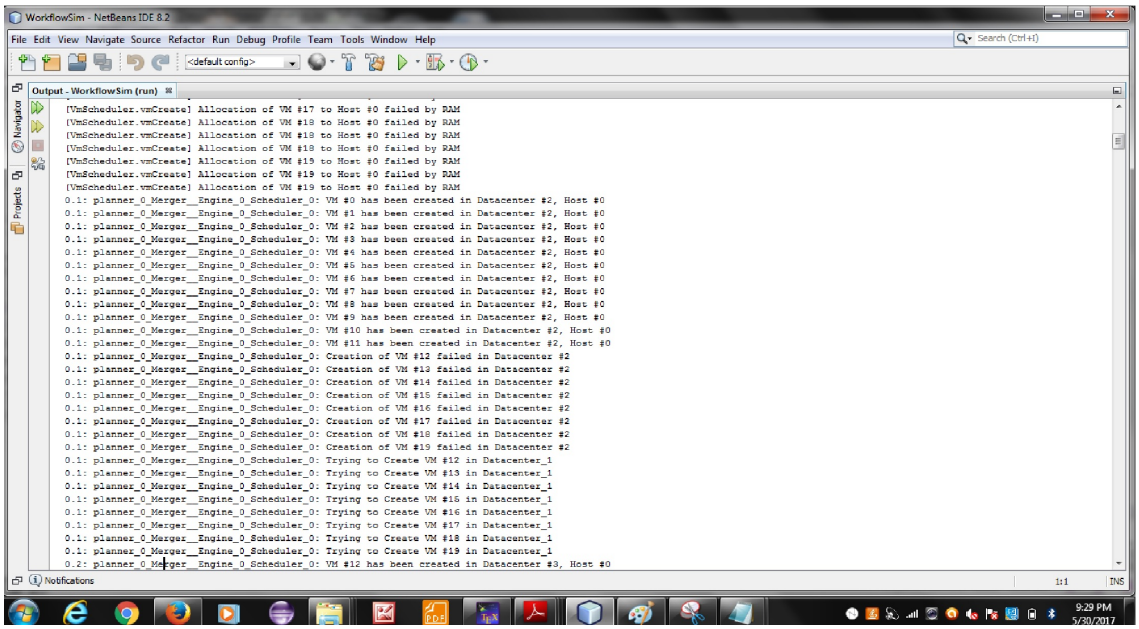

Figure 5.6: Screenshot of VM creation over the host

In fig. 5.7 it is shown the execution of Montage workflow having 53 task nodes belongs to medium category range, having total depth 9 get executed over VMs in time 38 seconds.

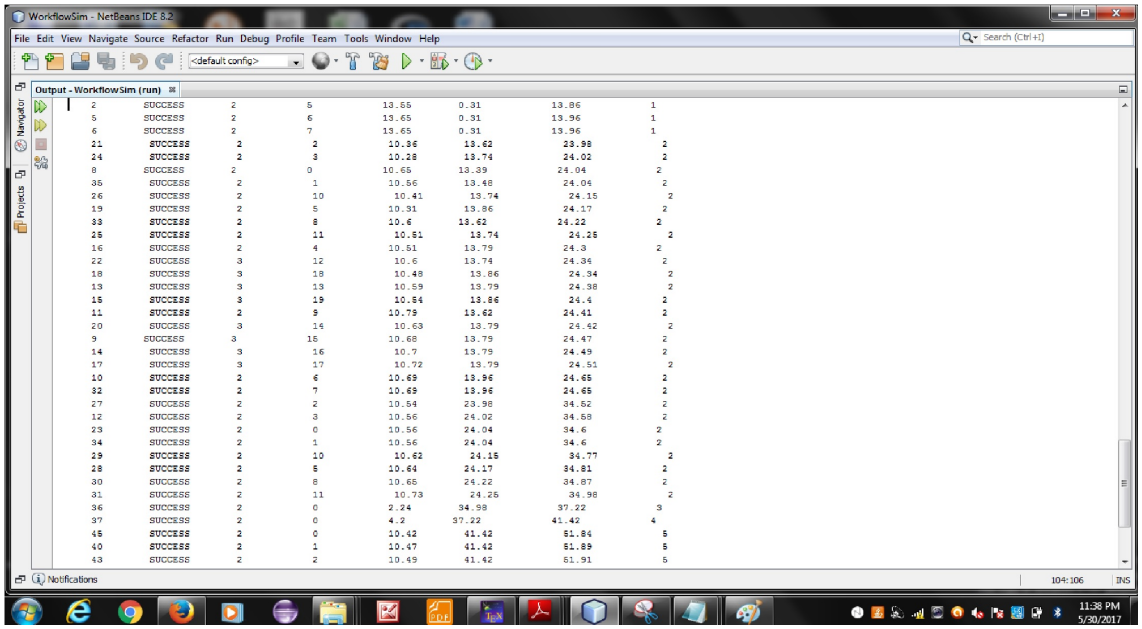

Figure 5.7: Screenshot of executing Montage workflow with 53 task nodes

In fig. 5.8 it is shown the execution of Inspiral workflow having 100 task nodes belongs to medium category range, having total depth 6 get executed over VMs in time 38 seconds.

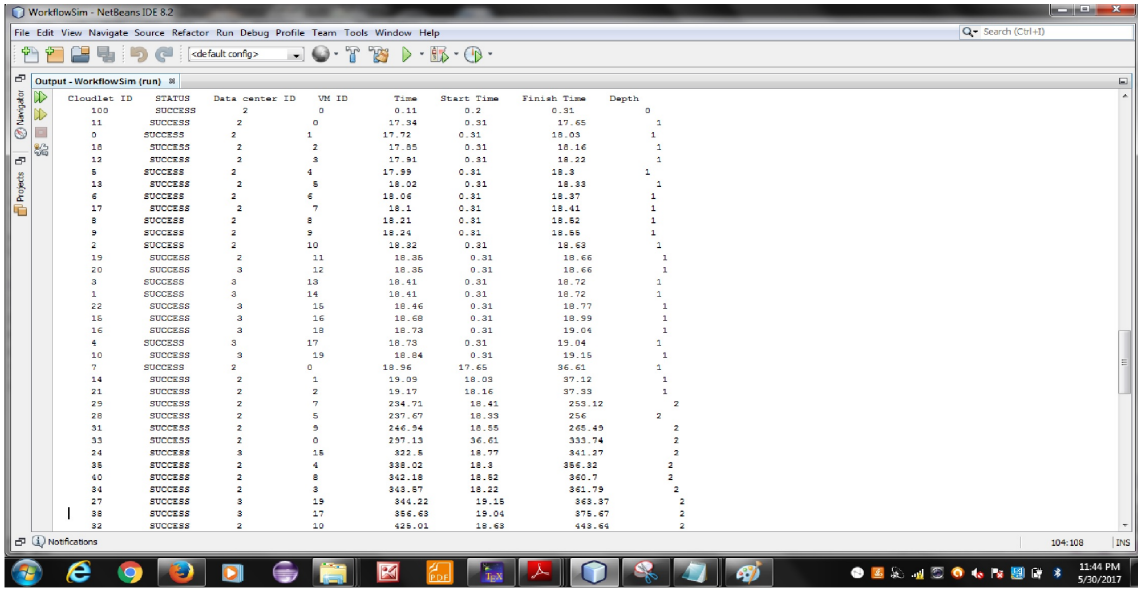

Figure 5.8: Screenshot of executing Inspiral workflow with 100 task nodes

In fig. 5.9 it is shown the execution of Sipht workflow having 484 task nodes belongs to large category range, having total depth 5 get executed over VMs in time 90 seconds.

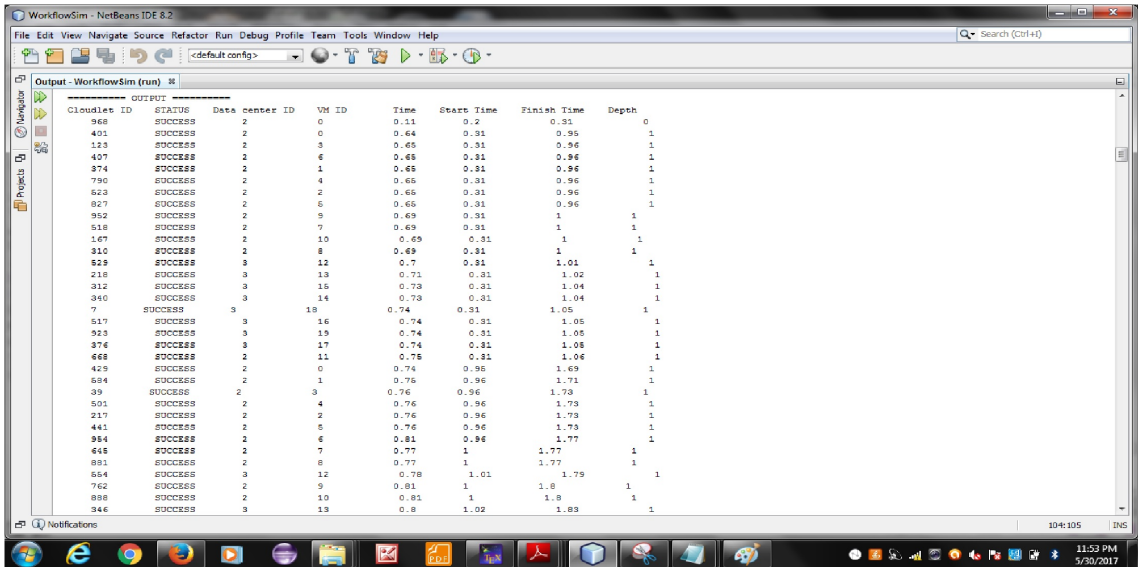

Figure 5.9: Screenshot of executing Sipht workflow with 484 task nodes

The execution time of various workflows and their depth is shown in table 5.3 and also represent graphically in fig. 5.10.

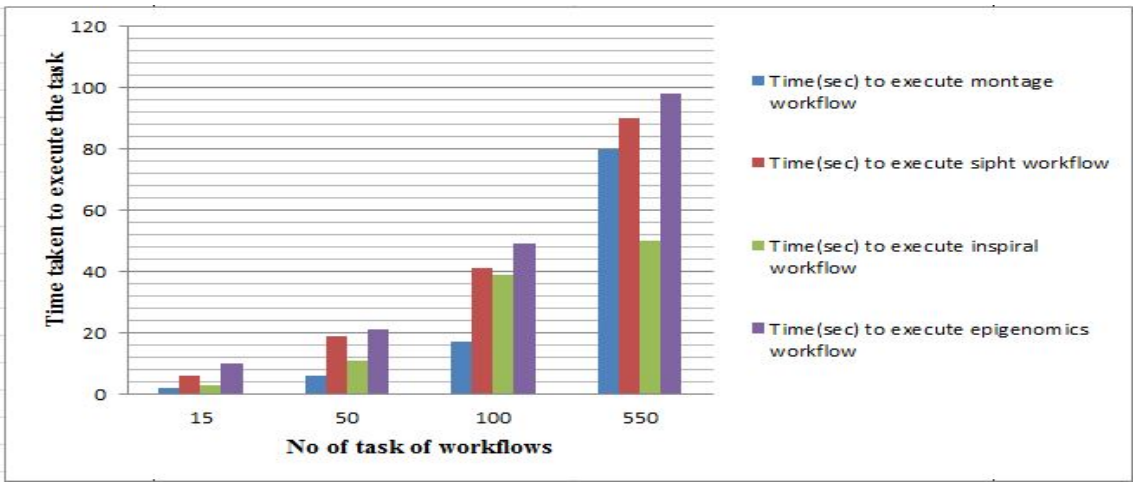

Figure 5.10: Time taken to execute different workflow task nodes

The execution time increases as the no of task of workflow increases. We have proved that our proposed scheme is better then other schemes, providing better results, showing maximum utilization of service, providing quality of service to user and it is shown in graphical form 5.11. Showing comparison of proposed scheme with Resource-Aware-Scheduling algorithm (RASA), Round-robin algorithm (RR) and First Come First Served (FCFS) [57].

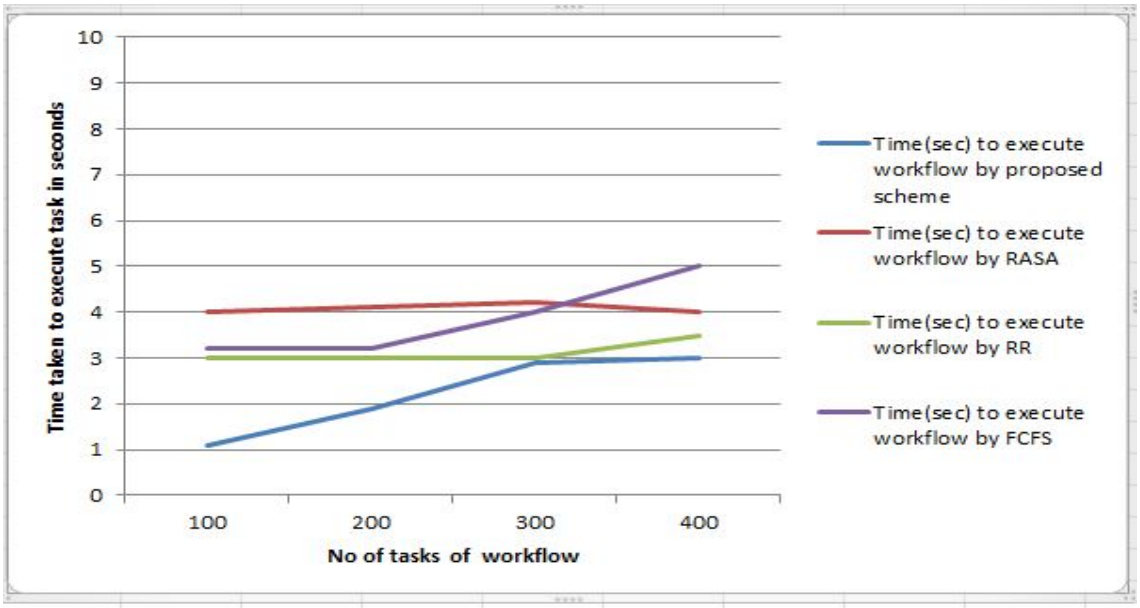

Figure 5.11: Execution time comparison between different schemes

Table 5.3: Showing depth and execution time of various scientific workflows

Various workflow execution component					
Workflow Type	Execution Time of Workflow	Depth	Workflow Type	Execution Time of Workflow	Depth
Montage workflow with 17 task nodes	3 seconds	9-levels	**Montage workflow with 28 task node**	7 seconds	9-levels
Montage workflow with 53 task nodes,	29 seconds	9-levels	**Montage workflow with 503 task node**	48 seconds	9-levels
Sipht workflow with 16 task nodes	6 seconds	5-levels	**Sipht workflow with 30 task nodes**	19 seconds	5-levels
Sipht workflow with 50 task nodes	41 seconds	5-levels	**Sipht workflow with 484 task nodes**	90 seconds	5-levels
Inspiral workflow with 18 task nodes	3 seconds	6-levels	**Inspral workflow with 26 task nodes**	11 seconds	6-levels
Inspiral workflow with 52 task nodes	38 seconds	6-levels	**Inspiral workflow with 502 task nodes**	50 seconds	6-levels
Epigenomics workflow with 16 task nodes	10 seconds	8-levels	**Epigenomics workflow with 46 task nodes**	21 seconds	9-levels
Epigenomics workflow with 100 task nodes	48 seconds	8-levels	**Epigenomics workflow with 503 task nodes**	98 seconds	9-levels

Chapter 6

Conclusion and Future Scope

In this thesis, an attempt has been made to solve the task and workflow scheduling problem using proposed Container-Based Edge Service Management System (CoESMS) for task scheduling and Workflow Scheduling Model over Fog-Cloud Layer. The main contribution of this thesis are done in several phases and they are shown as follows:

1. To enable seamless, real-time data processing, the concept of edge/fog computing, that is the extension of cloud is proposed, that make ease of high quality data transfer to distributed end users.

2. We propose Container-Based Edge Service Management System (CoESMS) for energy-efficient task selection and scheduling using the cooperative game theory.

3. A game theoretical models is developed between broker to broker and container to container in order to minimize the overall energy utilization of servers and makespan of tasks while scheduling the tasks on lightweight containers.

4. Extensive simulations have been performed on workload traces obtained from PlanetLab and the results indicate that our proposed scheme performs better than the existing scheme with an energy utilization reduction of almost 21.75% and on average, a decrease of 8.42% for the number of SLA violations.

5. In the workflow scheduling scheme the VM clustering scheme is proposed using K-Means algorithm.

6. The objective of workflow scheduling proposed scheme is to provide real time service to scientific applications and to reduce execution overhead by VM clustering.

7. We shows that optimal utilization of resources have achieved by using unused space of VM first, that leads to reduce monitoring and management overhead and improve computational granularity of fog system.

8. The proposed scheme shows that our workflow scheduling approach works faster then Resource-Aware-Scheduling algorithm (RASA), Round-robin algorithm (RR) and First Come First Served (FCFS).

6.1 Scope for future work

Despite our contribution in current thesis for task and workflow scheduling in fog computing environment, there are number of uncovered research issues that need to be addressed. Since research is a continuous procedure so some of the suggestion for future work is described below.

1. We plan to extend proposed task scheduling algorithm for rescheduling tasks by calculating networkload.

2. The workflow scheduling algorithm is extended by using other factor apart from deadline like energy consumption.

3. In future we may propose new optimal efficient method to solve workflow scheduling problem.

Bibliography

[1] O. Osanaiye, S. Chen, Z. Yan, R. Lu, K. Choo, and M. Dlodlo, "From cloud to fog computing: A review and a conceptual live vm migration framework," *IEEE Access*, 2017.

[2] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, no. 2011, pp. 1–11, 2011.

[3] C. Wang, Z. Bi, and L. Da Xu, "Iot and cloud computing in automation of assembly modeling systems," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1426–1434, 2014.

[4] X. Qiu, Y. Dai, Y. Xiang, and L. Xing, "A hierarchical correlation model for evaluating reliability, performance, and power consumption of a cloud service," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 3, pp. 401–412, 2016.

[5] T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. Ieee, 2010, pp. 27–33.

[6] W. Tan, Y. Sun, L. X. Li, G. Lu, and T. Wang, "A trust service-oriented scheduling model for workflow applications in cloud computing," *IEEE Systems Journal*, vol. 8, no. 3, pp. 868–878, 2014.

[7] U. Wajid, C. Cappiello, P. Plebani, B. Pernici, N. Mehandjiev, M. Vitali, M. Gienger, K. Kavoussanakis, D. Margery, D. G. Perez *et al.*, "On achieving energy efficiency and reducing co 2 footprint in cloud computing," *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 138–151, 2016.

[8] G. Rastogi and R. Sushil, "Cloud computing implementation: Key issues and solutions," in *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*. IEEE, 2015, pp. 320–324.

[9] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of internet of things," *IEEE Transactions on Cloud Computing*, 2015.

[10] M. Aazam and E.-N. Huh, "Fog computing: The cloud-iot\/ioe middleware paradigm," *IEEE Potentials*, vol. 35, no. 3, pp. 40–44, 2016.

[11] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, and R. S. Tucker, "Fog computing may help to save energy in cloud computing," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1728–1739, 2016.

[12] M. Aazam and E.-N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*. IEEE, 2014, pp. 464–470.

[13] W. Lee, K. Nam, H.-G. Roh, and S.-H. Kim, "A gateway based fog computing architecture for wireless sensors and actuator networks," in *Advanced Communication Technology (ICACT), 2016 18th International Conference on*. IEEE, 2016, pp. 210–213.

[14] Z. Hao, E. Novak, S. Yi, and Q. Li, "Challenges and software architecture for fog computing," *IEEE Internet Computing*, vol. 21, no. 2, pp. 44–53, 2017.

[15] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," *arXiv preprint arXiv:1601.02752*, 2016.

[16] F. Al-Doghman, Z. Chaczko, A. R. Ajayan, and R. Klempous, "A review on fog computing technology," in *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 001 525–001 530.

[17] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for iot services in the fog," in *Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on*. IEEE, 2016, pp. 32–39.

[18] S. Chakraborty, S. Bhowmick, P. Talaga, and D. P. Agrawal, "Fog networks in healthcare application," in *Mobile Ad Hoc and Sensor Systems (MASS), 2016 IEEE 13th International Conference on*. IEEE, 2016, pp. 386–387.

[19] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare internet of things: A case study on ecg feature extraction," in *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence*

and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on. IEEE, 2015, pp. 356–363.

[20] M. A. Al Faruque and K. Vatanparvar, "Energy management-as-a-service over fog computing platform," *IEEE Internet of Things Journal*, vol. 3, no. 2, pp. 161–169, 2016.

[21] Y. Xiao and C. Zhu, "Vehicular fog computing: Vision and challenges," in *Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on*. IEEE, 2017, pp. 6–9.

[22] S. Selvarani and G. S. Sadhasivam, "Improved cost-based algorithm for task scheduling in cloud computing," in *Computational intelligence and computing research (iccic), 2010 ieee international conference on*. IEEE, 2010, pp. 1–5.

[23] S. Bilgaiyan, S. Sagnika, and M. Das, "Workflow scheduling in cloud computing environment using cat swarm optimization," in *Advance Computing Conference (IACC), 2014 IEEE International*. IEEE, 2014, pp. 680–685.

[24] V. Chavan and P. R. Kaveri, "Clustered virtual machines for higher availability of resources with improved scalability in cloud computing," in *Networks & Soft Computing (ICNSC), 2014 First International Conference on*. IEEE, 2014, pp. 221–225.

[25] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. IEEE, 2014, pp. 1–8.

[26] S. Parsa and R. Entezari-Maleki, "Rasa: A new task scheduling algorithm in grid environment," *World Applied sciences journal*, vol. 7, no. Special issue of Computer & IT, pp. 152–160, 2009.

[27] C. Lin and S. Lu, "Scheduling scientific workflows elastically for cloud computing," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 746–747.

[28] S. Ambike, D. Bhansali, J. Kshirsagar, and J. Bansiwal, "An optimistic differentiated job scheduling system for cloud computing," *International Journal of Engineering Research and Applications (IJERA)*, vol. 2, no. 2, pp. 1212–1214, 2012.

[29] H. Zhong, K. Tao, and X. Zhang, "An approach to optimized resource scheduling algorithm for open-source cloud systems," in *ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual*. IEEE, 2010, pp. 124–129.

[30] S. Wu, H. Chen, S. Di, B. Zhou, Z. Xie, H. Jin, and X. Shi, "Synchronization-aware scheduling for virtual clusters in cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2890–2902, 2015.

[31] D. Kliazovich, P. Bouvry, and S. U. Khan, "Dens: data center energy-efficient network-aware scheduling," *Cluster computing*, vol. 16, no. 1, pp. 65–75, 2013.

[32] J. Meena, M. Kumar, and M. Vardhan, "Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint," *IEEE Access*, vol. 4, pp. 5065–5082, 2016.

[33] S. Ghanbari and M. Othman, "A priority based job scheduling algorithm in cloud computing," *Procedia Engineering*, vol. 50, pp. 778–785, 2012.

[34] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on iaas cloud systems," *Journal of Parallel and Distributed Computing*, vol. 72, no. 5, pp. 666–677, 2012.

[35] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, and Y. Yang, "A compromised-time-cost scheduling algorithm in swindew-c for instance-intensive cost-constrained workflows on cloud computing platform," *International Journal of High Performance Computing Applications*, 2010.

[36] M. Adhikari and T. Amgoth, "Efficient algorithm for workflow scheduling in cloud computing environment," in *Contemporary Computing (IC3), 2016 Ninth International Conference on*. IEEE, 2016, pp. 1–6.

[37] N. Chopra and S. Singh, "Deadline and cost based workflow scheduling in hybrid cloud," in *Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on*. IEEE, 2013, pp. 840–846.

[38] L. F. Bittencourt, C. R. Senna, and E. R. Madeira, "Scheduling service workflows for cost optimization in hybrid clouds," in *Network and Service Management (CNSM), 2010 International Conference on*. IEEE, 2010, pp. 394–397.

[39] L. Wang, R. Duan, X. Li, S. Lu, T. Hung, R. N. Calheiros, and R. Buyya, "An iterative optimization framework for adaptive workflow management in computational clouds," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*. IEEE, 2013, pp. 1049–1056.

[40] E. N. Watanabe, P. P. Campos, K. R. Braghetto, and D. M. Batista, "Energy saving algorithms for workflow scheduling in cloud computing," in *Computer Networks and Distributed Systems (SBRC), 2014 Brazilian Symposium on*. IEEE, 2014, pp. 9–16.

[41] I. Gupta, M. S. Kumar, and P. K. Jana, "Task duplication-based workflow scheduling for heterogeneous cloud environment," in *Contemporary Computing (IC3), 2016 Ninth International Conference on*. IEEE, 2016, pp. 1–7.

[42] S. Prathibha, B. Latha, and G. Sumathi, "Improving energy efficiency of computing servers and communication fabric in cloud data centers," in *Research in Computational Intelligence and Communication Networks (ICRCICN), 2016 Second International Conference on*. IEEE, 2016, pp. 17–21.

[43] A. Deldari, M. Naghibzadeh, and S. Abrishami, "Cca: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud," *The journal of Supercomputing*, pp. 1–26, 2016.

[44] C. Pahl, "Containerization and the paas cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015.

[45] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[46] R. Duan, R. Prodan, and X. Li, "Multi-objective game theoretic schedulingof bag-of-tasks workflows on hybrid clouds," *IEEE transactions on cloud computing*, vol. 2, no. 1, pp. 29–42, 2014.

[47] R. Kaewpuang, S. Chaisiri, D. Niyato, B.-S. Lee, and P. Wang, "Cooperative virtual machine management in smart grid environment," *IEEE Transactions on Services Computing*, vol. 7, no. 4, pp. 545–560, 2014.

[48] W. Saad, Z. Han, M. Debbah, A. Hjorungnes, and T. Basar, "Coalitional game theory for communication networks," *IEEE Signal Processing Magazine*, vol. 26, no. 5, pp. 77–97, 2009.

[49] B. Adrian and L. Heryawan, "Analysis of k-means algorithm for vm allocation in cloud computing," in *Data and Software Engineering (ICoDSE), 2015 International Conference on*. IEEE, 2015, pp. 48–53.

[50] D. Garg, P. Gohil, and K. Trivedi, "Modified fuzzy k-mean clustering using mapreduce in hadoop and cloud," in *Electrical, Computer and Communication Technologies (ICECCT), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1–5.

[51] E. E. Mon, M. M. Thein, and M. T. Aung, "Clustering based on task dependency for data-intensive workflow scheduling optimization," in *Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS), 2016 9th Workshop on*. IEEE, 2016, pp. 20–25.

[52] L. T. Thai, B. Varghese, and A. D. Barker, "Algorithms for optimising heterogeneous cloud virtual machine clusters," in *8th IEEE International Conference on Cloud Computing Technology and Science*. IEEE, 2016.

[53] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Containercloudsim: An environment for modeling and simulation of containers in cloud data centers," *Software: Practice and Experience*, 2016.

[54] K. Park and V. S. Pai, "Comon: a mostly-scalable monitoring system for planetlab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 65–74, 2006.

[55] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[56] M. Blackburn, "Five ways to reduce data center server power consumption. 2008," *The Green Grid*.

[57] E. Meriam and N. T. Mediatron, "Multiple qos priority based scheduling in cloud computing," in *Signal, Image, Video and Communications (ISIVC), International Symposium on*. IEEE, 2016, pp. 276–281.

Publications

SCI Publication

Kuljeet Kaur, Tanya Dhand, Neeraj Kumar, and Sherali Zeadally ”Container-as-a-Service at the Edge: Tradeoff between Energy Efficiency and Services Availability at Fog Nano Data Centers” in IEEE Wireless Communications.

Video Presentation Link

https://youtu.be/r42ZBWWfMoU