Cost Based Job Scheduling in Fog Computing

A DISSERTATION

SUBMITTED INPARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF DEGREE

OF

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by :

PANKHUDI SWAROOP

2K17/CSE/013

Under the supervision of

Prof. Rajni Jindal

(HOD CSE Department)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College Of engineering)

Bawana Road, Delhi-110042

JULY, 2019

DECLARATION

I (Pankhudi Swaroop), Roll No. 2K17/CSE/013 student of M.Tech (Computer Science Engineering), hereby declare that the project Dissertation titled ”Job Scheduling in Fog Computing” which is submitted by me to the Department of Computer Science & Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi **PANKHUDI SWAROOP**

Date: **(2K17/CSE/013)**

CERTIFICATE

This is to certify that the Major II Report entitled “Job Scheduling in Fog Computing" submitted by Pankhudi Swaroop (2K17/CSE/013) as the record of the work carried out by her, is accepted as the Post Graduate Project Work Report submission in partial fulfilment for the requirement of the award of degree of Master of Technology in Computer Science and Engineering in the Department of Computer Science and Engineering at Delhi Technological University, Delhi during the academic year 2018-19 .

Place: DTU-Delhi

Date: **Dr. Rajni Jindal**

(HOD CSE Department)

ACKNOWLEDGEMENT

The success of a Major II project requires help and contribution from numerous individuals and the organization. Writing the report of this project work gives me an opportunity to express my gratitude to everyone who has helped in shaping up the outcome of the project. I express my heartfelt gratitude to my project guide Dr. Rajni Jindal for giving me an opportunity to do my major I project work under her guidance. Her constant support and encouragement has made me realize that it is the process of learning which weighs more than the end result. I am highly indebted to the panel faculties during all the progress evaluations for their guidance, constant supervision and for motivating me to complete my work. They helped me throughout by giving new ideas, providing necessary information and pushing me forward to complete the work.

I also reveal my thanks to all my classmates and my family for constant support

Pankhudi Swaroop

ABSTRACT

Fog Computing is coming up as a big revolution in the field of distributed computing by making available the resources and services nearer to the edge devices. It acts as an intermediary layer between the client and cloud server and aims at reducing the load on the cloud servers by dealing with the client's requests nearer to the edge devices. It has proven to be successful in solving latency issues of high priority and sensitive tasks by processing and analyzing the data rapidly and efficiently. The main issue is not data rather how to manage, store, access and utilize this data so as to ensure efficient and quick responsive overall system. This is where fog computing comes into action. The main objective of fog computing is to make the same system more competent by reducing the interaction with the cloud server for data storage and retrieval. In order to achieve this goal, in fog computing, data is tried and kept closer to the end user. Fog Servers (FS) are set nearer to the edge devices. These servers are equipped with the capability to store, process and analyze data. The client, instead of sending his request all the way to the centralized Cloud Server (CS) now sends it to its nearest fog server thus reducing the overall computation time and latency. For fog computing to work successfully, it is important to deal with the problem of Job Scheduling. In my work I have proposed a dynamic Job Scheduling algorithm that will aim at maintaining the cost factor as minimum as possible, along with completing the task within the given deadline. The results of the proposed algorithm are compared with some of the famous Job Scheduling algorithms and results are represented graphically.

Keywords: Fog Computing, Cloud Computing, Job Scheduling, Fog Servers, Workflow, Cost Makespan, IoT

CONTENT

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Preamble

With the advent of IoT and ever-growing magnitude of data, both its storage and management became of the major concerns. The size of the data storage sites needed to be increased and at the same time it became a tedious task to retrieve and process those data. Speed became a major issue and as a result engineers had to come up with new and better technologies that could deal with the problem of latency at the same time be safe and secure. Cloud Computing became a famous technology as soon as IoT made its impact in the market. But along with all the advantages that cloud computing brought, it also had some setbacks like complete dependency on a single central server, latency issues etc. In order to solve these and many other problems, the concept of fog computing came into light.

CISCO gave light to the concept of fog computing in order to make execution of applications across systems more efficiently and at an increased speed by bringing the processing of data nearer to the edge devices. Fog computing makes use of the famous and ever-growing technology of

Internet of Things (IoT) to which the devices are connected. IoT makes connection and interaction between the edge devices, the Fog Server and Cloud Server easier and quick. Cisco IOX framework was developed so as to enable the users to develop, manage and run software applications on them. This encourages developers to come up with new ideas and develop applications that could run on this framework and exploit fog computing characteristics.

But before all this, we need to understand what led to the development of fog computing concepts, what is cloud computing and where all it failed to perform better. And then we would move forward to understanding fog computing and all its aspects.

1.2 Cloud Computing outlook

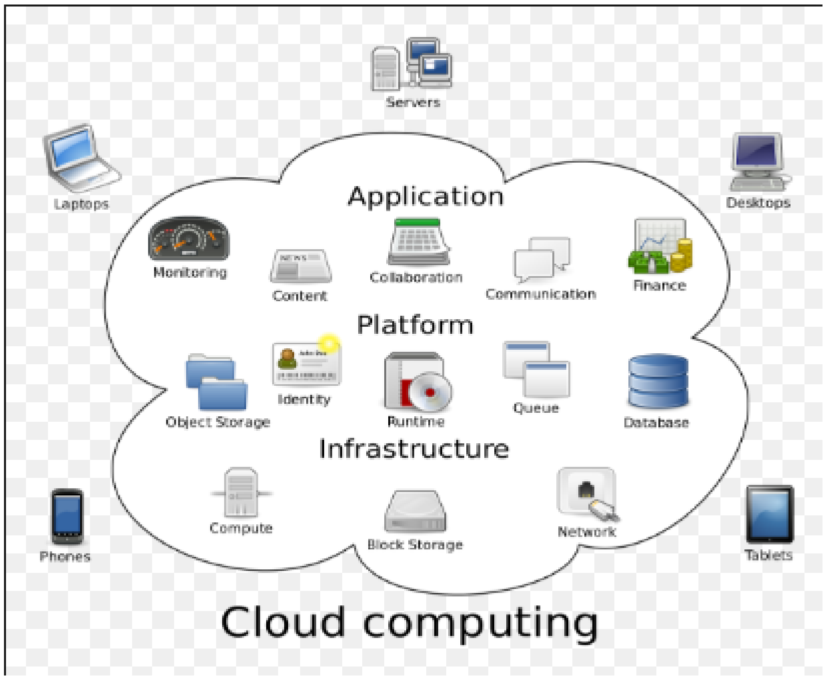


Figure1: Cloud Computing idea

Cloud Computing is an influential technology that has brought revolution in the field of computing and storage. It is a collection of shared resources and services that could be distributed all over the network for utilization. Main moto of this technology is to enable sharing of resources that maintains lucidity and economies of scale. It is basically a large interconnected mesh of computers that are all connected to a central server for sharing resources and services. This central server of the interconnected systems is responsible for maintaining consistency and coherence among the participating systems. It provides services to end users as per their requests and thus it becomes reasonable for the users to pay only for the services they need. It follows the pay-as-you-go policy of trade management. There are various pillars which all-together makes this technology beneficial, say load balancing, resource management, bandwidth operation, network interaction etc. Cloud computing gives freedom to its clients on the other end to employ whichever operating system or software that they wish to, the technology being provided to them remains unaffected by the local configuration of end devices.

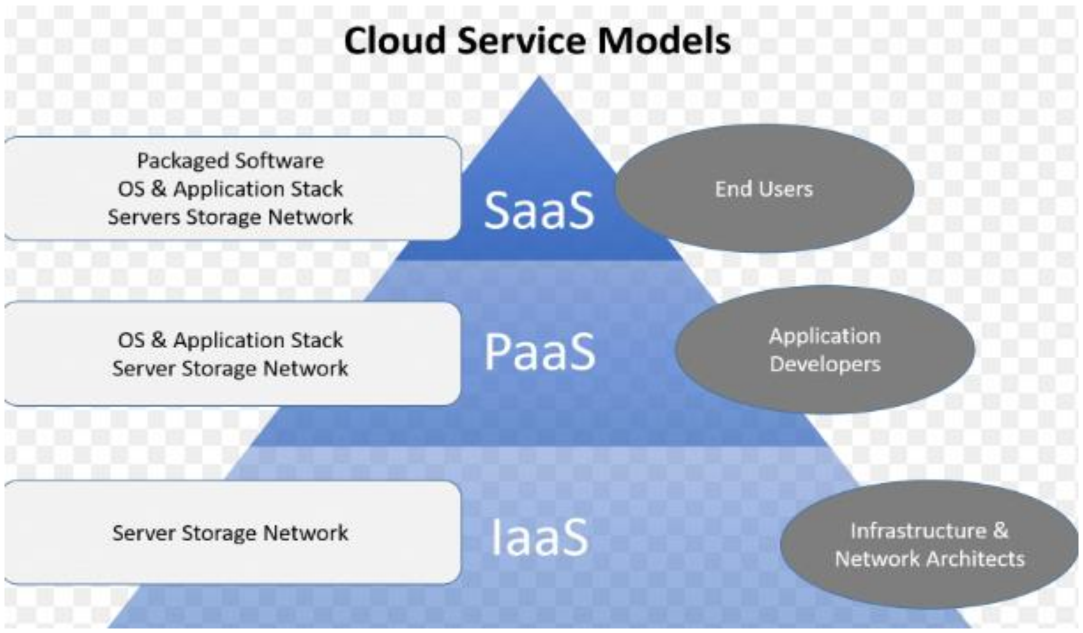


Figure2: Cloud Service Models

There are 3 basic models of cloud computing- SAAS (software as a service), PAAS (Platform as a service), IAAS (Infrastructure as a service). IAAS along with providing infrastructure such as virtual machines, it also provides resources like virtual machine disk library, block and file-based storage, firewalls, load balancer etc. It also acts as a basic layer in the cloud computing model. Few examples include: DigitalOcean, Amazon Web Services, Cisco Metapod etc. PAAS provides its client with computing platform that might include operating system, programming language execution environment, database, web server. Examples: AWS Elastic Beanstalk, Windows Azure, Heroku etc. SAAS saves the client from the headache of installation or maintenance of the software upon your system or coding of that software. It directly provides access to the software or application services installed at the server. Operation and access to the software is directly possible for the local system. No special kind of software or OS needs to be installed as all of that is taken care at the server end. Software maintenance and setup is done at the server end, we just need to pay for the services we undertake. For example: Google Apps, Microsoft Office365 etc. 4 deployment are famous in this, namely: Public Cloud which allows general clients to access the services provided by the hosting sites, Private Clouds are owned by some private organisations, Hybrid clouds are a mixture of both Public and Private clouds, Community Clouds are owned by organisations having common objectives.

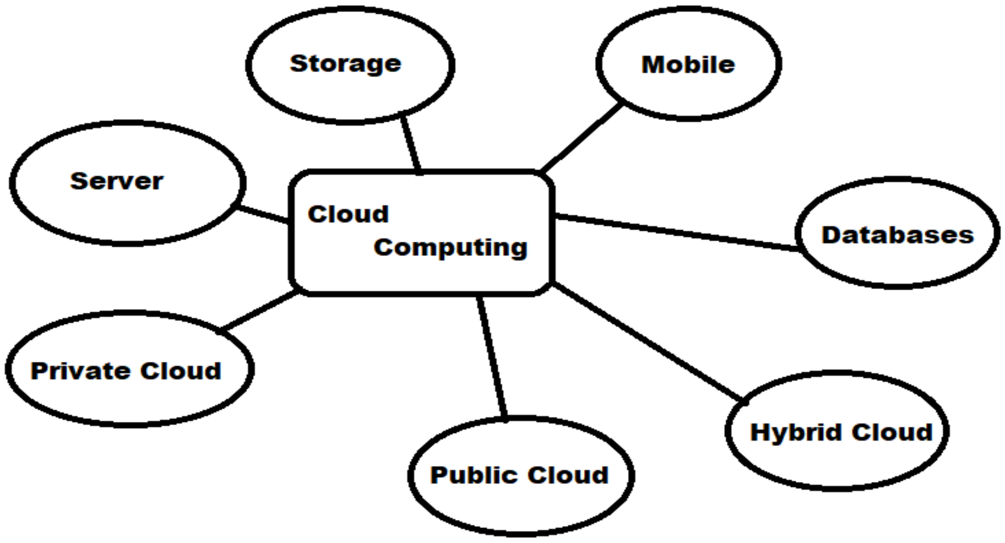


Figure3: Cloud Computing contribution

1.3 Need for Fog Computing

With all the advantages of Cloud computing keeping into consideration, there are also various issues related to it that cannot be overlooked. Since the data is located with the central server, every client node that wishes to access those data or resources situated at the central server need to first access the server and then the corresponding data is transferred on the client machine. Now, this leads to high latency in the execution of requests. Every time some data is required, client has to wait for the central server (located remotely) to provide them the access. Also, the resource with the central cloud server is also shared among various clients. This can sometimes lead to shortage of resources. The required resources might not be available with the client when requested. This also leads to a loss in throughput and efficiency of the system. Along with this, cloud computing requires high bandwidth requirement for the data to move from the client all the way to the server and then back again. This is done for every single request that the client makes. Since, the processing is done on a remote computer and managed by a central server, it might lead to privacy and security concerns. Cloud Computing tends to be inefficient with certain data applications. As a solution the concept of Fog Computing came into existence. Information and computing requirements are increasing exponentially. Centralized servers and datacenters are no longer ideal when you consider IoT devices. The Fog Computing IoT shows the current state and everything on the Internet, growing concepts (IoE) and presenting a new type of network architecture that still closes the cloud and partly decentralizes storage and logic functions.

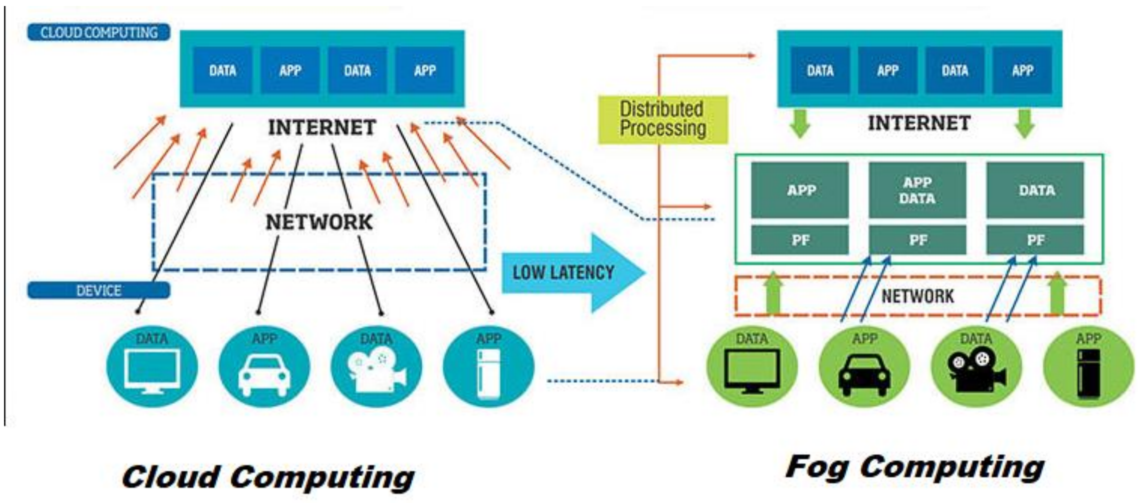


Figure4: Transition from Cloud to Fog Computing

Fog Computing is a thunder technology that has made computing easy, fast and laidback. It uses the edge devices to perform a major part of request execution, storage and communication. Small servers called fog servers are set nearer to the edge devices. These fog servers are capable of storage of data and information, computational abilities, resources to serve requests and communication between various clients and also among other fog servers. On November 19,2015 OpenFog Consortium was formed by some biggies like Cisco Systems, Dell, Intel, Microsoft etc. This was done with an intent to encourage developments and advancements in the field of fog computing. Since then various research works have been undergoing by different researchers in order to promote and enhance this technology so that it could efficiently solve the problems of cloud computing latency and resource shortage. Each Fog Servers have data stored upon them and capable of serving client’s request. The client sends a job request to its nearest fog server, if the corresponding fog server is capable of fulfilling the request, it deploys required resources for the job execution, performs calculations over the data and the corresponding results are forwarded to the client. This saves the system from the computational and communication latency that the system has to otherwise deal with in cloud computing case. The fog servers are now nearer to the client device hence latency is highly decreased. Fog Computing thus has brought computations and storage nearer to the user.

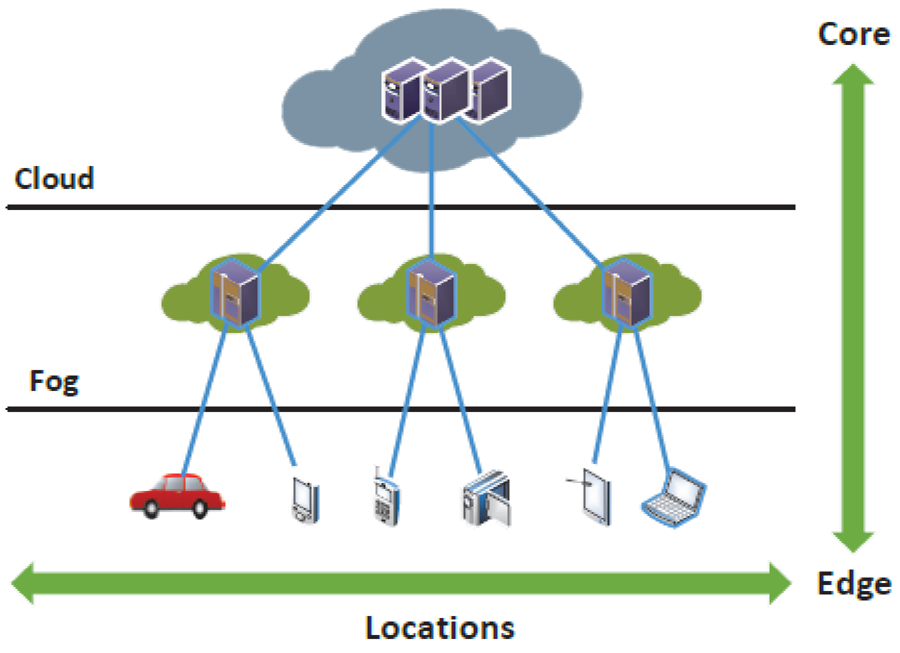


Figure5: Fog Computing outlook

Talking about all the benefits that fog computing has been providing.

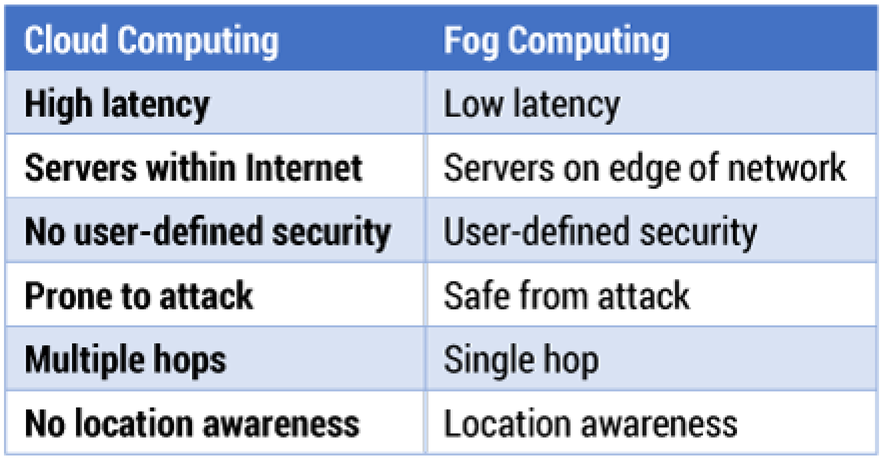
Privacy Control: Fog Computing allows the user to store, manage and process the data locally nearer to the edge device without the need of sending it to a far-a-way central server. That way the data is not shared among a large number of users and its confidentiality can be preserved. Also, the IT team can monitor, track, and control any device that collects or stores data since it is being held locally.

Increased Business productivity and Agility: Business productivity has increased since the inception of fog computing. Also speed of the business process has increased since the processing is now available near to the edge nodes. For example, only in the presence of anomalies, application data is flagged. Also, only those data that needs immediate client interaction or that are highly real time are retrieved through the processes employed in fog computing rather than bringing out all possible visible data related to the search query. This highly reduces the time and efforts required to detect possible issues. With the advancements in the Artificial Intelligence (AI) field which deals with a huge amount of data, fog computing can potentially lower the load that falls upon the project managers.

Data Security: Data Security is one of the major concerns in todays growing Internet use and information sharing. Privacy and Confidentiality of data is a must to maintain. A number of threats and attacks have grown at a huge rate. Fog computing allows connection and information sharing among various systems connected through a network. But instead of making information hub at a single point i.e. the centralized server, it is distributed all over the local end points as one central server might be more vulnerable to cyber-attacks and information theft. Fog Computing practice makes it possible for the threats to be identified at the lower levels and also it is not allowed to seep into the higher levels of network, infecting other devices.

Low Latency and location awareness: Fog Computing leads to reduced latency in information sharing since the data is processed neared to the edge devices. The round-trip time is considerably reduced. Also, the client is now aware about the location of the server where his/her data is being processed. It improves quality of service in real time applications by dealing with them with lesser response time and higher throughput.

1.4 Cloud Computing Vs. Fog Computing



Cloud Computing	Fog Computing
High latency	Low latency
Servers within Internet	Servers on edge of network
No user-defined security	User-defined security
Prone to attack	Safe from attack
Multiple hops	Single hop
No location awareness	Location awareness

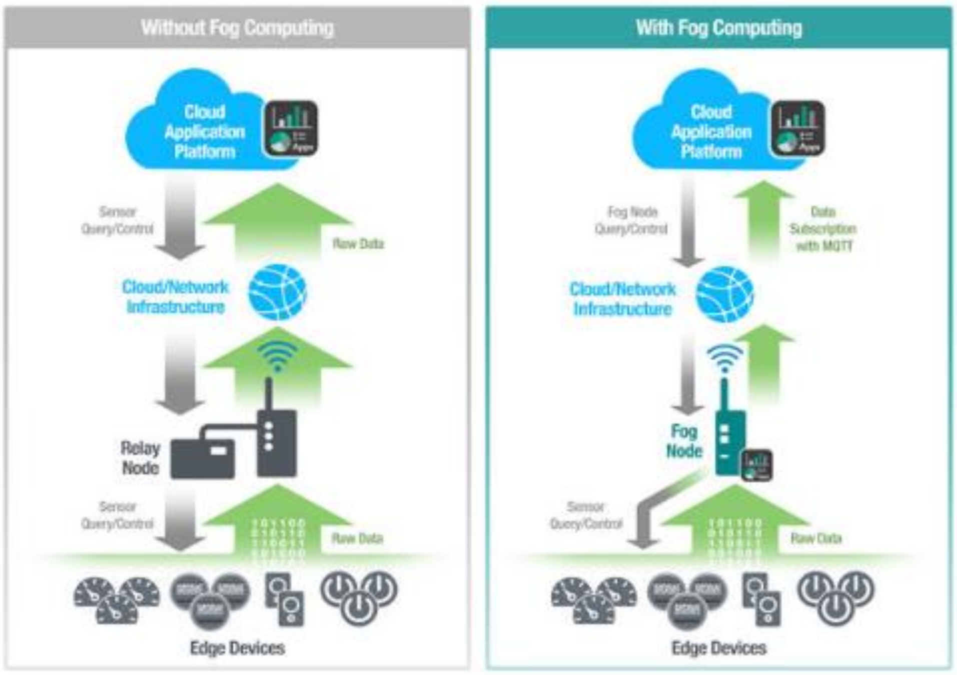


Figure6: Cloud Computing vs. Fog Computing

1.5 Architecture of Fog Computing

A three-level architecture has been designed for Fog Computing environment. Here, each edge device is attached to one of the Fog devices/ servers. These Fog devices are unified and each of them is connected to the cloud server. Fog Computing is a thunder technology that has made computing easy, fast and laidback. It uses the edge devices to perform a major part of request execution, storage and communication. Small servers called fog servers are set nearer to the edge devices. These fog servers are capable of storage of data and information, computational abilities, resources to serve requests and communication between various clients and also among other fog servers. On November 19,2015 OpenFog Consortium was formed by some biggies like Cisco Systems, Dell, Intel, Microsoft etc. This was done with an intent to encourage developments and advancements in the field of fog computing. Since then various research works have been undergoing by different researchers in order to promote and enhance this technology so that it could efficiently solve the problems of cloud computing latency and resource shortage. Each Fog

Servers have data stored upon them and capable of serving client's request. The client sends a job request to its nearest fog server, if the corresponding fog server is capable of fulfilling the request, it deploys required resources for the job execution, performs calculations over the data and the corresponding results are forwarded to the client. This saves the system from the computational and communication latency that the system has to otherwise deal with in cloud computing case. The fog servers are now nearer to the client device hence latency is highly decreased. Fog Computing thus has brought computations and storage nearer to the user.

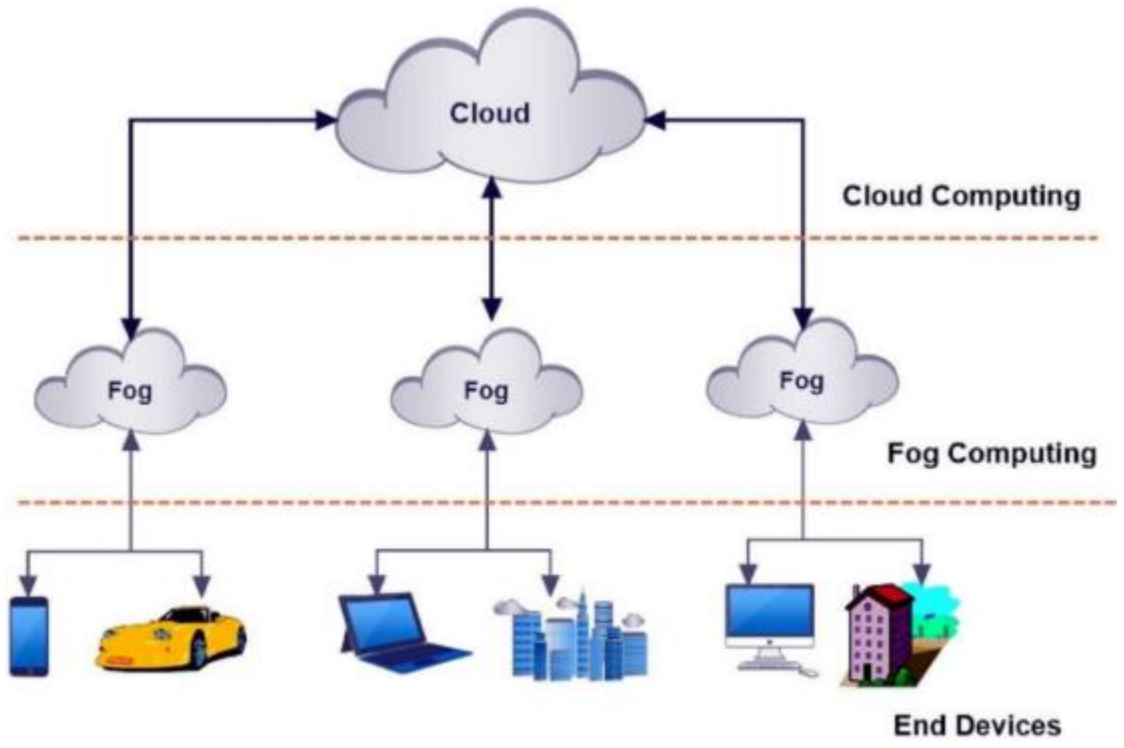


Figure7: Architecture outlook

First layer comprises of edge devices. These devices might be a part of some network or
might be standalone. These are the client devices from where the requests are sent and received. Second layer is the fog layer. It consists of a number of Fog Servers (FS) and Fog Server Managers (FSM). These FSs could be any devices like router, servers, switches etc. FSMs are responsible for managing the working of various FSs those are under them.

Cloud layer forms the third layer. It consists of a Cloud Server (CS). Whenever, the requests that are sent to the FSs, couldn't be completed, they are forwarded over to the CS. The CS then finally processes the data and sends the result to the edge devices.

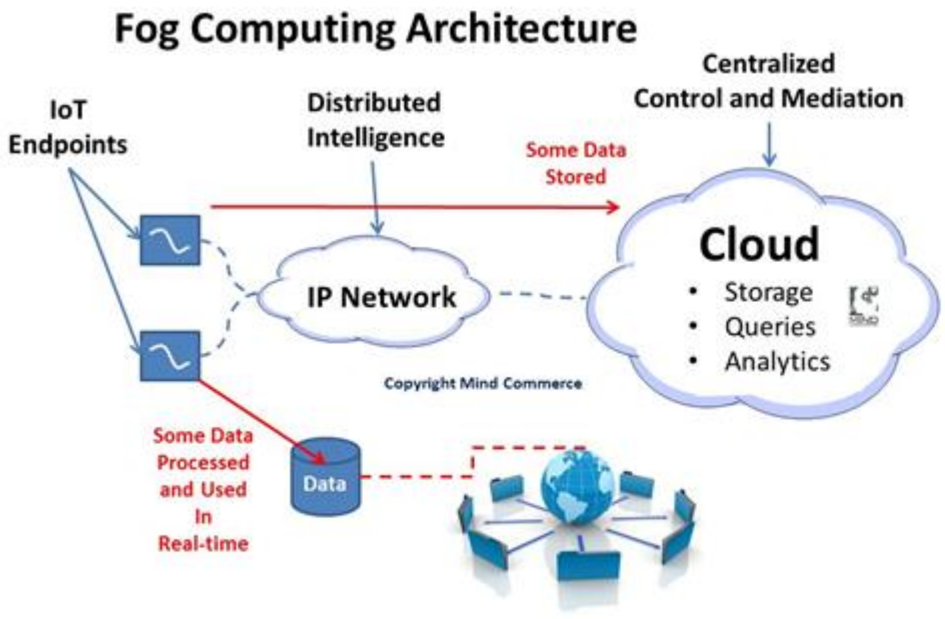


Figure8: Fog Computing Architecture

The client using the edge devices forwards his/her request to the nearest FS. The FS then checks if it has all the resources available to fulfil the request. If yes, the request is processed and results are sent over to the edge device that had been requesting. Else, if FS does not find itself capable of fulfilling the request, FSM handling the FS comes into action. It contacts with other FSs under it or other FSMs managing various other FSs. Now, if the request could be fulfilled at the fog layer, the results after processing is forwarded to the client. Else, the request is forwarded to the CS. The CS then works on the request, processes it and sends back the result to the requesting client through the corresponding FS.

1.5.1 Hardware Architecture

With the growth of Fog Computing, it has become an issue of utmost important to provide hardware architecture that would make working with Fog Computing easy and efficient. Since, Fog Computing works in a distributed fashion, ease of use is an important factor that should be taken care of. ARM systems have been introduced widely. They are both efficient and easy on pocket.

- Power consumption increases with Fog Computing hence the hardware has a power saving and management unit.
- Cooling units are provided with the Fog nodes and Fog Servers to deal with the problem of overheating.
- Storage subsystems are provided with highest possible storing capacity.
- In order to ensure parallel processing quick response, co-processors are installed at each Fog Servers.

1.5.2 Software Architecture

Bonomi et. al suggested a software architecture outlook for Fog Computing in which Two layers were proposed for the system.

- Abstraction Layer
- Orchestration Layer

Abstraction Layer maps Openstack which virtualizes heterogenous resources with fog structure. Openstacks run preferably on ARM and ADM.

Orchestration Layer is the one that plans the execution of all submitted tasks. The tasks are analysed against deadline, cost and other QOS factors, schedule is prepared and on the basis of this schedule execution manager processes the tasks.

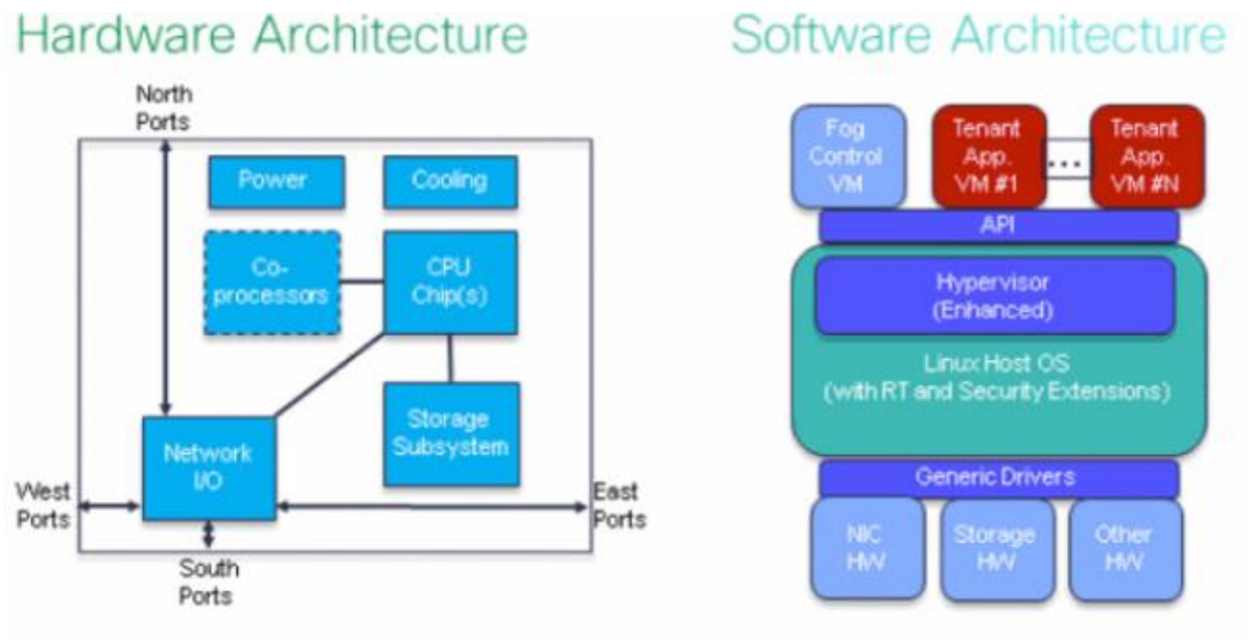


Figure9: Hardware and Software Architecture for Fog Computing

1.6 Fog Computing System Working

Following are the basic steps in which a problem that comes to a Fog Server is solved.

1. Task submitted to the Fog Server is broken into chunks.
2. These chunks are then assigned to the assigned Fog Nodes which will be working towards managing task completion.
3. Before any execution decisions are taken, these chunks are queued for scheduling process.
4. Based on scheduling algorithm, chunks who have been allocated nodes are sent up on the channels for transmission. All other chunks wait for their turn.
5. Once these chunks reach the processing Fog Nodes, they are sorted in the order of their finish time.
6. After processing, these chunks return back host site.
7. At host site, these chunks are reunited.

1.7 Applications of Fog Computing

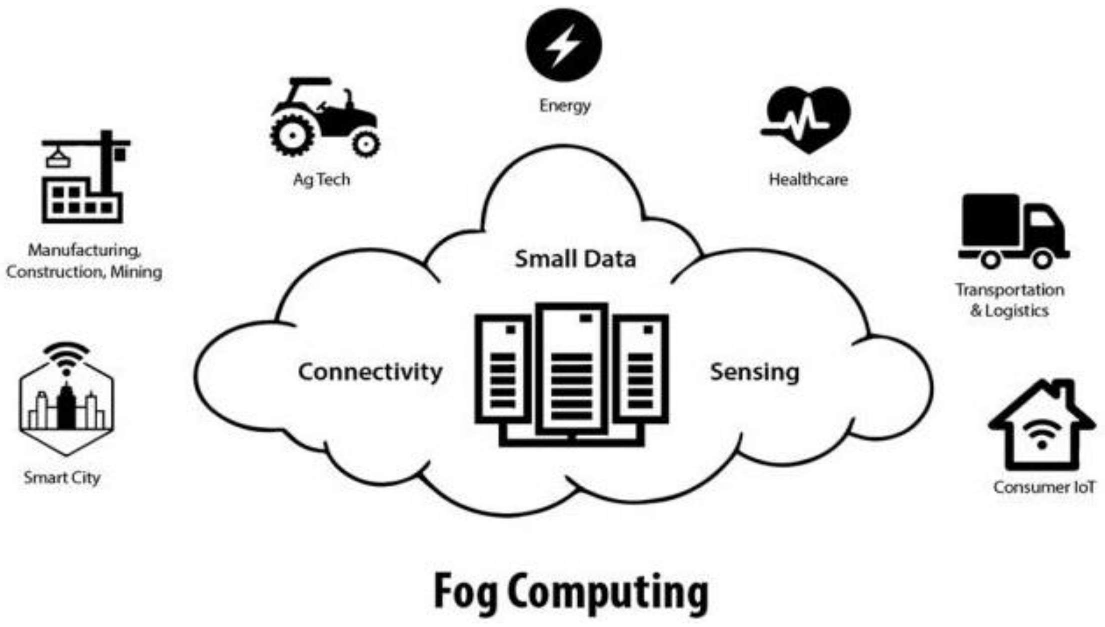


Figure10: Applications of Fog Computing

Smart Grid: Energy load balancing applications are running successfully on edge devices like, smart meters and micro grids. These devices have sensors and actuators attached to them. These sensors keep measuring real-time data upon which analysis is done. Based on this analysis, keeping in mind the energy demand, availability and cost incurred, the devices ca switch to alternatives like wind and solar energy. Figure 11 shows the working of smart grids. The fog collectors working at the Fog Layer, collects data from the edge devices like sensors, analysis is done and commands are sent to the actuators. They select out data that are to be processed locally at the Fog Servers and send the rest of the data to higher layers for visualization, real-time reports and advanced transactional analysis. Fog allows transient storage at the lowest tier to semi-momentary storage at the highest tier. Global reporting is provided by the Fog with business intelligence analytics.

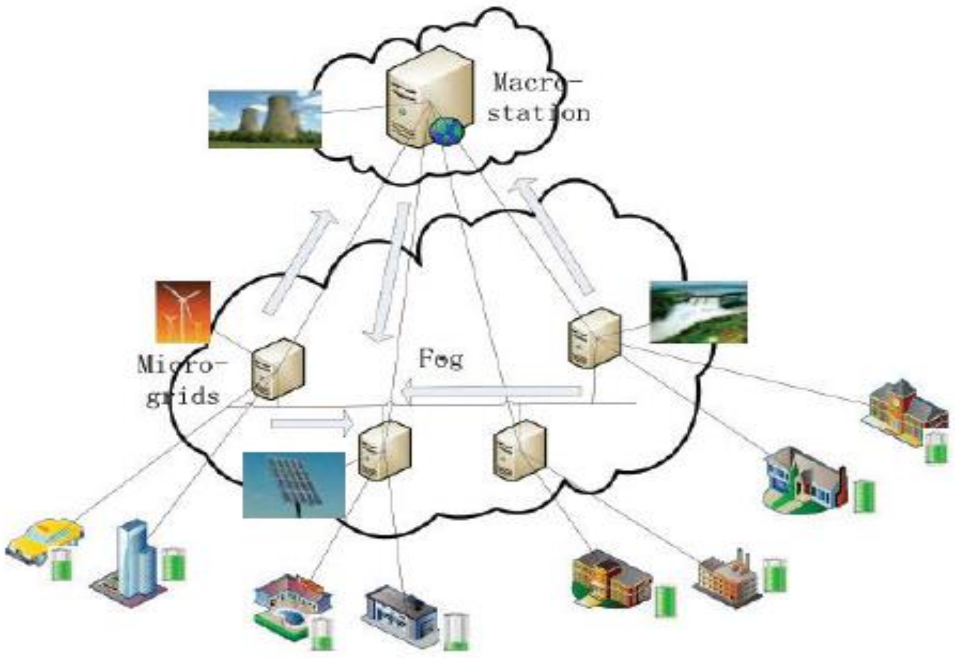


Figure11: Fog Computing- Smart Grid

Smart Traffic Lights and Connected Vehicles: Video cameras installed at the traffic lights have sensors connected through them. Whenever these cameras capture sensitive vehicles like ambulance or police vans, the lights of that particular lane is turned green. Smart street lights intermingle with the local sensors and analyse the current traffic, pedestrians, vehicles etc. Speed of the approaching vehicles are measured and also its distance from the zebra crossing. As soon as this analysis is done, intelligent lights are turned on, allows the sensitive vehicles to pass through and switches-off once traffic subsides. Smart lights in the vicinity act as Fog devices, they have the capacity to alter green traffic lights and they send alert signals to other traffic lanes making them aware about the sudden change in the lights. Wireless access points are established along the roads. These access points could be Wi-Fi, 3G, road-side units and smart traffic lights. Interaction between Vehicles to vehicles, vehicles to access points and access points to access points makes this system even more reliable and efficient.

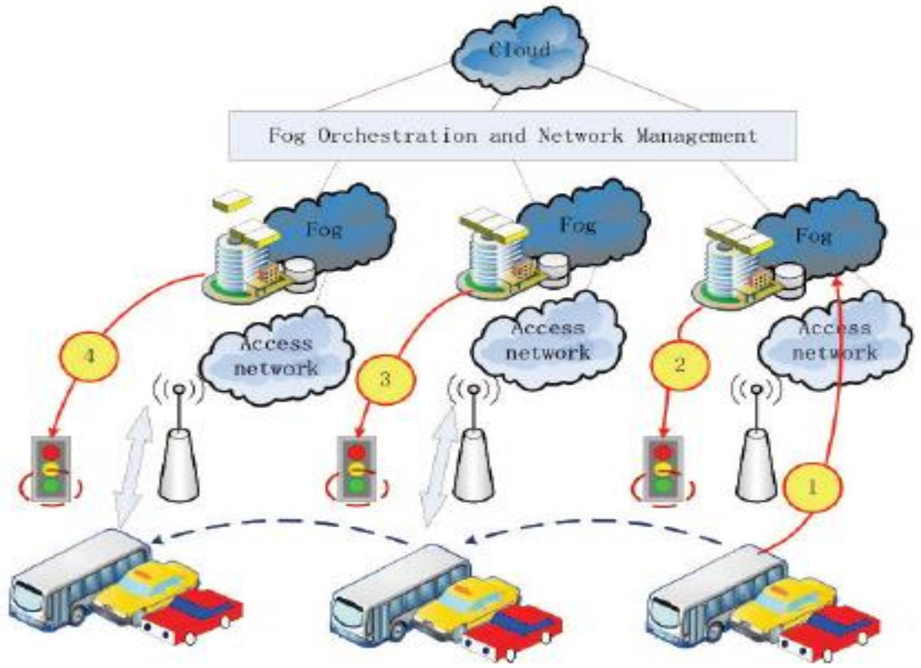


Figure12: Fog Computing- Smart Lights

Wireless sensors and Actuator networks: With the traditional wireless sensor networks there arise a problem with applications that includes more than sensing and tracking. The role of actuators in performing physical actions like opening, closing or even carrying sensors are also not so efficient with traditional methods. With the advancement in fog computing, the actuators can serve as a Fog device and control the measurement process, steadiness and oscillatory behaviours by establishing a closed loop system. For example, the sensors acting as fog device could be attached to the ball-bearings of the trains. These sensors could measure heat and temperature of the train and can alert the train operator to stop the train at next station and avoid potential derailment. In lifesaving airwind systems, the sensors keep measuring the in and out air flow through the mines and alert the automatically alter air flow if the conditions are deadly to the miners.

Decentralised smart building control: With this application of fog computing one can measure temperature, pressure and humidity inside a building or levels of various gases as well. Here the data of various sensors within a floor, are exchanged and readings from them could be united to give away reliable results. They use combined results from all the fog devices to react to a particular situation formed. As a result of the analysis done, the smart building could open windows to level air pressure, inoculate fresh air or lower the temperature to resolve issues. Air conditioners could level the humidity rate of air by either removing away moisture from it or increasing the humidity. Smart Lights could be turned on or off as a reaction to movements in the building. Fog devices installed at each floor could work together to achieve higher levels of actuation. With this application, smart building could preserve energy, water and other resources by working smartly using sensors.

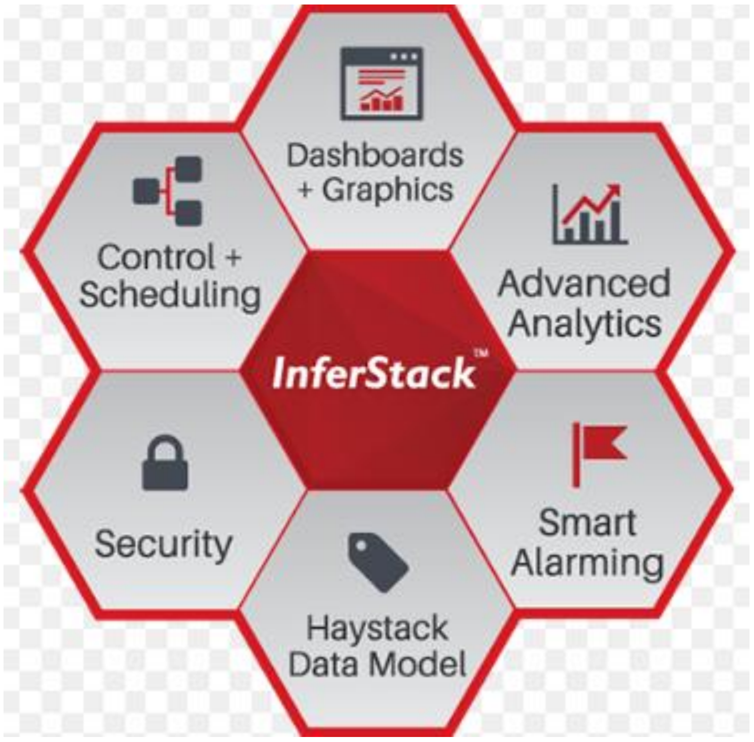


Figure13: Smart Buildings

IoT and Cyber physical Systems (CPSs): Fog Computing is playing a major role in IoT services and CPS. IoT is a network that connects a number of ordinary edge devices with an unique address. CPS includes a tight mixture of the computational and physical elements of the system. CPS also manages the interaction and communication between computers and central, physical and engineered systems. IoT and CPS have the potential to bring large scale advancements through the intermingle of computer-based controls and communication systems, engineered systems and reality base. Fog Computing takes help of embedded systems in which software and edge device systems programs are embedded in devices for more than just computation. Main goal is to combine the importance of abstraction and accuracy of software programs with dynamics and uncertainty of physical reality. Using this scenario, we can build better and intelligent medical devices and systems, smart highways and buildings, advanced robotics systems.

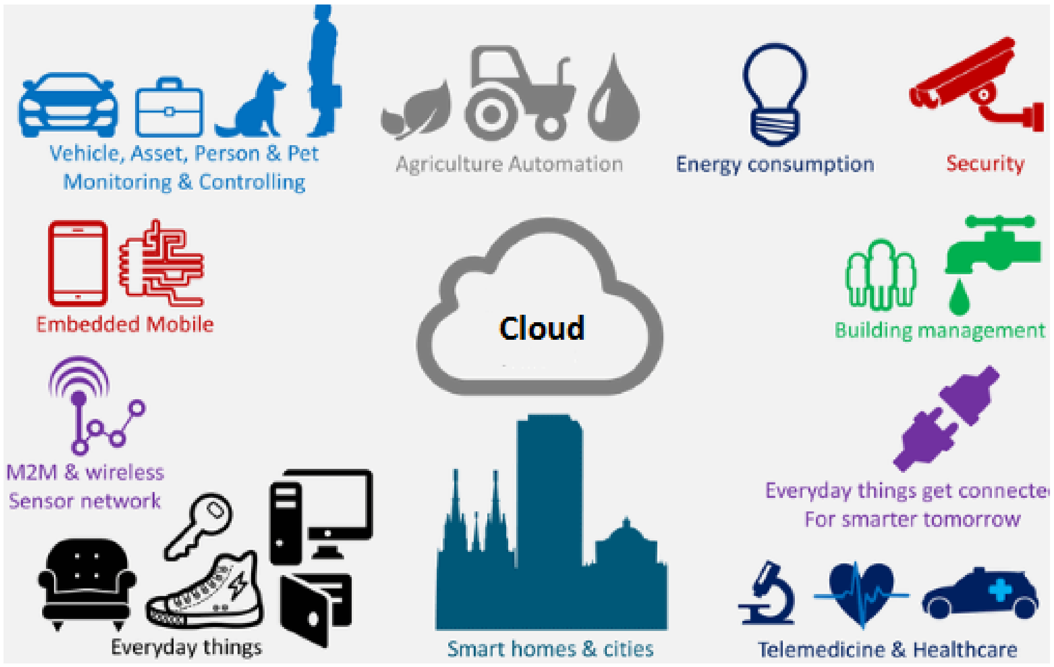


Fig14: Fog Computing enabled IoT

Software Defined Networks (SDN): Fog computing concept could be applied to SDN systems for vehicular networks. SDN is this new name in the market, a network paradigm and has become a raging topic of research in the IT industry. It has a separate control and data communication layer. Control is handled at the central server and nodes have to process and send data over the paths decided by the servers. There is also a distributed approach to implement the centralized server. SDN was first introduced in LAN, Wireless sensors an d mesh networks, but earlier they didn't involve multihop wireless communication and routing. Peer communication was also a problem. SDN when combined with Fog Computing can solve the issues of vehicular networks, recurrent connectivity, packet loss rate etc. by supplementing vehicle-vehicle with vehicle-infrastructure communications and centralized control.

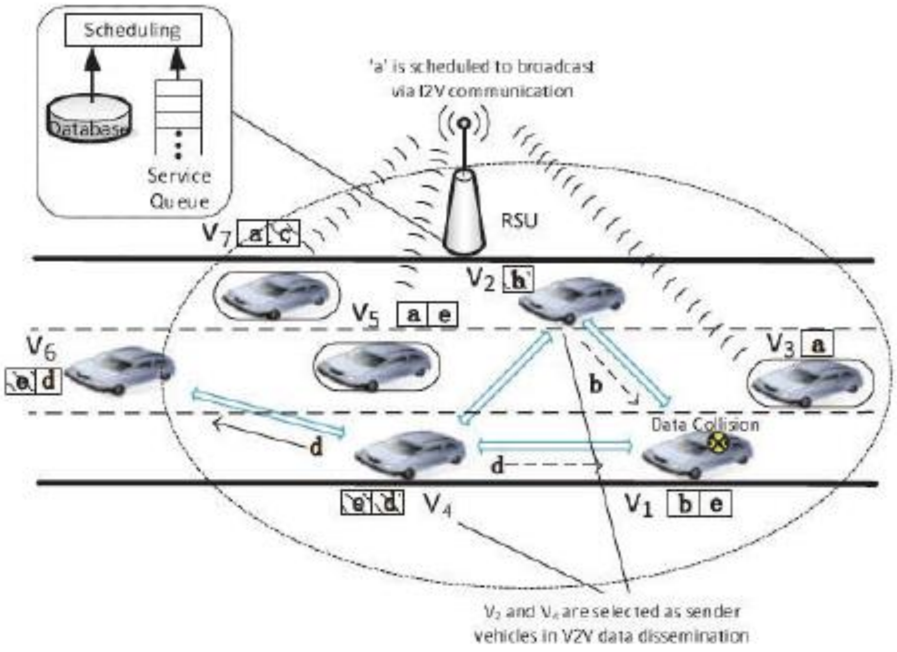

Scheduling
Database
Service Queue
'a' is scheduled to broadcast via I2V communication
RSU
V7 a c
V2 b
V5 a e
V6 e d
V3 a
b
d
Data Collision
d
V4 e d
V1 b e
V2 and V4 are selected as sender vehicles in V2V data dissemination

Figure15: Fog Computing- SDN vehicular network

1.8 Issues with Fog Computing

Talking about various issues that Fog computing has to deal with are:

- Security is one of the major concerns with Fog Computing. Moment the security algorithms fail, sensitive user data can get exposed to hackers and malicious agents. Other security threats might include IP Address Spoofing, wireless network security, man-in-the-middle etc.

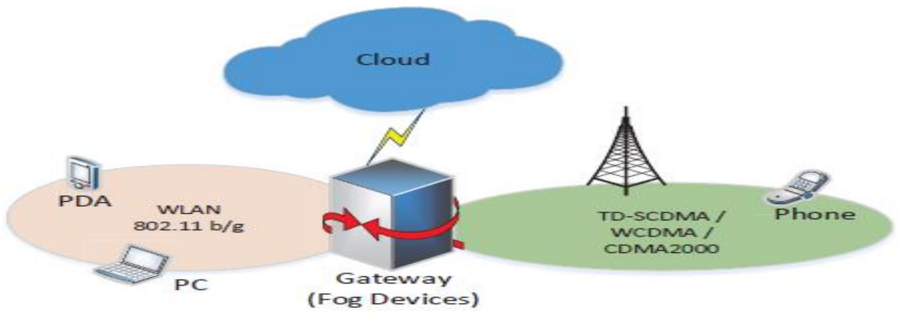


Figure16: Man-in-the-middle attack in Fog

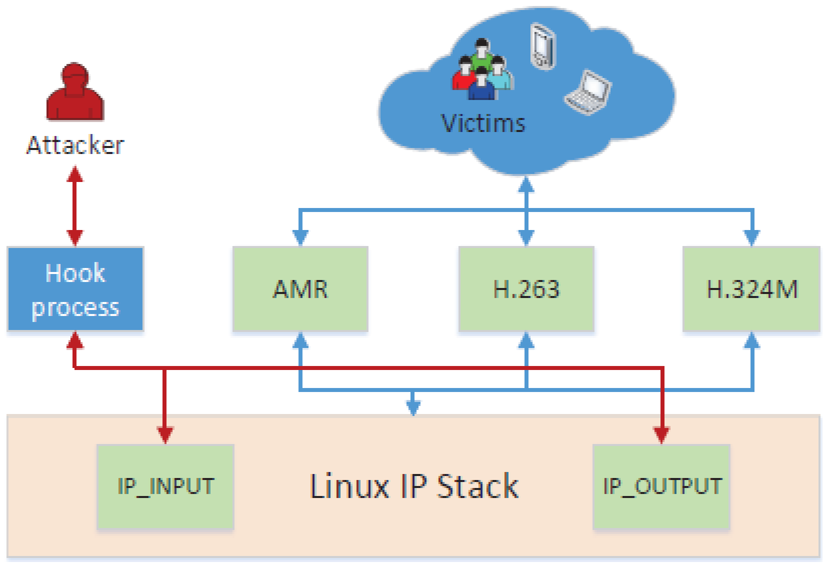


Figure17: Man-in-the-middle attack system design

- Since, data is distributed and replicated (in some cases) over various servers, maintain data consistency becomes a bigger challenge with fog computing.
- Although fog computing can provide greater processing speed and high amount of data storage but managing this bulk of data is a big problem.
- Authentication of valid users, data confidentiality and trust schemes are a major concern
- Since, there is a shuffling of data and request between edge devices, Fog Servers and Cloud Server, scheduling this data effectively so as to increase the throughput and efficiency of the system is also a challenge.
- Power consumption increases highly in fog computing environment rather than in cloud computing scenario.

1.9 Job Scheduling in Fog Computing

The above discussed advantages and issues, command the development of advanced resource scheduling algorithms custom-made for a Fog environment which may generate cost and performance operative solutions for scientific workflow execution. Job Scheduling refers to allocating resources required to complete the user task in such a way that rquest gets completed within defined constraints. These constraints could be time, cost etc. The system handles prioritized requests, high priority jobs or hard deadline jobs are given greater importance. In order to assure smooth functioning of the system and on-time completion of the workflow, job scheduling needs to be handled wisely. An efficient job scheduling algorithm can guarantee efficient concurrent processing of independent tasks of a workflow. A good job scheduling technique in fog computing becomes even more important because here the data has to flow between edge devices, Fog Servers and Cloud Server. A scheduling algorithm involves two important decision to be made-

- Determine tasks that could be executed in parallel
- Determine where to execute theses parallel tasks.

This decision is either based on some pre-defined values or dynamic data attained during task execution.
There are two main stages involved with Fog Computing scenario:

- Resource Provisioning phase that detects, sorts and provisions resources required to complete the request.
- Task Mapping phase in which a suitable server/processor is identified and the task is mapped to that very server. This task is computed based on the schedule created for the workflow using specific algorithm.

These decisions at each step are taken keeping in mind the cost constraint and deadline assigned by the user. Most of the previous that will be discussed in the coming segments, focused only upon the deadline constrained. Their main aim was to reduce the total time taken for execution of the workflow. The algorithm proposed by me not only keeps in mind the deadline issue but also tries to reduce the cost factor as much as possible. The tradeoff between time and cost must be kept in check.

1.9.1 Difficulties in devising a Scheduling Strategy

The design of the scheduling algorithm will depend on the following constraints:

1. *Cost of tasks*: Points that are to be considered here are how much information is available regarding the cost of the tasks employed? Are all of them same? When can this be cost be known during execution?
2. *Dependencies between tasks*: How many dependent and independent tasks are there? What are the child processes to every parent process? When can this relation be made available to the algorithm?
3. *Locality*: Where can dependent and independent tasks be executed to reduce the overall cost? How can communication cost be minimized? How to know about the communication requirements?

1.9.2 Scheduling solutions

Static Scheduling: The main idea about static job scheduling algorithms is that the decision is taken at compilation time. Using some static analysis methods, the size of the workflow is determined. Obtaining this information at compilation time is hard to obtain and usually incomplete.
In the later stage a static mapping on the parallel architecture of the search tree is done. Although, an optimal mapping is NP- complete.
A Directed graph is built for the workflow. Nodes representing the tasks and links representing dependencies. Communication order is then calculated for the tasks.

Dynamic Scheduling: This is also known as adaptive work sharing practice. It uses the computational information about the workflow state at a particular instant during program execution to make decisions.

Dynamic scheduling is a better approach as a large section of problems have a solution space that could be obtained using a search tree. Now these problems are computationally demanding, require parallelization strategy and dynamic load balancing.

1.10 Related Work

In heterogeneous environments, despite numerous efforts, task scheduling still remains a big challenge. As usual presentation, each application is comprised of multiple interdependent tasks and each of which is specified by an amount of processing works. It can be modeled as a Directed Acyclic Graph (DAG), in which vertices represent application tasks and edges represent inter-task data dependencies. The primary goal of task scheduling is to schedule tasks on processors and minimize the makespan of the schedule, which has been shown to be NP-complete problem. The most common task scheduling algorithms are list-scheduling heuristics. For example, the Earliest Time First (ETF) algorithm [2] computes, at each step, the earliest start times of each tasks on all processors and then selects the one with the smallest start time. The Dynamic Level Scheduling (DLS) algorithm [3] selects the task-processor pair that maximizes the value of the dynamic level (DL), which is the different between the static level of a task and its earliest start time on a processor. Meanwhile, the heterogeneous earliest-finish-time (HEFT) algorithm [4] selects the tasks with the highest upward rank and then assigns it to the processor that minimizes its earliest finish time.

However, how to achieve good tradeoff value between the makespan and the monetary cost is not considered in these algorithms. minimize the makespan of the schedule, which has been shown to be NP-complete problem. The most common task scheduling algorithms are list-scheduling heuristics. For example, the Earliest Time First (ETF) algorithm [2] computes, at each step, the earliest start times of each tasks on all processors and then selects the one with the smallest start time. The Dynamic Level Scheduling (DLS) algorithm [3] selects the task-processor pair that maximizes the value of the dynamic level (DL), which is the different between the static level of a task and its earliest start time on a processor. Meanwhile, the heterogeneous earliest-finish-time (HEFT) algorithm [4] selects the tasks with the highest upward rank and then assigns it to the processor that minimizes its earliest finish time.

However, how to achieve good tradeoff value between the makespan and the monetary cost is not considered in these algorithms. For a large-scale environment, e.g. cloud computing system, there had been also numerous scheduling approaches proposed with the goal achieving both the better application execution and cost saving for cloud resources. Bossche et al. [5] introduce a cost-oriented scheduling algorithm to select the most proper system (private or public cloud) for executing the incoming workflows based on the ability of meeting the deadline of each workflow and cost savings. The budget constraints for using the cloud resource are considered in ScaleStar [6], whose task assignment is based on a novel objective function Comparative Advantage (CA). This algorithm achieves good balance between cost savings and schedule length; however, the high complexity of CA hinders the algorithm to be applied to the large-scale workflows.
For the purpose of judging the performance of the proposed algorithm, it is being compared with two other famous job scheduling techniques are being considered.

According to Xuan-Qui Pham et al., they consider the workflow as a Directed Acyclic Graph (DAG) where the nodes represent tasks of the workflow and the edges represent communication links and interdependencies between the tasks. The algorithm has two main phases:

1. *Task Priority Determination:* Priority of all the submitted tasks are calculated. The order of execution largely depends upon this priority value of each tasks. The tasks are ordered on the basis of their scheduling priorities that are based on the upward ranking. The upward ranking described is basically the maximum length of the critical path from the node under consideration to the exit task, which also includes the computational time for task $v_{i.}$ The priority value for task $v_{i\,,}$ pri (v_i) is then calculated recursively. Also, average data transfer time between two tasks i and j is also calculated.

2. *Appropriate Node Selection for Task Execution:* Here, two most important parameters are set, Earliest Start Time (EST) and Earliest Finish Time (EFT). A task can not enter the running state until ll the inputs and resources are made available to it. Latest Start Time for the concerned task is calculated, i.e. maximum by what time will all the inputs be made available to the concerned task. Data Transfer time from each parent node to its child node is calculated based on which above define Earliest Start time and Earliest Finish Time is calculated.

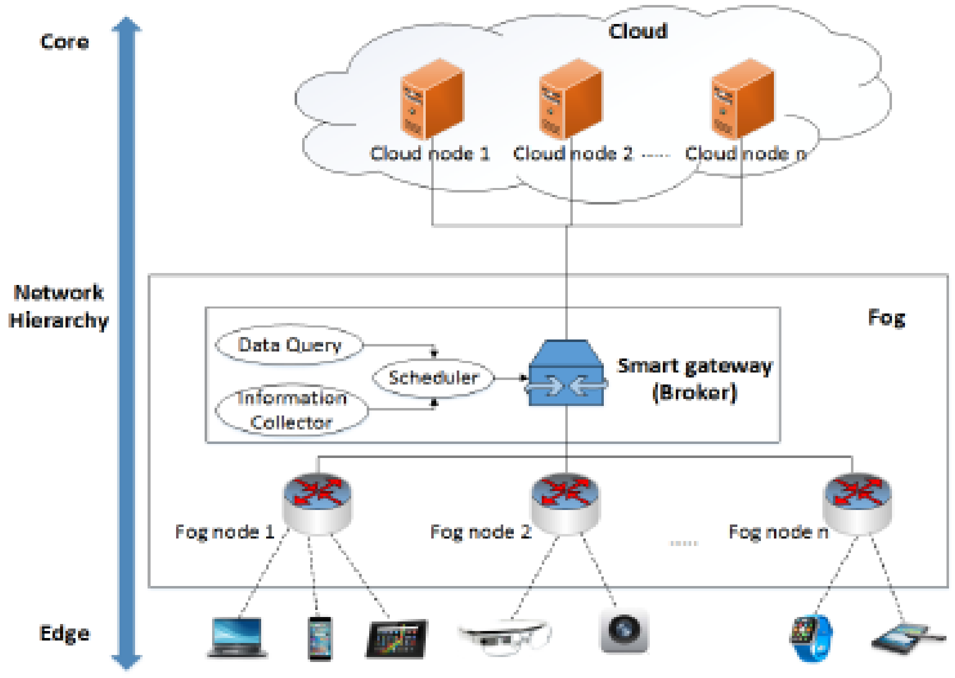


Figure18: Working Architecture

Total cost incurred in executing the workflow is calculated using various parameters and values. This algorithm shows a good tradeoff between cost and time. The algorithm proposed in this work provides better tradeoff between cost and time.

The results are also compared with Greedy Approach for time. In this approach, the algorithm has been designed with the aim of completing the workflow in the least time possible. For this, at each step the Virtual Machine (VM) with least execution time possible is selected no matter how much it might cost.

To implement this approach, Breadth First Approach is employed. Queue as a data structure is being used here. The workflow submitted to the system for execution is converted into a Directed Acyclic Graph where nodes represent tasks and edges represent communication links and inter-dependencies among tasks. Separate matrix information about execution time, data transfer time, cost per VM is also provided to the system. The algorithm picks up each task one by one, detects which virtual machine will give the least execution time possible. Then the VM on which the parent task of the under-consideration child task is running is picked and data transfer time + Minimum execution time is calculated. If this is less that execution time of the VM on which parent task is running, then the task is assigned to the minimum execution time VM. Else, it is assigned to the VM on which the parent task is running.

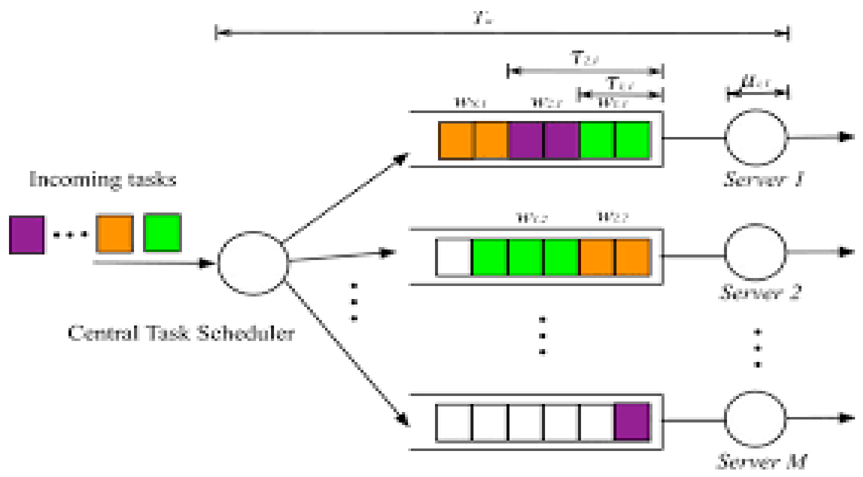


Figure19: Greedy for Cost Job Scheduling

CHAPTER 2

METHODOLOGY

2.1 Architecture for workflow execution in Fog Computing

This section explains the application model, architecture for Fog Computing implementation and overall architecture for computing framework.

2.1.1 Application Model

The Workflow presented in the work is converted into a Directed Acyclic Graph, G= {T,E} where T= {t_1, t_2, …, t_n} Tis a set of tasks that have been submitted under the workflow G. E is a set of edges between these task nodes. Now these edges represent communication and data links between different tasks and also interdependencies between them. An edge is represented as e_{ij} , This means it's an edge between task node i and task node j, pointing from node i to node j. Child node j is dependent on parent node i for communication and data links. This also represents an important factor that has to be kept in mind, only when the execution of all the parent tasks of a particular child task has completed execution, then only the child task can start its processing. i.e. once both data and control restrictions are fulfilled

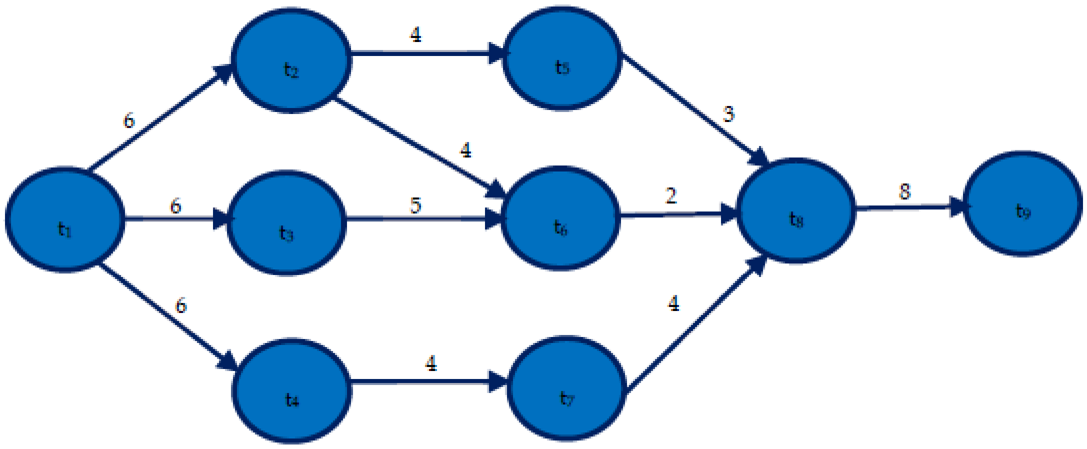


Figure 20: A sample workflow

Deadline D for task completion is defined as the maximum time within which the workflow must be completed. A sample has been given below in figure20. Nodes represent tasks and edges represent communication and data links.

2.1.2 Fog Resource Model

The Fog model considered in the given setup, works on the virtual resources provided by IAAS Fog service providers. The services provided are largely distributed into 2 categories: Computational services and Storage services. The Computational services provided basically includes all the tools and software made handy to the users to run their own applications. Examples include: Amazon Elastic Cloud Compute. Storage facilities provides its users with specific storage locations to store local input and output files. Examples include Amazon Elastic Bookstore.

The computation and storage processes are all done in the local vicinity of the concerned user. Now Virtual Machine (VMs) are the entities that provide computational support to the system. A number of virtual machines are a part of the whole module, each of which are different in terms of size, cost incurred, memory size, computation power, computation speed, CPU time etc. and hence a proper selection of these VMs is a major part of the implementation. Whenever VM is assigned with its first job, the system takes a certain amount of extra time to boot up. Similarly, when the VM shuts down, extra time is required to shut it off properly.

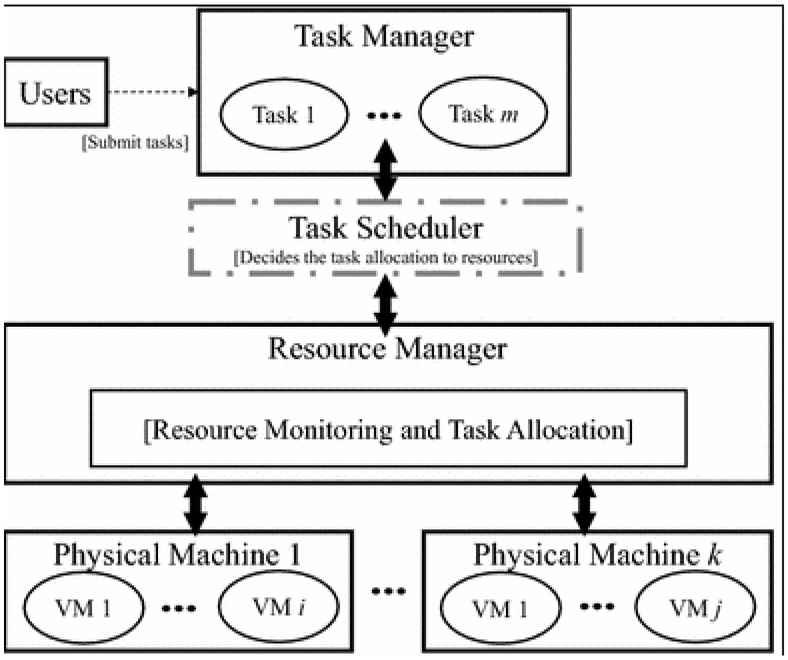


Fig21: Fog Resource Model

In the proposed model, storage cost isn't considered because that varies from service providers. Also, data transfer cost is not considered because that is free in most of the Fog environment. So, in the given setup it is considered zero. Another reason for not considering these costs is because they are independent of the scheduling algorithm used and hence won't affect the comparison result.

In the model, VM is considered as a collection of two tuples $\{(ETti)v, Cv\}$, where the first tuple represents the Execution Time for task t_i on VM v and the second tuple C_v represents the cost for VM v per interval. The cost of running a taskti on a VM of typeVMv, is calculated as $\lceil (ETti)v/time\ interval \rceil \times Cv$. Also, data transfer time between each task are given in the matrix. If both child and parent tasks have to run on the same VM, data transfer time becomes zero.

2.1.3 Computing Platform Model

The computing platform considered in this system is depicted in figure 22. The workflow is executed on the basis of the below model. In order to successfully execute the given tasks, working of the algorithm is divided into two important phases. In the first phase, all the required resources for task execution are decided. This is basically the provisioning phase where the resources are analyzed and provisioned to the tasks. In the second phase, workflow schedule is generated. This is done on the basis of conditions given in the algorithm. The resources are mapped onto tasks keeping in mind the proposed algorithm. Deadline for workflow completion has to be kept in mind always while deciding this schedule. Also reducing overall cost is the major aim of the proposed algorithm.

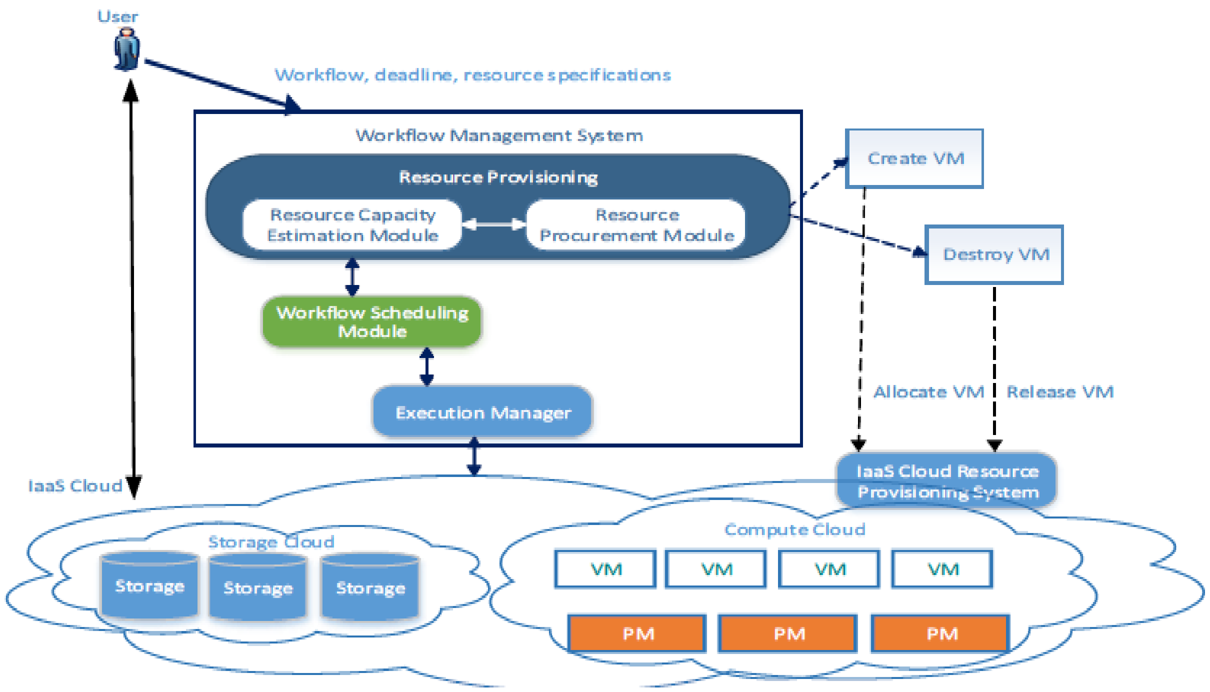

User
Workflow, deadline, resource specifications
Workflow Management System
Resource Provisioning
Resource Capacity Estimation Module
Resource Procurement Module
Workflow Scheduling Module
Execution Manager
Create VM
Destroy VM
Allocate VM
Release VM
IaaS Cloud Resource Provisioning System
IaaS Cloud
Storage Cloud
Storage
Storage
Storage
Compute Cloud
VM
VM
VM
VM
PM
PM
PM

Figure22: Computing platform model

A user submits a workflow along with the associated QoS requirements e.g. deadline constraint and resource specifications to the workflow management system (WMS). The deadline constraint specifies the time limit and the resource specifications describe the resource requirements (compute, memory, I/O) of the applications. Given these inputs, the WMS would automatically identify and provision the required resources, schedule tasks onto the provisioned resources and manage the workflow execution. In order to reduce the cost of running the application, the WMS acquires the resources as and when they are needed and releases them immediately after use. WMS consists of three main modules: *Resource Provisioning Module, Workflow Scheduling Module* and *Execution Manager*. Resource Provisioning module consists of two sub-modules: *Resource Capacity Estimation Module* and *Resource Procurement Module*. Resource Capacity Estimation module analyses the workflow structure to determine the amount of resources required. The Resource Procurement Module negotiates with the IaaS re-source provisioning system to acquire the identified amount of resources. The Workflow Scheduling Module, in coordination with the Execution Manager identifies the mapping between the provisioned resources and the tasks of the workflow. The scheduled tasks are then executed by the Execution Manager. The main difference between this computing model and other traditional high-performance computing models is that resource allocation is workflow driven and the resource set size may vary during runtime.

2.2 Problem Definition

Scheduling algorithms and provisioning heuristics might have varied objectives in general. But the algorithm proposed in this work has the main objective to follow Just-in-Time approach and finish the workflow within minimum cost possible. As a first step to proceed with the execution, it is made sure if the deadline provided by the client is achievable or not. If the tasks are so interrelated and data transfer and execution time are such, that in no way can the given deadline be met, algorithm declares that no feasible solution is possible for the workflow. The client then checks and changes the deadline. But if the deadline is achievable in the first place, main aim lies with the fact that the system has to create such a schedule that minimum cost is incurred on the overall execution of the workflow and also the deadline is met.

On this basis the algorithm can be seen as a combination of 2 steps:

1. *Selection of Cheapest possible VM for a task:* In this decision task, a VM that would be cheapest for the task waiting to be scheduled is decided and picked.
2. *Scheduling the workflow:* Based on the VM selected in the above step, resources are provisioned to it and accordingly schedule is decided as to when execute the selected VM.

It has to be noted that the data needs to be transferred from the parent task VM to child task VM before scheduling the child task for execution. Fog servers provide local and temporary storage of data hence, this transfer needs to be done within expiration limit of the particular VM. Also, not always it is possible that the schedule runs exactly on-time. Any delay in the start and end time of execution is defined by Actual Start time and Actual Finish time. This delay could be caused by performance variations. The difference between Actual and Expected times must be kept into notice. And according to this difference other unscheduled tasks are processed.

Thus, the main problem is to construct a schedule S for the Workflow given, such that user deadline is reached and total cost incurred in execution stands out to be minimum.

2.3 Proposed Scheduling Algorithm

The proposed algorithm works on the principle of Just-in-time (JIT) for Fog environment. It schedules various tasks of a workflow before the workflow starts executing. In order to assure on-time completion of tasks, performance variations are also considered. On the basis of these variations required delays are applied in the algorithm.

In order to deal with the acquisition delay, the algorithm takes the expected VM acquisition delay as input and accordingly decides the performance variations incurred. Also, termination delay hardly affects the workflow execution time to meet the deadline. Service providers have now considerably reduced prices for acquisition intervals. This altogether lessens the delay cost.

We need to first know about various calculative terms that are used in the algorithm, which is discovered further.

Various notations used in this work along with their meaning is given in the table below:

Table1: Notation

Symbol	Meaning
VMset={VM1,VM2,…VMm}	Set of all VM instances available
D	Deadline defined by the user for workflow execution
ET(ti,VMv)	Execution Time required by task ti when executed on VMv.
TT(e_{ij})	Data transfert time from task ti to task tj
MET(ti)	Minimum Execution Time required by task ti
{t_{entry}}	Task that is not dependent upon any other task, i.e. no parent
{t_{exit}}	Task with no child task
EST(ti)	Earliest Start Time for task ti
EFT(ti)	Earliest Finish Time for task ti
AST(ti)	Actual Start Time for task ti
AFT(ti)	Actual Finish Time for task ti
XST(ti)	Expected Start Time for task ti
XFT(ti)	Expected Finish Time for task ti
LST(ti)	Latest Start Time for task ti
LFT(ti)	Latest Finish Time for task ti
XET(ti,VMv)	Expected Execution Time of task ti when scheduled on VMv
MET_W	Minimum Execution Time for Workflow

2.3.1 Basic Definitions

There are a number of basic terms used in the scheduling algorithm,

1. *Minimum Execution Time MET (t_i):* Minimum Execution Tine for task t_i is the minimum execution time possible for a task on an VM instance such that ET (VMv, ti) is minimum and VMv € VM_{set}. According to equation (1)

   $MET\ (ti) = \min VMv \in VMset\ \{ET\ (ti, VMv)\}$ (1)

2. *Earliest Start Time EST(t_i) :* It is the minimum time at which a task ti can start its execution such that all its parent tasks have been terminated and data has been transferred to the VM on which ti has to run. According to equation (2)

   $EST\ (tentry) = 0$

   $EST\ (ti) = \max tp \in ti's\ parent\ \{EST\ (tp) + MET(tp) + TT\ (epi)\}$ (2)

3. *Earliest Finish Time EFT(ti):* It is the minimum time in which a task can finish its execution. According to equation (3)

   $EFT\ (ti) = EST\ (ti) + MET(ti)$ (3)

4. *Expected Finish Time XFT(ti):* It is the actual time in which the task ti has finished its execution, keeping in mind all the delays in the scheduling. According to equation (4)

   XFT(ti) = $AST(ti) + ET\ (ti, type(vk))$,if ti is in execution (4)

   XFT(ti) = $\max tp \in ti's\ parent\ \{XFT(tp) + TT\ (epi)\} + ET\ (ti, type(vk))$,if ti is waiting for execution

5. *Expected Start Time XST(ti) :* It is the actual time at which a task ti goes into running stage, once all of its parent tasks have actually completed their execution. According to equation (5)

   $XST(\ tentry) = acquistiondelay$

   $XST(\ ti) = \max tp \in ti's\ parent\ \{XFT(tp) + TT\ (epi)\}$ (5)

6. *Latest Finish Time LFT(ti) :* It is the maximum permissible time in which a task must complete its execution so as to ensure that the user deadline is met. According to equation (6)

   $LFT(\ texit) = D$

$LFT(ti) = \min_{tc \in ti's\ childern} \{LFT(tc) - MET(tc) - TT(eic)\}$ (6)

7. *Latest Start Time LST(ti) :* It is the maximum time at which a task ti must start its execution such that all its parent tasks have completed their execution within user deadline D and also all the tasks that precede task ti i.e. all its children tasks must also be executed in such a way that they complete within deadline D.. According to equation (7)
   $LST(ti) = LFT(ti) - MET(ti)$ (7)

8. *Expected Execution Time XET (VMv, ti) :* It is the total execution time taken to execute a critical path from task ti till the end, such that task ti has been scheduled on VMv. According to equation (8)
   $XET(texit, VMv) = ET(texit, VMv)$
   $XET(ti, VMv) = ET(ti, VMv) + \max_{tc \in ti's\ children}\{XET(tc, VMv)\}$ (8)

9. *Minimum Execution Time MET_W :* It is the critical path length for the workflow such that all of its tasks are executed on the fastest VMs. According to equation (9)
   $MET\_W = \max_{ti \in W}(EFT(ti))$ (9)

2.3.2 The Proposed Algorithm

The proposed algorithm works on the principle of Just-in-Time (JIT) where the main aim is to complete the execution of the given workflow within user specified deadline along with maintaining lowest time possible. In order to execute a given workflow with some deadline, it is required to first make sure if the given deadline could be achieved in the present computation scenario. This is done using the minimum execution time. MET_W is calculated for the workflow, if this is greater than the deadline D defines, user is suggested to reconsider the deadline as it is unachievable. If MET_W is lesser than D, further steps are taken.

If the given deadline could be achieved, some pre-processing needs to be done on the workflow and then it is sent into a monitor control loop. In the pre-processing step, pipeline tasks are collapsed into a single task. This considerably reduces computation time. During monitor control phase, resource provisioning is done and scheduling decisions are taken. VMs are selected so as to reduce the total cost incurred on the system. Also, data transfer times are to be considered.
The 2 phases are explained in the next sections.

2.3.2.1 Pre-processing Phase

In this step, pipelined tasks are identified and collaborated as a single task. If a task has single parent task or single child task, those two tasks can be clubbed together as a single unit. The Execution time per VM is also added up together for those tasks. This not only reduces computational overheads but also data transfer times between these clubbed tasks is also removed. It also strengthens the dynamic scheduling process. Queue is used as a data structure. One by one every task node is visited and stored in the queue. A task tp from the front end of the queue is extracted, if this task has a single child task node tc, tc and tp are combined and tc+tp is pushed back into the queue. This process continues until all the tasks are visited.

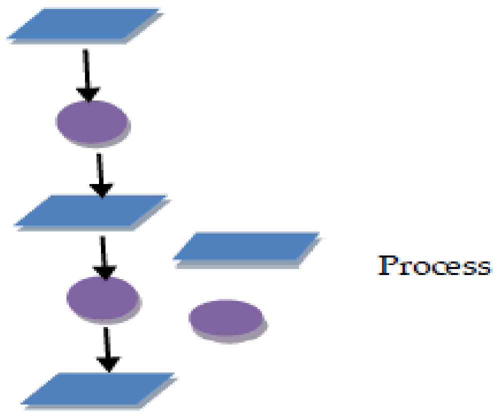


Figure23: Pipeline Workflow example

In the example, tasks T1 and T2 are pipeline tasks, as T1 has a single child task and T2 has single parent task, so these two tasks are clubbed together as T1+T2. Now T2 can be executed on the same VM as T1. Data transfer time from T1 to T2 is saved and overhead is reduced largely from the monitor control loop.

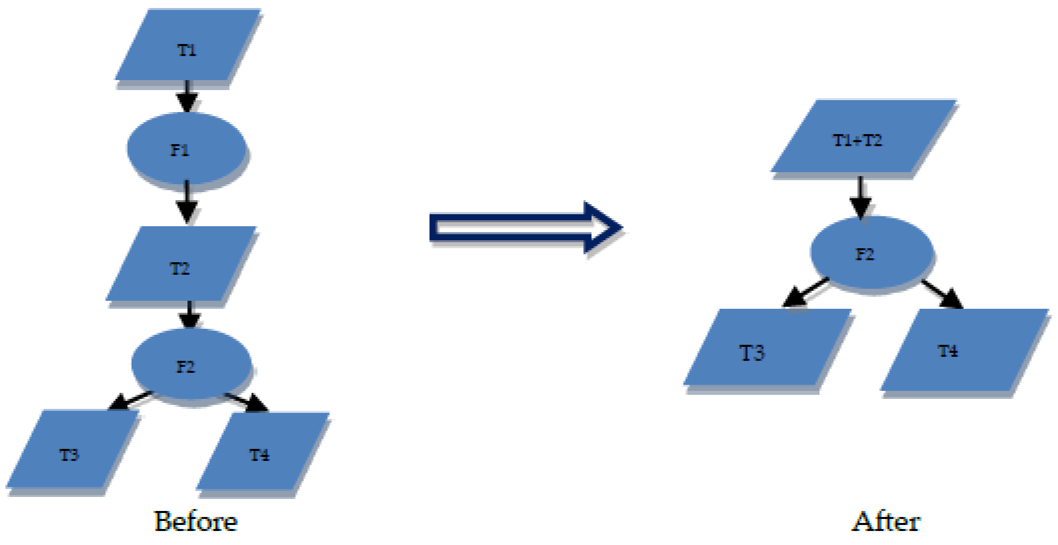


Figure24: Pre-Processing illustration

2.3.2.2 Monitor Loop

Once the workflow has been processed and pipelined tasks have been clubbed into single unit, next task of the scheduling algorithm is to go through all the tasks of the workflow one by one and find the cheapest VM on which these tasks could be executed. First the entry task is sent to the function handling the job to find the cheapest VM available. After the entry task has been dealt with, the algorithm goes into a monitor control loop where surviving the deadline as well as finding the cheapest VM for all the tasks becomes major concern. With every task being assigned a VM type, Actual Start Time (AST) and Actual Finish Time (AFT) is updated. Monitor Control Loop lasts until all the tasks of the given workforce has been not dealt with and assigned with a proper

VM to work on. For every task of the workflow the *cheapestVM* function is called in order to assigned a VM. AST and AFT are updated simultaneously.

Various inputs are to be provided to the algorithm at this stage:

- DAG(T,E) created out of the workflow is fed to the algorithm in form of matrix
- Cost matrix for all VM instances
- Transfer Time TT (nxn) matrix
- Execution Time ET (nxm) matrix
- Deadline D specified by user
- Acquisition delay for VM

After taking in the input values, first of all Minimum Execution Time of the workflow is calculated and checked if it is less than D, i.e. can the deadline assigned be achieved or not. If the result is negative, user is asked to reconsider deadline value. Else, the process continues.

Minimum Execution Time matrix and Latest Finish Time matrix and Expected Execution Time matrix is constructed using the equations discussed in the above section. Pre-processing function is called to club pipelined tasks into single unit. Once this is done, for all the entry tasks, *cheapestVM* function is called to assigned appropriate VM to these entry tasks. After entry tasks have been dealt with, a loop is applied in which every other task except entry tasks are sent to *cheapestVM* function, their Actual Start Time and Expected Finish Time is updated following which the schedule is updated.

2.3.2.3 CheapestVM mechanism

The CheapestVM function takes a task as an input and maps it to its cheapest possible VM such that the deadline set by the user could also be met. Here one might think why not directly choose a VM that is minimum in cost, but that would be a wrong practice. Taking such a step, without considering its affects on the children tasks could force them to run on a costlier VMs. Also, a case might arise that the deadline could not be met because of running them directly on the cheapest VM.

CheapestVM at each step helps choosing VM in such a way that all the tasks of the critical path are assigned VM in such a away that overall cost comes out to be minimum, also the time taken to execute the workflow is reduced considerably. Data Transfer time is also taken under control. In order to reduce this data transfer time, algorithm tries to find if the current task t could run on the same VM v_p on which the parent of t witch the highest Expected Finish Time has been running. To do this, Expected Start Time (XST) of t is evaluated, but this XST is the one when t will be executed on v_p. Expected Idle Start Time for the VM v_p is also calculated. Now if XIST for v_p is lesser than XST for t, t can be assigned upon $v_{p.}$ This means at the time when task t is supposed to start its execution, VM v_p is free and could be loaded with task. Hence, the data transfer time is reduced and task could run easily. If XIST for VM v_p is greater than XST for task t, then t can not run on v_p as it is not free at the time of execution. XST for t is then updated accordingly on the basis of above explained equations.

In such a case, other steps are taken to ensure a VM is available to execute the task. Here XST for t is updated as a maximum sum of Expected Finish Time of the parent of task t and Data Transfer Time from that parent to task t.

In case, t is an entry task, XST is the acquisition delay for VM.
Once XST for t has been updated, A set of VMs are selected on the basis of given condition. If for a particular VM j, XST (t) + XET (VMj, t) <=D, VMj is added into the set. For all the VM added to this set, cost to execute task t is calculated according to equation (1). VM with the minimum cost is selected for the execution of task t.
At last, overall cost and total time taken for the workflow execution is calculated.

2.4 Illustrative Example

An example has been taken in this work to give a better understanding of the algorithm. The workflow given in the example is represented by the DAG shown in figure25. All the steps discussed in the algorithm above is explained in the given example.

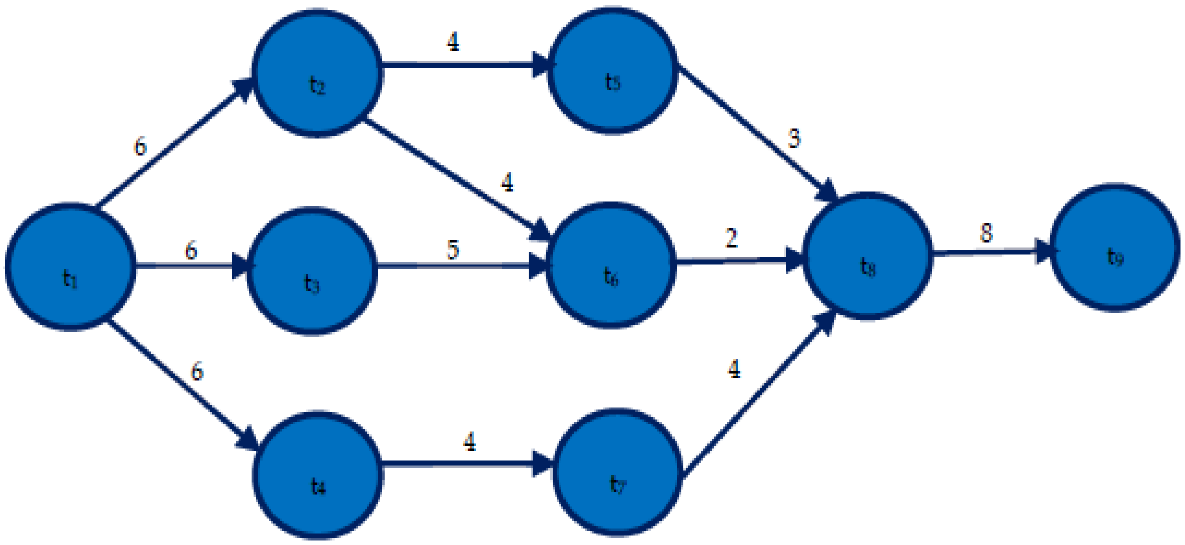


Figure25: workflow before pre-processing

In the given workflow, we take tasks from t1 to t9. All these are tasks are connected with each other with edges. Numbers on these edges show the data transfer time between those two task nodes. Three VMs have been considered in the example, i.e. VM_{small}, VM_{medium}, $VM_{large.}$ Cost for each of the is given in the matrix. VM_{small} costs \$1, VM_{medium} costs \$2, VM_{large} costs \$4.

Workflow deadline D as set by the user here is 50 unit of time. Also, in this example Actual Start Time (AST) is equal to Expected Start Time (XST). Provision delay has been neglected here in case of performance variations. Since there is no performance variation delay, AST for every task can be considered same as XST for every task.

To start with the execution of the algorithm, few data matrices are computed for calculation purpose. Minimum Execution Time (MET) is calculated for all 9 tasks. Earliest Start Time EST) matrix for all the tasks is computed. Earliest Finish Time (EFT) for all tasks is also computed. MET_W i.e. Minimum Execution Time for the complete workflow is computed according to equation (9). If MET_W<=D, further steps of algorithm are processed. Else, process terminates with a message to reconsider the data. In the example, MET_W=49 which is <=D hence, algorithm proceeds.

In the very first step, this workflow is fed to the pre-processing unit where pipelined tasks are clubbed together to form a single task. As a result of pre-processing, tasks t_4 and t_9 are clubbed together t_{4+9}. Similarly, tasks t8 and t9 are also clubbed together as t8+9. The modified Execution Time matrix and Transfer Time matrix are updated according to pre-processing step as shown in figure26.

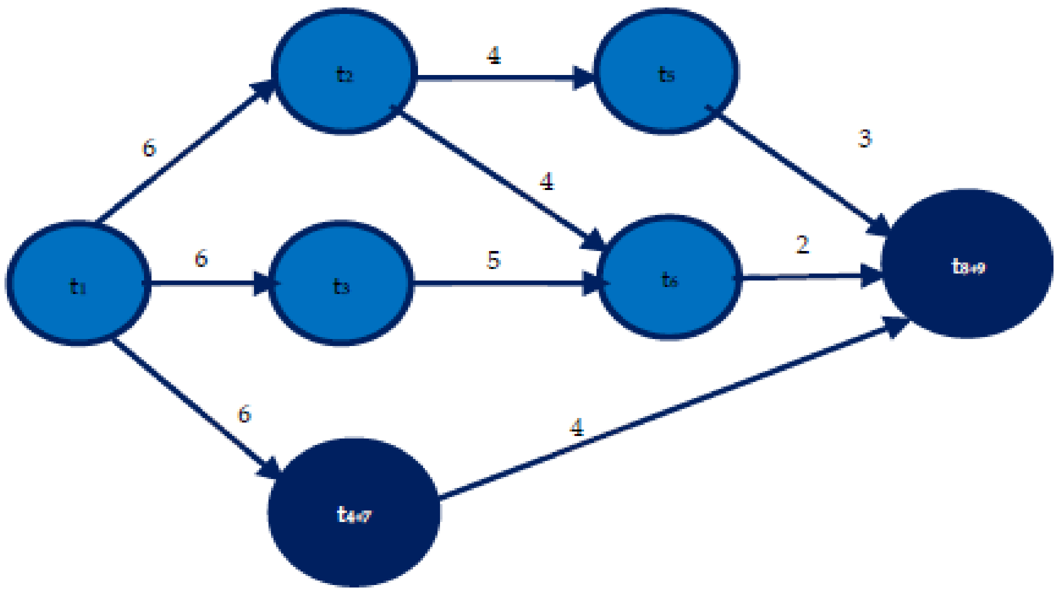


Figure26: Workflow after pre-processing

$$ExecTime = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \end{array} \begin{array}{c} \begin{array}{ccc} v_s & v_m & v_l \end{array} \\ \begin{bmatrix} 4 & 2 & 1 \\ 6 & 4 & 2 \\ 16 & 9 & 6 \\ 12 & 7 & 4 \\ 11 & 8 & 5 \\ 7 & 3 & 2 \\ 18 & 12 & 8 \\ 13 & 9 & 5 \\ 15 & 12 & 9 \end{bmatrix} \end{array}$$

Figure27: Execution Time matrix

$$TransferTime = \begin{array}{c|ccccccccc} & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 \\ \hline t_1 & 0 & 6 & 6 & 6 & 0 & 0 & 0 & 0 & 0 \\ t_2 & 0 & 0 & 0 & 0 & 4 & 4 & 0 & 0 & 0 \\ t_3 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ t_4 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ t_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ t_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ t_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ t_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \\ t_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Figure28: Transfer Time matrix

After pre-processing, main function calculates some more data matrices like Minimum Execution Time (MET) for the updated workflow, Expected Execution Time (XET) for all the tasks and Latest Finish Time (LFT) for all tasks. Data is shown in figure27 and figure28.

Task t_1 is the entry task here. Once fed to the *CheapestVM* method, it allocates t_1 to VM_{medium} and XST for t_1 is set to 1 because it is an entry task hence, XST for all entry task is set to acquisition delay. Similarly, all the other tasks are fed to monitor control loop, which on the basis of the algorithm, are allotted VMs from the lot. Final provision list has been given below.

$$ExecTime = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_{4+7} \\ t_5 \\ t_6 \\ t_{8+9} \end{array} \begin{array}{c} \begin{array}{ccc} v_s & v_m & v_l \end{array} \\ \begin{bmatrix} 4 & 2 & 1 \\ 6 & 4 & 2 \\ 16 & 9 & 6 \\ 30 & 19 & 12 \\ 11 & 8 & 5 \\ 7 & 3 & 2 \\ 28 & 21 & 14 \end{bmatrix} \end{array}$$

Figure 29: Execution Time after pre-processing

$$TransferTime = \begin{array}{c|ccccccc} & t_1 & t_2 & t_3 & t_{4+7} & t_5 & t_6 & t_{8+9} \\ \hline t_1 & 0 & 6 & 6 & 6 & 0 & 0 & 0 \\ t_2 & 0 & 0 & 0 & 0 & 4 & 4 & 0 \\ t_3 & 0 & 0 & 0 & 0 & 0 & 5 & 0 \\ t_{4+7} & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ t_5 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ t_6 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ t_{8+9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Figure 30: Transfer Time after pre-processing

Deadline D=50							
Tasks	t_1	t_2	t_3	t_{4+7}	t_5	t_6	t_{8+9}
MET	1	2	6	12	5	2	14
LFT	14	24	27	32	33	34	50
XET(V_s)	62	45	51	58	39	35	28
XET (V_m)	42	33	33	40	29	24	21
XET (V_l)	27	21	22	26	19	16	14

Figure 31: Values of MET, LFT, XET for the workflow

Table 2: Provision List

Task	VM
T_1	1
T_2	1
T_3	1
T_{4+7}	1

T_5	0
T_6	1
T_{8+9}	1

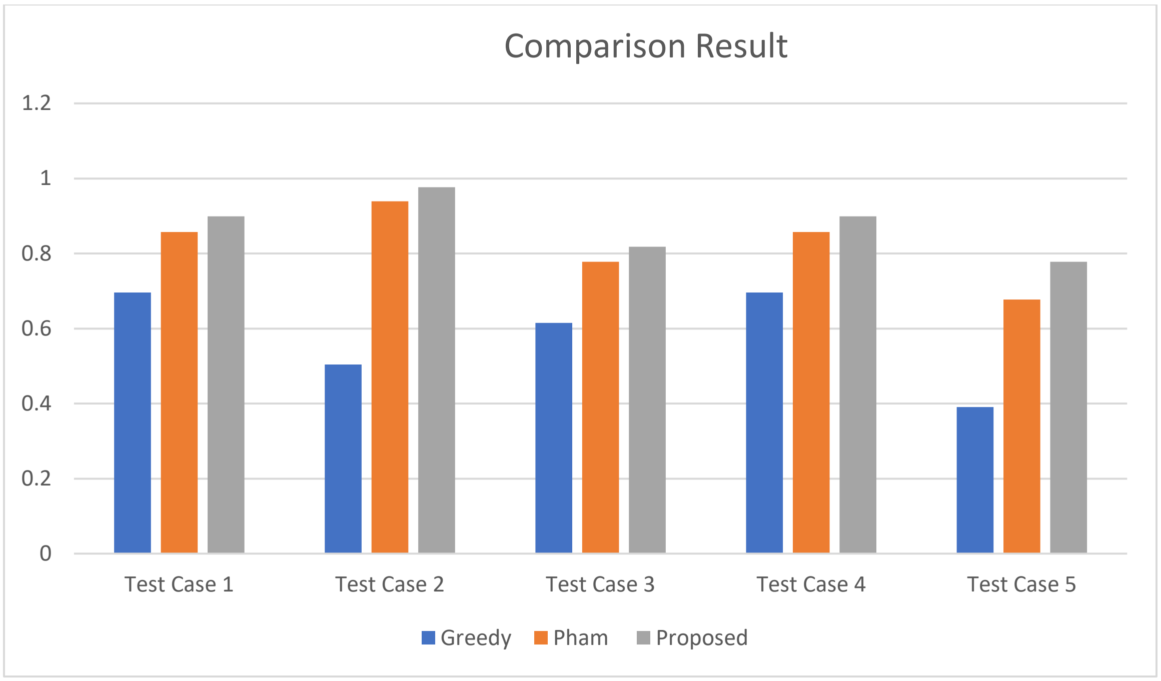
Thus, total cost incurred in the execution of complete workflow is 12.7 seconds.
Total time taken by the scheduling algorithm is 43 which is less than deadline D.
In order to make comparison with two other famous algorithms, *cost-makespan tradeoff (CMT)* has been calculated and compared. CMT is described as a tradeoff between total cost incurred in workflow execution and total time taken to execute it. It is represented as:

$$CMT(a_i) = \frac{\min[cost(ak)]}{cost(ai)} * \frac{\min[makespan(al)]}{makespan(ai)}$$

Where,
a_k ,a_l € $\{a_1,a_2,\dots a_n\}$ i.e. list of all scheduling algorithms used in comparison
Higher the level of CMT better is the performance of that algorithm. CMT maximum can reach 1, this happens when time and cost both is minimum when compared to other algorithms.

2.5 Result and Comparison

For the purpose of comparison, two of the other algorithms have been implemented and tested on the same test cases as for which the proposed algorithm has been executed.
Comparative results show that the proposed algorithm proves to be more efficient in most of the cases. CMT value for the proposed algorithm is highest as compared to the other 2 algorithms.



Graph1: Comparative Result

Here, X-axis represents the test cases

Y-axis represents CMT value

The above graph shows 5 different test cases, for every test case CMT value of each algorithm is also represented. CMT value for proposed algorithm shows to be best.

CONCLUSION

Fog Computing has brought breakthrough in the field of computing by reducing the latency in computation and communication. As Fog computing is implemented at the edge of the network, it provides low latency, location awareness, and improves quality-of-services (QoS) for streaming and real time applications. In order to deal with the setbacks of Fog Computing, efficient job scheduling algorithms must be employed. The algorithm proposed in this work keeps a tradeoff between total cost incurred and time taken to execute the workflow. Also, comparison has been made with 2 other scheduling algorithms. In this comparison the proposed algorithm gains highest CMT value in most of the test cases. As a future improvement, this algorithm needs to be made more efficient when dealing with higher number of tasks. Also, the processors needs to be more powerful.

REFERENCES

[1] Jyoti Sahni, Deo Prakash Vidyarthi, A Cost-Effective Deadline-Constrained Dynamic Scheduling Algorithm for Scientific Workflows in a Cloud Environment, DOI 10.1109/TCC.2015.2451649, IEEE Transactions on Cloud Computing, 2017.

[2] Xuan-Qui Pham, Eui-Nam Huh, Towards task scheduling in a cloud-fog computing System, The 18th Asia-Pacific Network Operations and Management Symposium (APNOMS) 2016

[3] Ivan Stojmenovic, Sheng Wen, The Fog Computing Paradigm: Scenarios and Security Issues, Proceedings of the 2016 Federated Conference on Computer Science and Information

[4] Redowan Mahmud and Rajkumar Buyya, Wiley STM / Editor Buyya, Srirama, Modelling and Simulation of Fog and Edge Computing Environments using iFogSim Toolkit, Fog and Edge Computing: Principles and Paradigms, Chapter 17 / Introduction to Fog and Edge Computing,2017.

[5] Nguyen Doan Man and Eui-Nam Huh, “Cost and Efficiency-based Scheduling on a General Framework Combining between Cloud Computing and Local Thick Clients,” in 2013 International Conference on Computing, Management and Telecommunications (ComManTel), pp. 258-263, 2016.

[6] L. Zeng, B. Veeravalli and X. Li, “ScaleStar: Budget Conscious Scheduling Precedence Constrained Many-task Workflow Applications in Cloud,” in 2012 IEEE 26th International Conference on Advanced Information Networking and Applications, pp. 534-541, 2016.

[7] Nguyen Doan Man and Eui-Nam Huh, “Cost and Efficiency-based Scheduling on a General Framework Combining between Cloud Computing and Local Thick Clients,” in 2013 International

Conference on Computing, Management and Telecommunications (ComManTel), pp. 258-263, 2013.

[8] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, Sateesh Addepalli, Fog Computing and Its Role in the Internet of Things, MCC'12, August 17, 2012, Helsinki, Finland

[9] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning,Tie Qiu, Survey on Fog Computing: Architecture, KeyTechnologies, Applications and Open Issues, Journal of Network and Computer Applications, Volume 98, 15 November 2017, Pages 27-42

[10] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li, Fog Computing: Platform and Applications, 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies

[11] https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf

[12] https://en.wikipedia.org/wiki/Fog_computing

[13] http://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-Fog-Computing.html