CONTEXT-AWARE RESOURCE ALLOCATION AND SCHEDULING FOR HYBRID MOBILE CLOUD

by

Hager Mohamed Ghouma

B.Sc., Tripoli University, Tripoli, Libya,

2004

A Thesis

submitted in partial fulfillment of the requirements for the

degree of

Master of Applied Science

in the Program of Electrical and Computer Engineering

Ryerson University

Toronto, Ontario, Canada

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Abstract

Hager Mohamed Ghouma

Context-Aware Resource Allocation and Scheduling of Hybrid Mobile Cloud

M.A.Sc, Electrical and Computer Engineering, Ryerson University, 2015

Cloud services can compensate for the resource constraints of mobile devices. However, challenges of utilizing the cloud service by mobile users arise from inherent characteristics such as user *mobility* and device *energy*. In this paper, we propose a scheme to monitor the energy level and communication quality as a part of a mobile user context information, and develop a resource allocation and scheduling scheme to adapt to the context changes by exploiting the slack time. The objective is to reduce the execution cost of the jobs while meeting the jobs deadlines set by the users. We developed Simulated Annealing based resource allocation algorithm and Earliest Deadline First scheduling. Simulation of our scheme using CloudSim and synthetic workload based on Google Cluster Traces shows benefits in reducing execution cost and improving resource utilization.

Acknowledgements

I would like to take this opportunity to express gratitude to my supervisor, Dr. Muhammad Jaseemuddin. I am indebted to him for sharing his expertise and guidance extended to me.

I am also grateful to my sponsor for providing me with this invaluable opportunity.

I wish to express my sincere thanks to my husband, my family and friends who supported me through this venture.

Dedicated to my loving family and to the memory of my beloved parents

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Cloud computing is poised to play a big role in wireless networks by providing computing service to an increasing number of mobile clients. The number of wireless cloud users worldwide is expected to have grown at a rate of 69% in 2014 [1].

Increasingly cloud clients use personal wireless communication devices such as smartphones, laptops and tablets with wireless connection to access the cloud. These devices have limited computing power, limited storage, and limited battery power. They also suffer from loss of their scarce energy when operating at full computing power. Cloud providers offer scalable and reconfigurable services such as computing, storage and network services. The cloud infrastructure can provide mobile devices access to scalable computation and storage capabilities. The cloud can also provide location-aware and power-aware services that provide energy savings at mobile devices. . However, the cloud providers providing services to mobile clients face new challenges in terms of managing and providing services at the client's expectations of agreed upon quality while dealing with large variations in loads, such as crowdsourcing.

The workflows of cloud-based applications are typically partitioned, such that the application interface runs on mobile devices while the data-intensive and/or compute-intensive components as a set of jobs are executed in the cloud. . Mobile clients interact with the cloud via Cloud Service Broker (CSB). The CSB manages cloud services to the mobile clients as on-demand pay-per-use service. The US National Institute of Standards and Technology (NIST)

defines the cloud service broker as “an entity that manages the use, performance and delivery of cloud services, and negotiates relationships between Cloud Providers and Cloud Consumers”. A cloud broker is responsible for service discovery, integration, data and service aggregation, customization, quality assurance and optimization [2].

Personalization and customization of cloud services is also achievable using mobile clients’ context information. For mobile clients, context could be defined by: (1) user context; such as the client ID, (2) device context; such as energy level of the device and location, (3) mobile service context such as the quality of wireless networks, and (4) the time context that are the time stamps at which the context information is gathered. In addition, mobile context information could be described as the knowledge the cloud receives about the mobile clients at a certain period of time. In the cloud, mobile context information could be used to optimize service offerings to the clients in terms of time and cost.

The challenges of providing cloud service to mobile clients arise from the inherent characteristics of the mobile devices, specifically mobile device’s *energy level* and *mobility*. When the energy level of a mobile device drops below a critical level, then it may cause disruptions in communication with the cloud, which results in loss of data. Mobility of devices induces change in their locations. Moreover, mobility affects the quality of communication by changing the connection type and/or data rate. For example, the connection type may change from WiFi to cellular or vice versa. The change of location may also cause variation in data rates even though connections type remains the same. The change in the connectivity and data rates of the mobile clients plays a significant role in estimating the expected response time for the service. The estimated response time has to be calculated such that it takes into account the

extra communication delay resulting from the connectivity changes to ensure that the cloud provider meet the Service Level Agreement (SLA) agreed with the users. Estimation of response time affects the allocation and scheduling of cloud resources that are constrained to meet the deadline specified by the cloud-based mobile application. Erroneous estimation of the response time may cause the management unit to allocate resources that would not satisfy the quality of service expected by the cloud users that may result in violation of the SLA.

The cloud resources are distributed over a long distance and available both globally and locally. The knowledge of user location can be incorporated in the cloud resource management to optimize the selection and allocation of resources. The resource allocation decision based on user's proximity to a datacenter can potentially provide lower service latency than in case of distant datacenter. However, the selection of location of the service could be hindered by the availability of specific services or data required by the user as data transfers between datacenters could accumulate significant costs and increase latency.

1.1. Motivation

The use of the scalable and powerful cloud services by mobile clients is attractive due to the limited computation, storage, and energy of mobile devices. However, wireless connectivity presents a bottleneck to mobile devices when taking advantage of the scalable resources powerful computations in the cloud. The variation in data rate may increase the communication time for the exchange of data between mobile clients and the cloud, which may affect, especially delay-sensitive, applications at mobile devices. The deadline on the execution of applications set by the mobile clients may unnecessarily restrain the system even when the mobile client is unable to receive the results from the cloud. As it is shown in this thesis, the

cloud service broker can utilize user context information in making intelligent resource provisioning decisions. For example, if a mobile device experiences long delays due to a drop in its network connection data rate, then this information would be significant for the resource provisioning unit in the cloud provider as the deadline on the execution of the application provided by the client could be extended to relax constraints on the resource allocation. Likewise, when the energy level of a mobile device is not sufficient to complete the transaction, the mobile application in the cloud can be suspended and resources are withdrawn until it replenishes enough energy to successfully communicate with the cloud. This scheme benefits both the mobile user and the cloud provider. As for the mobile user, the resources are allocated only when they are used for useful computation, thus relieving the mobile user from paying for a service that could not possibly be received. When the mobile device regains sufficient energy for communication only then the mobile application in the cloud resumes and the user is billed for the service. In addition, for the cloud provider, the resource thus freed could be allocated to other jobs that suffer from longer wait times, thus increasing the throughput of the cloud provider resources.

We can summarize the above discussion in a problem statement as follows:

> *For a number of jobs N and a number of cloud resources R, an allocation map that binds a job with a resource is needed such that the resource meet the job deadline and the total cost of the resources do not exceed the client's budget.*

Once the resources are provisioned to the users, an adaptive algorithm reads the user context information and makes a decision on whether the same resource allocation continues or changes to reflect the changes of the mobile client context.

1.2. Contributions

This thesis aims primarily at cloud resource provisioning and scheduling for mobile users using cloud-based mobile applications. The mobile user context information specifically the mobile device location, connection quality and energy are used when allocating the services for the application workflows that are dispatched and processed in the cloud as a set of tasks.

The main contribution of this research

- A system model for a context-aware resource provisioning and scheduling of a hybrid cloud in mobile environments. The system mainly consists of three parts: mobile client context engine, cloud broker and the cloud resources (datacenters). The functionality components are the context information collector and monitor, resource allocation unit and the resource scheduling unit.
- The resource allocation is implemented using a meta-heuristic algorithm; simulated annealing. The adaptive simulated annealing algorithm takes into account the turnaround time of the requests, the location of the mobile clients and the energy consumed by the mobile devices when finding the best suited resources for the mobile client's requested jobs. Figure 1.1 shows a simple diagram of the proposed changes to the cloud system.
- Task scheduling is done using Earliest Deadline First algorithm. The algorithm calculated the start and end time for each tasks in the client job as well as updating the start lease time and end lease time for the resources provisioned to these tasks.

- Simulation of the proposed system was done using a discrete event simulator, CloudSim. The data used for the experiments was extracted from the Google Cloud Trace released in May, 2011 [3].
- Evaluation of the proposed system was achieved using three different parameters; violations of deadline, execution time for each user and CPU utilizations of the cloud provider machines.

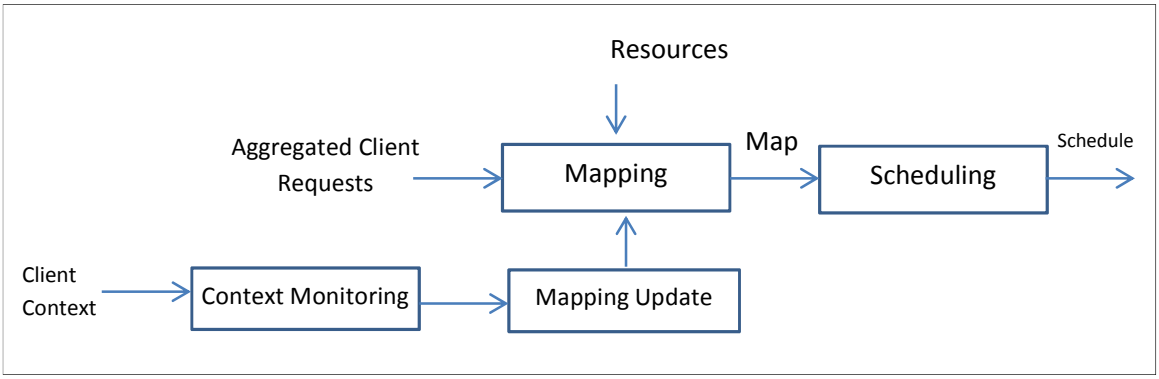

Figure 1.1 Diagram of proposed system changes.

1.3. Organization of Thesis

The remainder of the thesis is organized as follows: in chapter 2, the benefits and challenges of using cloud computing for mobile environment were highlighted. In addition, the related work in the area of mobile cloud computing and resource management in cloud computing was reviewed. In chapter 3, the proposed system architecture and system model are presented. In addition, in chapter 4, the simulation of the proposed system model is done using CloudSim. Three parameters were used to evaluate the system and the results of the experimentations were analyzed. Chapter 5 is the conclusion and future work.

Chapter 2

Background and Literature Review

2.1 Cloud Computing

Cloud computing is delivering hosted services over the internet at a cost. NIST [4] defined cloud computing as a model where service providers enable the cloud customers to run their IT infrastructure in the cloud using a pool of configurable compute resources that can be managed with minimal effort or interaction from the service providers. These compute resources are a dynamically provisioned collection of interconnected and virtualized computers; i.e. cloud computing offer computation, storage and network resources.

The main characteristics of cloud computing are: on-demand self-service, resource pooling, elasticity, (which is the ability to use as many resources as needed to meet the cost and timing constraints), and pay-per-use measured service. Measurement, controlling and reporting the resource usage could also be done, which will provide transparency of the utilized services for both the cloud service consumer and the cloud service provider [4].

Cloud computing main deployment models are divided into the following models: private cloud, community cloud, public cloud and hybrid cloud. A private cloud could be hosted internally or externally and either managed by a third party or the organization that owns it or a combination of them. The key feature of the private cloud is that it is operated and provisioned to be used exclusively by a single organization. A community cloud is operated and provisioned to be used exclusively by multiple organizations sharing common concerns such as mission or security policies, and similar to a private cloud it is managed by one of the organizations or a third party or a combination of them. The community cloud also could exist internally or

externally. A public cloud is hosted on the premises of the cloud provider and its infrastructure is provisioned to be used publicly over a network. A hybrid cloud is combination of two or more clouds: private, public or community; each preserving their own unique entities although they are bound together. The hybrid cloud benefit by using different deployment models, cloud bursting is enabled such that the private cloud could use the resources of the public cloud as the demand increases. This is an advantage as the services of the public cloud will only be used when needed. Figure 2.1 shows the cloud deployment models.

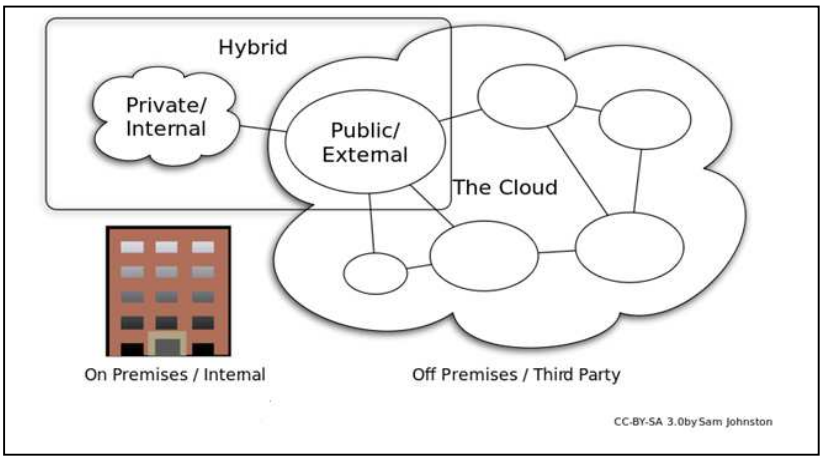

Figure 2.1 Cloud Deployment Models

The provided cloud services are divided into three service models: infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS). IaaS is the providing provisioning of servers, storage, networks in addition to other resources to be used by the consumers to run their software. The consumer could control the operating systems, storage and the consumer's deployed applications. However, the consumer will not have the capability to control and manage the cloud infrastructure. PaaS is providing an infrastructure for a consumer to deploy applications as a solution stack. The consumer creates software and controls their deployment and configuration settings using the tools, libraries and programming

languages along with the infrastructures services provided by the service provider. SaaS is the service of providing packaged software running on the cloud infrastructure for the consumer.

In SaaS the software and the data are hosted in the cloud and the consumer does not have the capability to manage the cloud infrastructure or the software capabilities, with the exception of the limited application configuration settings that are specific to the user. The consumer accesses these services through an interface; such as the web browser or a program interface. The cloud resources in the system level and the components of the core middleware are responsible for delivering the IaaS. The user level middleware components are responsible for providing the PaaS and the top layer of the user level that consists of the cloud applications uses the services provides by the lower layers to provide the SaaS.

Cloud resources are interconnected and distributed over different locations in the network to increase availability, and reduce both bandwidth cost and latency due to the distance proximity of the services to the users.

Cloud Computing Applications Workflows: Cloud customers build their applications in the cloud with almost no upfront or startup cost of the infrastructure. Cloud applications use APIs of Internet or network accessible services. This on demand scalable services has efficient resource utilizations capabilities and the scalability and management mechanisms are hidden from the cloud customer, i.e. self-management system. Different types of application could run on the cloud; however the application workload should have the ability to be arbitrarily partitioned into concurrent instances. In addition, if the communication between the concurrent instances is intensive, the application might not perform well as it will face bandwidth limitations due to the network-centric nature of the cloud computing system [5]. Examples of applications that

could use the cloud architecture include but not limited to processing pipelines such as document conversion pipelines and video transcoding pipelines, batch processing systems such as log analysis and automated unit testing and deployments, and instant websites or seasonal websites. In addition, mobile interaction applications and science and engineering applications could be deployed on the cloud as they would benefit greatly from the massive storage and scalable computational power of the cloud system.

Cloud applications consist of a series of independent units of works to be performed in the cloud known as tasks. The workflow is the simplified description of these tasks complex activity that consists also of the data elements, control sequences and data dependencies [6].

2.2 Mobile Computing: Mobile Users and the Cloud

There are two types of mobile cloud computing according to a survey done by Khan *et. al* [7]. The first type is an ad-hoc mobile cloud where is all mobile devices within the vicinity of the user act as cloud agents and provide the user with access to the cloud services residing in the internet. Also, the near mobile devices might have a set of services that could be used by other mobile devices, hence forming their own local cloud which is known as cyber foraging. Ad-hoc mobile cloud interstice characteristic is the mobility of the services [7][8]. The second type of mobile cloud is infrastructure based cloud, and unlike the ad-hoc mobile cloud, the services remain static. The infrastructure based cloud provides services to the mobile devices via the internet. The main focus of the work of this thesis is on the latter type, i.e. infrastructure based cloud.

Smart mobile devices access the cloud by using one or more of various application processing frameworks. The framework offloads the mobile application to the cloud such that

a part or the whole of the application is stored and processed in the cloud. Application offloading could be done by either migrating the whole application to the cloud, partitioning the application and migrating the computation-intensive and/or data-intensive partitions to the cloud, or processing a VM image of the application to be sent and run in the cloud [7]. An example of a mobile application partitioning that uses the cloud is voice to text conversion applications such as Apple Siri, which is a voice command search engine. The voice is acquired using the mobile device, iPhone or iPad, and then sent to the cloud to convert it to a text and a search for the text will be performed. The search result are then sent back to the mobile device.

Cloud based mobile applications are presented as Task Interaction Graphs TIGs and offloaded to the cloud for execution. The application will be executed in the cloud as a set of jobs and each job consists of group of dependent and indivisible tasks. In addition to the jobs, the requests also define other information such as the job's requirements and jobs' deadline.

2.3. Context Aware Mobile Computing

Mobile context information has been used recently to provide the mobile users with more personalized and customized services that improve their over-all experience. Context aware mechanism is based on three parts: (1) obtaining the context information at the user mobile device, (2) sending this information to the cloud, (3) efficiently use this information for cloud resource management to improve the user's quality of service.

2.3.1. Mobile Devices' Network Connection

Mobile users access the cloud using either 3G/4G LTE via the mobile network or a Wi-Fi connection via in-home Wi-Fi or a Wi-Fi hotspot. According to CEET white paper [1], 33% of wireless cloud users connect via Wi-Fi and 67% of these users connect via 4G LTE. The level of

wireless connectivity of the access network is one of the main bottlenecks when achieving the user requested quality of services. These mobile users experience variant data rates depending on type of connection. While 3G networks offer wide area connectivity, they suffer from long delay and slow data rate [9]. Alternatively, Wi-Fi connections offer low communication latency [9]. Wi-Fi deployment is used to connect to either a local cloud or a public cloud via the internet depending on type of the mobile application.

2.3.2. Service Location and User Location

Another inherent characteristic of mobile users using the mobile network is mobility. Location of wireless cloud users is crucial to the cloud provider to determine the optimal placement of the required services in distributed datacenters in the cloud[10][11]. The services are provided according to the relevant location of the users to the distributed datacenters. It is important to notice that in the mobile cloud, the location of the users and the location of services are of finer granularity in the ad-hoc mobile cloud than in the infrastructure based mobile cloud. As in the later the services reside in datacenters that cover larger areas where in the former the mobile devices offer the services and the mobility plays a big role in changing the location more frequently. In addition to the location, service placement techniques may consider other factors such as the temporal and economical states of both the users and the provider when the request was launched.

2.3.3 Mobile Energy

Battery lifetime is another challenge for mobile devices. Energy of the mobile device has a significant impact on the quality of service the users will experience. The decision of offloading the application to the cloud to save the mobile device energy must take into account the

energy consumed due to increased communication resulting from connecting to the cloud. Therefore, mobile device must have an efficient level of energy to communicate with the cloud while running the cloud based application. In addition, for the offloading process to be energy efficient the mobile has to gain in time and computation capacity [12]. The time gain and computation complexity of the partitioned mobile applications, which will be offloaded to the cloud, are done using different methods and mechanisms and they are out of the scope of this thesis.

2.4. Resource Management in the Cloud

The on-demand resource provisioning and pay-per-use features of the cloud services require that the service providers use minimal infrastructure and maximize their resource utilization while meeting the Quality of Service, QoS.

Quality of Service is the ability to provide different jobs with a service that guarantees a certain level of performance to a job. Service level Agreement, SLA, is an agreement between the user and the provider that specifies that required service level parameters the user requires in terms of speed, size, bandwidth and delay.

The cloud is a group of datacenters interconnected and managed using a cloud controller. Storage units are used in combination with network resources to synchronize between the different datacenters. A datacenter is a group of interconnected computing units called hosts. Hosts are equipped with hypervisor that enable them to run one or more virtual machines *VMs*. Each datacenter has a local controller/broker that is responsible of managing the deployment of VMs and execution of the different tasks.

In general, workflow management is done using three components: (1) user portal to compose, submit and monitor workflow applications, (2) a workflow editor to facilitate the workflow composition and virtualization, (3) and a resource broker that acts as a mediator between the users and the resources, it performs the resource discovery and passes the requests to the corresponding cloud controller that is responsible for allocation and scheduling [11].

Cloud providers manage their resource on different levels using many different modules or components such as service placement, data placement, resource provisioning, and task scheduling strategies. As for the resources of the cloud, the management entails allocating the resources to the users' requests, sending the requests to the specified resource in the cloud datacenters, scheduling the resources, and balancing the load between the cloud resources.

A resource is the unit(s) that carries out the operation of executing the job. Also as mentioned before, the job will be executed on different types of resources such as a computation unit (CPU), a storage unit (memory) and network link for data transporting (bandwidth). To achieve efficient performance, effective resource allocation and scheduling mechanisms are needed. Resource allocation is mapping tasks to given resources in a given time period according to certain constraints and optimization criteria. This map is then submitted to the corresponding datacenters and the jobs are executed according to criteria defined by a scheduling policy.

In addition, the user sends a request in the form of a job (computational activity) consisting of a set of tasks to be executed in the cloud; a task is an indivisible minimum computation unit to be run on the resource. The user request also describes the tasks dependencies, the job

processing requirements, the input data, and job priority. In addition, the job description might also include some constraints such application deadline and budget of the user. The job constraints could be categorized into application centric such as user specified deadline or budget and server provider centric that could be one or a combination of the following: maximize resource utilization, number of successfully completed jobs or minimization of response time.

Allocating and scheduling the job tasks to cloud resources could be done in many different methods and using various algorithms depending on the requirement of the system. From this point on in the literature, the term scheduling will be used for both allocating and scheduling the tasks in the cloud. Scheduling could be static or dynamic. In static scheduling the data is pre-fetched and the tasks are lined for execution according to their specified execution durations required before runtime, which reduces overhead in the system. Alternatively, in dynamic scheduling information about task execution duration is not known and the scheduling is done in real time as the application executes. In the next section, a review of some resource managements mechanisms is presented.

2.5. Literature Review

Several areas of cloud resource management will be reviewed in this section including load prediction [13][14][15][16][17], data placement [18], resource allocation [11][19][[10][20], resource sharing[13][22] and task scheduling[19][20][28][20].

2.5.1. Cloud Resource Provisioning for Mobile Environment

Ferber *et. al.* [19] presented a middleware based on Amazon Compute Cloud (EC2) to relocate the Java computing intensive applications to the cloud resources, these application are

called the cloud-assisted mobile applications. It implements cloud server features such as hosting the remote services, allocation and management of services and finally handling the billing and accounting information. Two different resource allocation strategies were used; the first strategy allows only one remote service to a single VM. This allows the remote service to use the full capacity of the VM but generates overhead. The second strategy returns a VM immediately upon request; the request will be multiplexed to an existing VM if there are no available VMs on standby. This strategy reduces the overhead by reducing the wait time; however, it increases the sharing of the same VM and may cause it to overload. Mei *et. al*. [21]proposed a framework to outsource the latency-tolerant mobile application to the cloud. The framework developed uses a scheduling mechanism that exploits the sharing of data across multiple mobile applications; the data sharing is detected by using data mining techniques.

Rahimi *et. al* [9][10] , presented a hybrid, tiered cloud architecture consisting of local and public clouds that provide services to mobile applications. A framework was developed to model the mobile application as a workflow of tasks. The Mobile usage patterns were directly translated from user mobility patterns. The user mobility pattern was presented by a trajectory of the mobile user and donated by a set tuples of location of the user and duration of time the user is residing in the location. In addition, the mobile application workflow consists of a sequence of logical and precise steps known as functions. Services that are capable of implementing a function were associated with that function. The workflow is presented as location time workflow, LTW; which consists of a sequence of sub-workflows indexed by the user trajectory tuple (location and duration). Furthermore, a heuristic algorithm based on simulated annealing called MuSIC, **M**obility-Aware **S**ervice Allocat**I**on on **C**loud, was developed

to search for the best set of services for each mobile user LTW. Their contribution significance was considering the user mobility when allocating the resources by defining the workflow for each user depending on their mobility pattern. Two mobility models were used: Random Waypoint (RW) model and Manhattan model. To study the performance of MuSIC two rich mobile applications, OCRS and video streaming and transcoding, were developed. Three parameters were used to evaluate the system; delay, power, and cost. Furthermore, two types of connection were considered, WiFi and 3G, the relevance of the connection type to the system model was addressed using the power the mobile device consumes when using either to connect to the cloud, and the delay which was defined as the time it takes between launching the request either on local or public cloud and the time the request is terminated. The two parameters power and delay, are user-centric. Alternatively, the price which is the cost of the service when executing the public cloud is provider-centric. The authors in this paper measured the delay on the mobile device and the measure of delay was not a factor when deciding the resource allocation for the requests. In this thesis, the mentioned delay is estimated by the proposed cloud system and incorporated in the allocation scheme as it will be evident in chapter 3.

Furthermore, another important aspect of mobile computing is Energy. Balakrishnana et. al [22] used a *slack time*, which is the difference between the deadline of the offloaded mobile workflow, which is presented as a TIG, and the actual response time of the cloud system for the same offloaded workflow. Whenever the slack time is larger, dynamic voltage and frequency scaling DVFS is used to reduce the processing capacity of the cloud resources by scaling the frequency of the processor. The task-resource assignment and resource-frequency assignment

is done using the slack time for the whole offloaded workflow, named *global slack time*. The worst case global slack time is distributed among the workflow tasks and a two level genetic algorithm is used to find the optimal assignment of task to resource and operation frequency of the resource such that the deadline is met. Their method showed that there will be a decrease in mobile energy consumption up to 25% due to offloading the workflow.

Alternatively, Zhang *et. al.* [22] investigated the partitioning of the mobile workflow. The mobile workflow is presented as a linear topology of tasks and each task is will be either executed on the mobile or the cloud. The decision of offloading considered achieving minimum energy consumption on the mobile device while meeting the application deadline. The task scheduling problem is this paper deals with whether the task would execute on the cloud or on the mobile and for that purpose the tasks were represented in a directed acyclic graph. The task scheduling problem was defined as a constrained shortest path problem and solved by using the Lagrangian Relaxation Based aggregated Cost (LARAC) algorithm. Furthermore, the task scheduling policy was done under the Markovian Stochastic Channel. The task scheduling decision takes into account the state of the wireless channel and if the state is "bad" leading to high communication delay then the task will be executed on the mobile instead of offloading it to the cloud. The experiment of this method considered only a one-climb policy, meaning only one time offloading from mobile to cloud.

2.5.2. Prediction of Host Load

Provisioning of server capacity to distributed applications is categorized into proactive or reactive. In the proactive measure the application models are trained to predict how much capacity is needed to provide certain mean response time for a given workload [13].

Cloud load changes vastly in the short term due to different cloud workloads; it is also difficult to model. Predicting load in the cloud is one of the measures taken to make task scheduling or load balancing decisions, thus enhancing system scalability and reducing network overload by optimizing the utilization of the system resources. The load data of the cloud have many different patterns and there yet to be a global prediction algorithm or method that can applied to all of them. However, many different prediction techniques and algorithms were developed to help solve the prediction problem of the server loads and all of these techniques and algorithms use a set of past and current cloud load measures to predict future load due to the repetitive nature of the human behavior.

Di *et. al* [13] used Bayes model to predict the cpu and memory load for the Google Compute Cloud. A new exponentially segment pattern was introduced and used in the prediction model. Google compute cloud platform was used for their prediction study; the study was based on load measurements of a Google data center that were traced by Google over a one-month period. The load traced events at minute resolution across 12,000 machines. The host load is defined in this study as the load of all the tasks that are running in that host (machine). A closer look at the Google load leads to the conclusion that it has frequent fluctuation and high noise. The mean load value is predicted over a single time interval and over consecutive time intervals. First, to characterize the host load fluctuation over a specific time period, a new metric is defined; the exponentially segmented pattern (ESP). The time interval is divided into segments of exponentially increasing lengths and the mean load is predicted over each segment. The evidence interval is also defined as the interval of which the recent samples used in the predictions are extracted from. Secondly, this segment pattern is transformed so that

each interval to be predicted is adjacent to the evidence window. An algorithm is used to first predict the mean load η_i, the predictor is a function of the length of each segment and the length of the evidence window. Secondly, the segment mean load ***l*** is calculated. The third step was to define nine feature of load fluctuation to be used with the Bayes model. These features are the mean load, the weighted mean load, fairness index, noise-decreased fairness index, type state, first-last load and N-segment pattern. Bayes theorem assumes that the features to be used are independent of each other, therefore linear correlation coefficients and Spearman's rank correlation coefficients are used to distinguish between the highly correlated features and low or non-correlated features. Based on this, a compatibility between the features is set such that two features are compatible if the correlation coefficient are less than 0.83 and non-compatible if the correlation coefficient is larger than 0.96. By using this compatibility, the features are divided into four groups where the elements in the same group cannot be used together. Fourth, the mean load prediction based on Bayes Model is constructed. The decision to choose the prediction value is made using two ways; the Naïve Bayes Classifier (N-BC) and the Minimized MSE (MMSE) based Bayes Classifier (MMSE-BC). Finally, the Bayes estimator was implemented as well as seven other prediction methods; namely, the last-state based method, the simple moving average method, the linear weighted moving average method, the exponential moving average method, the prior probability based method, the auto-regression method and the hybrid model that integrated the Kalman filter and Savitzky-Golay filter. The authors concluded that there are four selective features that can accurately characterize the mean load for the future; these features are the mean load, the fairness index, the type state and the first-last load. The MMSE-BC was found to be more

accurate, with higher success rate and lower MSE, than the N-BC, therefore it was the method used when comparing the different load prediction techniques.

Saripalli et al. [15] used a two-step method to predict the load of a web-based cloud platform called the Runaware enterprise cloud platform. First, the measured raw data is used to help represent the load trend. Second, a prediction algorithm was applied to the load trends. The load was tracked and represented using cubic spline interpolation [17]. An algorithm for hot spot detection is also used with the prediction algorithm. A hot spot of the cloud load is a sudden spike of traffic and the ability to detect is of vast importance in the cloud environment as it could be used to ensure the requirements of elasticity of the cloud are met. The hot spot detection is done using auto-correlation functions and linear least squares extrapolation as the basis [16]. The hotspot level H is defined as the maximum capacity of the server managing the load that is being studied; this value depends on the type of the application. The load prediction of the SaaS seasoning using the CS method was able to predict the load with a 25-75% margin of error and the authors state that it would still be a valuable tool to be used for resource provisioning during runtime as the cubic spline based load tracking provide high correlation among the load values [17].

Research has also been conducted on prediction of the load of mobile phones and the outsourcing the computational problems from the mobile to the cloud. Heo *et. al*. [14] proposed a user demand prediction method to predict the execution time and average volume of the transmitted application data of smart phones. The long-term logged application usage pattern from a virtual smartphone is statistically analyzed to be used in the prediction. Each smart phone saves battery power and uses the computational power of one exclusive VM

machine that is an image of the mobile operation system and applications. The application is executed within the VM and data is exchanged between the mobile and the VM using a 3G and a Wi-Fi connection environment. The prediction was done using double exponential smoothing with Holt's linear method. However, the prediction values were not compared to the actual values and only the trend prediction was documented.

2.5.3. Control Theory applications for cloud resource management

A resource management system could be designed by using concepts of the control theory; the objective function of the system representing multiple performance requirements and constraints could be expressed as the cost functions. The main components of the control system are: the inputs (sensors), the control system (monitors and actuators) and the outputs. The inputs could be but not limited to the workload of the system and admission control policies or the capacity allocation. The control system components are the used to estimate the performance measures and implement system policies. The output is the resource allocation to the application tasks requesting the resources. The control system could face instability, the adjustments of the control system should be only done after the system has stabilized and measurement of the time for stabilization and adaptation for an application should be done by the controller [5]. Additionally, the upper and lower thresholds should be set apart enough from each other as the workload fluctuation is very large and may cause instability of the system [5].

Zhang *et al.* [11] used both the control theory and game theory for dynamic service placement decisions in geographically distributed clouds. Figure 2.2 shows the control architecture for a single service provider. The system consists of: request routers also known as

redirectors that are responsible for redirecting the requests to the appropriate servers, monitoring modules ware used for collecting statistics of the demand received by the different request routers and the prices of the services offered by each data center. The other components of the system are the analysis and prediction module that is responsible of modeling the demand and the price changes and also for predicting the future values for them, and the resource controller that is responsible for dynamically adjusting the number of servers leased in each data center while satisfying the SLA requirements by minimizing the latency and resource rental cost. It also communicates with the request routers and informs them of the number of servers in each data center, the request routers in return will find the best assignment for the demands to the servers available. The goal of the system designed is to minimize the total cost of server allocation and reconfiguration cost while meeting the demand constraint, the data center capacity constraint and the SLA performance constraint. The resource management of the multiple providers is modeled as a multiplayer non cooperative game theory.

Another application of the control theory was used to control overloading of servers. As the cloud workloads fluctuates greatly, the chance of spikes or flash crowds in the load could not be prevented, this might cause the cloud servers to overload causing delay in the response time of the applications below acceptable levels. Guitart [23] proposed an overload control strategy in secure environments to deal with overload in web applications in SMP hosting platforms. The strategy involves an admission policy and dynamic resource provisioning. The admission control was based on SSL connections: accept the requests with the already initiated SLL connections before the requests with the new requested SSL connections.

The control system designed is shown in Figure 2.3 and it consists of the following components: eDragon CPU Manager (ECM) which distributes the resources among the different running applications using a scheduling policy. The applications and the ECM communicate to relay information to each other of the resources needed and the resources available respectively. This communication is done using a shared memory, and a new JAVA class was developed to achieve the allocation of the resources (i.e. processors) to the application by the ECM. The application would then apply an admission control policy using the eDragon Admission Control (EAC) component to limit the requests according to the number of the resources available given by the ECM, this limitation is done to prevent degrading the QoS. The admission control policy accepts a certain number of requests containing new SSL connections and the entire requests with existing SSL connections.

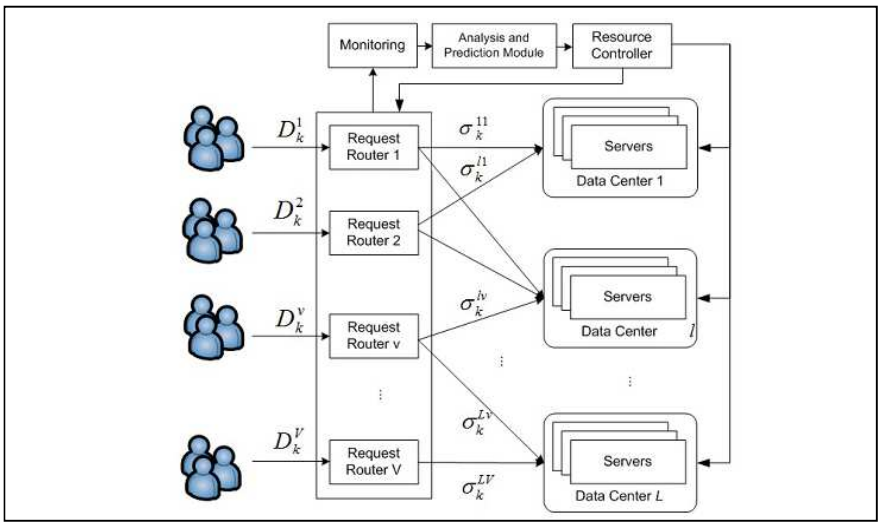

Figure 2.2 System Architecture for a single service provider [11]**.**

The ECM would then assign the requests to the resources using a scheduling policy that would be explained in the task scheduling section. The eDragon Load Monitor (ELM) component is added within the server that runs the web application, and it is responsible for

continuously monitoring the incoming secure connections to the server, it distinguished between the new SSL connections $O_i(k_{APL})$ and the resumed SSL connections $N_i(k_{APL})$ and counts them every sampling interval k_{APL}. Both ELM and EAC are responsible for monitoring the incoming SSL connections, where ELM estimates the resources requirements for the application, the EAC decides on the accepting of the new SLL connections. The average computation time of a resumed SLL CTO*i* connection and the average computation time of a new SLL connection CTN*i* were measured using static profiling of the application.

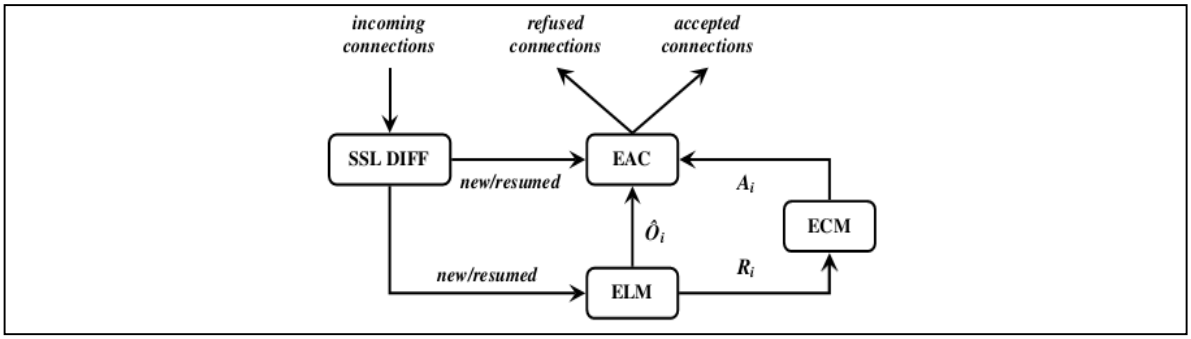

Figure 2.3 Components of the control strategy for dynamic provisioning of resources [23]**.**

The EAC calculates $AN_i(k_{APL})$, the maximum number of new SLL connections that the application i could accept without overloading and degrading the QoS. This value depends on the number of processors allocated to the application $Ai(k_{APL})$ and the computation time of resumed SLL connections that were accepted during k_{APL}. The rest of the SSL connections were refused. These calculations are done prior to accepting or negotiating any SSL connections; therefore the overhead caused by them is not noticeable. The results of this study showed that the control policy strategy allowed the server to attend to more users than a server that uses its own dynamic provisioning by refusing new SLL connections. This SSL refusal is also showed to be less than the timed out connection in the server than is not using the control strategy.

2.5.4. Data placement in the cloud

Data management in cloud computing is a challenge especially for some types of workflows i.e. scientific workflows that have enormous data amounts that reaches up to terabytes of size. Cloud data is distributed among different centers and the connections between them has bandwidth limitation that would put a constraint on their movement. Additionally, the distributed data sets have dependencies between them and oftentimes the application tasks needs to use more than one data set making the data set movement an inevitable process. The cost of the moving the datasets could be more than the cost of scheduling the tasks [18], hence the data sets need to be placed in the data centers in such a way that their movement will be minimized during application tasks executions.

Yuan *et al*. [18] proposed a matrix based k-means clustering strategy for data placement in scientific workflows. The data sets in scientific workflows were categorized according to their flexibility. The data sets were divided into fixed location data sets that could not be moved and flexible location data sets that could be moved. The inflexibility of the movement of data set in scientific workflow could be due to many reasons: some data might need to be on one location to be processed by specific equipment, some very large data sets could not be moved efficiently or the reason could be a matter of the ownership and limited access rights of the data set that could hinder it from being moved from a specific datacenter. The structure of the data was not considered.

The clustering of the data sets was done first by defining the dependency between the datasets. The datasets are dependent on each other if the same task used them. To execute a task, all the datasets need to be located in the same datacenter. By storing the dependent data

sets in the same data center, less movement of the datasets would be required and most of the datasets needed by the task were found in the same datacenter. Each dataset i is given two attributes: T_i a set of tasks that use this particular dataset and the s_i size of the dataset. The dependency between two datasets i and j is then defined as: $dependency_{ij} = Count\ (T_i \cap T_j)$ Where T_i is the task that will use the dataset i and T_j is the task that will use the dataset j. Based on this dependency, then the datasets were clustered in different data centers using k-means clustering strategy consisting of two stages build-time and runtime. The k-initial partitions representing the existing datasets were clustered into k datacenters for the k-means algorithm in the build time state. And the newly generated datasets during the workflow execution were clustered in k datacenters during the runtime stage according to their dependencies that were calculated dynamically.

During the build-time stage, a dependency matrix DM is setup and clustered. The clustering in the build-time stage was done using the Bond Energy Algorithm (BEA) which is a permutation algorithm used to group the similar items together by vertical partitioning of large tables. The dependency matrix DM was used as an input and a clustered dependency (CM) matrix was generated using the BEA. The second step of the build-time stage is partitioning and distribution of datasets. A recursive binary partitioning algorithm was developed and used to partition the clustered dependency matrix while trying to find the best data centers that have storage capacity matching the datasets' sizes. This is done by placing the datasets that have higher dependencies with each other together in one partition and lower dependencies with the datasets in the other partitions. Distribution of datasets is done by examining the datasets in the system for flexibility. Furthermore, the datacenters were assumed to have enough

storage to host all the application data in the system. The k-means clustering algorithm was used dynamically in the runtime stage to calculate the to place the generated data to one of the k data centers. Simulation of these algorithm showed the movement of the datasets were greatly reduced when using the build-time algorithm, however, the movements were not affected when using the run-time algorithm only because of the pre-allocation of the datasets to the wrong datasets due to it being prior to the task scheduling that was based on the data placement not their dependencies to the datasets. All the results show that the build-time and run-time algorithms when used together are found to reduce the datasets movement by 50.8% compared to the random situation. The simulation done in the different experiments only measured the data movements of datasets. No other parameters were measured to show the effectiveness of the algorithms in terms of execution cost or communication cost. In addition, the cost of placing the data in the different datacenters was not included in the study.

2.5.5. Task scheduling in the cloud

The problem of mapping tasks to distributed resources from an optimization perspective is an NP-hard problem, therefore, meta-heuristic techniques could be applied for solving the problem. The resource allocation mechanism is usually done using algorithms that will find the near optimal solution for an objective function. The objective function of the resource allocation algorithm is designed using different criteria and constraints depending on the aim of the optimization. The resource allocation algorithm will map jobs to resources either by maximizing or minimizing the objective function.

Scheduling workflows can be according to different objectives that vary from application to another. Some applications are data intensive applications while other are compute intensive.

The objective are either user-center and/or provider/centric. Provider centric objectives could be, but not limited to, any (or a combination) of the minimization of the following: total data transfer time/cost (communication cost), total execution time/cost and storage space usage. User- centric objectives might be defined as minimum amount of completion time or minimum cost to be procured by executing the job. When scheduling is done, different tasks will achieve different execution times and costs depending on the computer nodes that they are assigned to. These computer nodes will have different bandwidths between them, thus will cost different amount when transferring the data between them. Scheduling the tasks could be done by assigning the tasks to compute nodes that have less execution cost without disregarding the communication cost that will take effect due the dependency between the data and the tasks.

Pandey *et. al*. [24] presented a model for task scheduling in cloud computing to minimize the total execution cost and developed a heuristic algorithm using particle swarm optimization to assign the tasks to the compute resources. A Directed Acyclic Graph (DAG) was used to denote the application workflow. Particle Swarm optimization (PSO) is a self adaptive, global optimization technique. It optimizes the problem by using a population of candid solution. The heuristic algorithm used the PSO algorithm to map the tasks to the resources according to the provided objective function. The scheduling heuristic algorithm was designed as follows: first initiating of parameters was done by calculating the average computation cost of all tasks in all compute resources and calculating the average cost of (communication/size of data) between resources. Secondly, setting the task node weight w_{kj} as the average computation cost and setting edge weight $e_{k1,k2}$ as size of file transferred between tasks. The next step was to compute

PSO({ti}). Then the mapping of the tasks started, where all the ready tasks were mapped to resources according to the solution provided by PSO. After finishing the mapping, all the tasks were dispatched to be executed by the resources. The scheduler was designed to wait for the tasks status to be acquired and get all the new ready tasks, the ready task list was then updated. Additionally, the average cost of communication between resources was updated according to the current network load and the new PSO is calculated for the new ready tasks. This whole process was repeated until there were no more scheduled tasks. The performance metric that was used is the cost in cents of the complete execution of an application and transferring of data; the pricing was obtained from the AWS Amazon packages. The results obtained from this study showed that the algorithm performed better than the best selection algorithm that was used for comparison. However, the study does not take into account the disproportion between the pricing of the execution costs values on the computer resources and the values for the cost of transferring the data between these resources. The execution cost is far greater than the communication cost such that when calculating the cost the significance of the communication cost was trivial in the total cost of executing the tasks. This disproportion needs to be addressed in the cost function equation.

Guo *et. al* [25] had a similar target but different modelling of the system than Pandey [24]. The objective was to minimize the communication time and execution time to reduce data movement. A Task Interaction Graph (TIG) was used to denote the task scheduling. It is presented by G(V,E), where: V={1,2,…..n} is the set of tasks of an application and E={Cij} is the information exchange between the tasks. .

Minimizing *Total (M)* = $C_{exe}(M) + C_t(M)$ is the objective function. Particle Swarm optimization again was used achieve this goal. Results show than not only that converges faster than the other algorithms used for comparison, it also runs faster than those algorithms in a large scale. However, the assumption was made that the tasks won't take longer than an hour to complete their execution and the cost function didn't take into account the execution time of the tasks, which significantly effects the cost function of their model.

The scheduling policy in [24] concerns scheduling applications to processors; it considers an e-business indicator by giving customers different priority classes, e.g. Gold, Silver, or Bronze. Pi denoted a priority class that indicated a customer domain's priority in relation to other customer domains. And according to this differentiation the customer with higher priority would get better service than the one's with lower priority. In this policy, the ECM (eDragon CPU Manager) is responsible of distribution of resources to different applications. Each application i receives a number of processors $\mathbf{A_i(K_{ECM})}$ at every sampling interval K_{ECM}. $\mathbf{A_i(K_{ECM})}$ is proportional to application i's request of processors $\mathbf{R_i(K_{ECM})}$ and to the application's priority class (P_i). The scheduling policy also allowed a processor sharing mechanism to achieve higher resource utilization in the hosting platform. The granularity used in this sharing mechanism is 0.5 processor, and processors where distributed such that application i would share its processors if it was assigned all the processors it needed and application j assigned a number of processors less than the requested. In this case even if a fraction of 0.5 processor was not used by processor i, it could be assigned to application j if it was needed.

One of the cloud desired characteristics is its elasticity. Applications are able to grow and scale in the cloud, the growth is met by transitioning the application components from one

capacity configuration to another either by replication or migration [26]. Sharma *et. al*. [26] developed kingfisher, a cost-aware elasticity provisioning system that choses between replication and immigration when processing the application in the cloud. The choice is made by kingfisher such that most cost effective server configuration is achieved.

Due to the heterogeneity and virtualization which enables sharing of the infrastructure, the performance of the CPU is not stable and suffers from variation in terms of execution time [20]. This performance variation may cause the application to miss its deadline. As many of the scheduling and mapping algorithms depend on the VM execution times, the tasks may be delayed beyond expectations of the system which might affect the overall execution of the workflow and performance of the system. Rodriguez *et. al.* [20] presented a solution by designing a resource provisioning and scheduling algorithms based on deadline and cost to avoid the problems caused by performance degradation delay by encompassing the VM variation and VM boot time when computing the execution times for the tasks. Three calculations were presented; execution time of tasks, communication times between different computing units that are processing dependent tasks, and the processing times of the tasks which is the combination of the execution times and the communication times of the same tasks.

First, the provisioning is done using Particle Swarm Optimization PSO to find an optimal set of resources from a pool of available resources. The set of resources was selected if it meets the specified deadline while minimizing the total cost of executing all the tasks. Furthermore, the PSO used considered a heterogeneous infrastructure and more than one VM type was used. The PSO produced a map of task-resource pairs is produced and the next step was scheduling

the resources that were choosing for executing the tasks. Secondly, a scheduling algorithm was developed to calculate the start and end time of each task in the map. A schedule S =(R, M, TEC, TET) was produced were: R is the set of resources that indicates the resource ID along with its start and end lease time, M is set of task-resource maps each containing the task end and start time along with the assigned resource , TEC is the total execution cost of all the tasks in the selected resources and TET is the total execution time of the tasks on the selected resources. In addition, earliest deadline first algorithm is used to map tasks to running VMs. The workflow that was used to evaluate the algorithm is the scientific workflow and evaluation of the system showed that the proposed meta-heuristic PSO algorithm have given better results than the base algorithms in terms of meeting the deadline while maintaining the least cost. However, the authors have indicated that their algorithm is more time consuming than the other two heuristic based algorithms and rationalized that with the fact that it is an offline algorithm that produces better schedules.

Finally, after reviewing different resource provisioning and scheduling algorithms, a characterization could be made such that the different schemes can be divided from the perspective of cloud user and/ or cloud provider. It is evident in the literature that the provisioning either considers user-centric or provider-centric parameters. The user-centric provision takes into account parameters that affect the users such as location of the user, application deadline, and cost. While the provider-centric provision considers for example the CPU utilization and energy consumption by cloud machines.

Chapter 3

Context Aware Resource Allocation and Scheduling for Hybrid Mobile Cloud

A hybrid cloud consists of a public cloud and a private cloud or a number of them. Mobile applications access the hybrid cloud services using an API on the mobile device. A cloud service broker CSB works as an intermediary between the mobile clients and the cloud provider. A detailed description of the architecture of a context aware hybrid cloud system is presented in section 3.1. Further, a mathematical system model is presented in section 3.2 along with the system service model. In section 3.3, the mathematical formulation of the problem and the deadline-based algorithm used to solve the problem are explained.

3.1. Proposed System Architecture

Figure 3.1 shows the system architecture of the context aware hybrid cloud. The system consists of four main components: Cloud-Based Mobile Application, Mobile Cloud API, Cloud Service Broker (CSB), and the Cloud Services Provider (CSP).

Each component consists of a number of functional units. The cloud-based mobile application interacts with an API specifically designed to collect certain context information using the mobile device sensors and built-in functions. The details for the cloud-based application and the API is beyond the scope of this thesis. The following explanation applies to the remaining components, namely the CSB and the CSP.

3.1.1. Cloud Service Broker (CSB)

The CSB functions as an intermediary between the clients and the cloud services. When a mobile client request services from the cloud, the CSB will authenticate the client. In addition,

the CSB uses the context engine to record and monitor the client context information that is sent periodically by the client's Context API. CSB has a service discovery unit as shown in Figure 3.1, which is responsible for collecting and maintaining a record of the cloud services information.

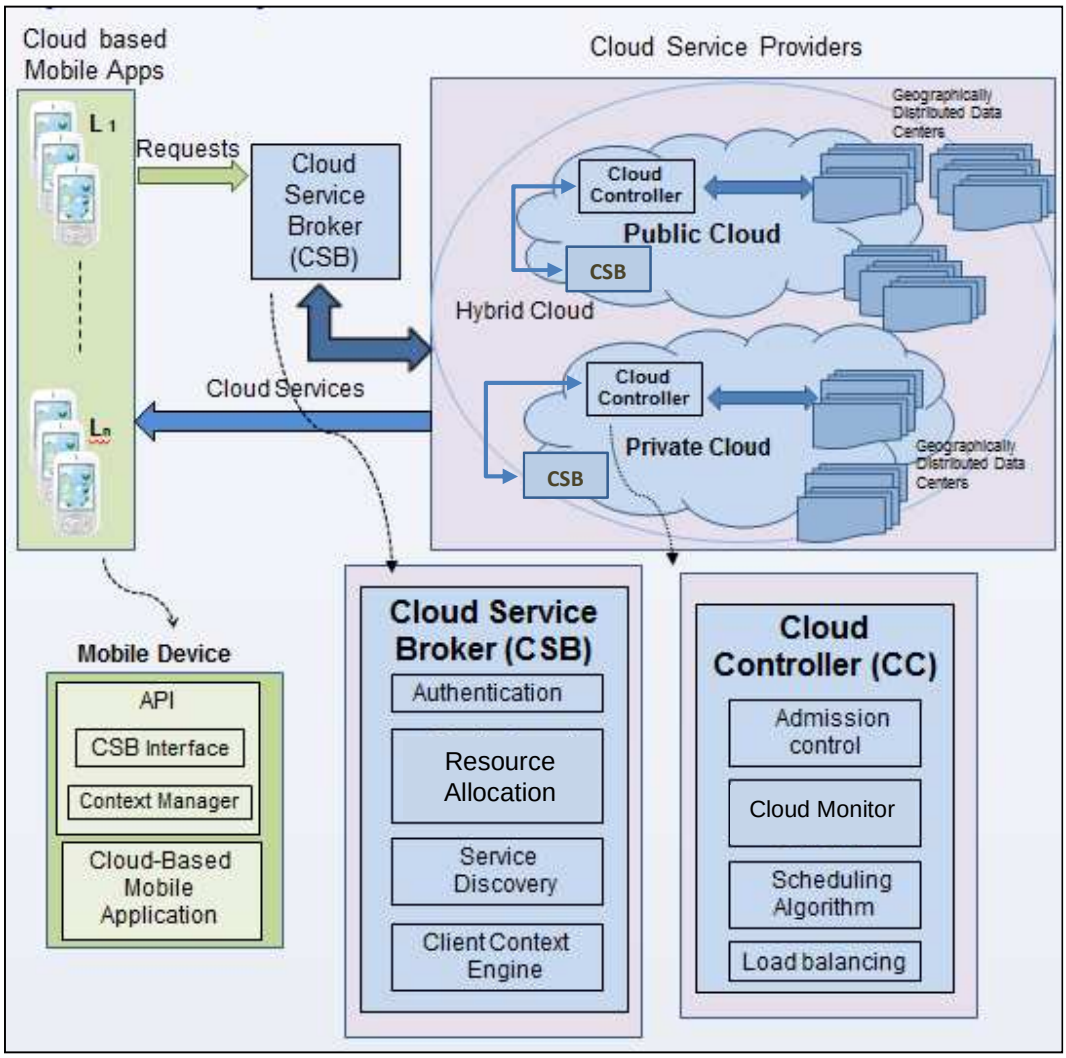

Figure 3.1. Context Aware Hybrid Cloud System Architecture.

The CSB receives the clients' requests with a set of constraints and the context information. It queries the cloud provider for resources that guarantee meeting SLA requirements. The

Service Level Agreement (SLA) between the cloud clients and the cloud provider ensure that the client receives cloud services that satisfy the Quality of Service (QoS) requested by the clients.

3.1.2. Cloud Controller CC

Each cloud service provider has one Cloud Controller CC for each cloud service cluster. The CC is responsible for controlling the volume of requests to be allocated resources in the cloud during admission control. Resource allocation is done by mapping the requests to resources while minimizing response time experienced by the client and the cost of running the resource in the cloud. In our proposed scheme, client contexts include location, connection quality, and mobile energy level that is used to make the allocation decision, as explained in the next section. The CC scheduler receives updates if there is a change in the client context, and the allocation of the resources to the requests is modified according to the new context information.

Figure 3.2 shows the service model of the proposed system. The mobile client application uses the mobile cloud API to gather the relevant context information and send it to the CSB. The CSB stores this information in the client context profile and monitors the new incoming context information from clients. The context analyzer evaluates the context information and triggers an event to change the allocation of resources to the jobs that are scheduled to run in the VMs located in cloud datacenters machines. In our scheme, the trigger is designed to set off when the context analyzer detects a change in either the connection quality or the energy level of the mobile device. In addition, the provider performs resource monitoring and discovery by finding available idle resources suitable for the request of the clients. The resources are monitored to determine their availability. Acquiring and releasing resources on demand is one

of the cloud providers' responsibilities. The jobs are processed in batches. After discovering the resources, they are assigned to the jobs in the list using a meta-heuristic algorithm, which in our scheme is simulated annealing algorithm. A Scheduling algorithm is an abstract model to define the order of execution for a set of tasks. The scheduler finds the needed resources for these tasks according to the resource-task map generated by the allocation algorithm. Finally, the VMs are scheduled to run on the physical machine and a VM manager is responsible for monitoring the VMs and launching and killing the VMs instances according to the demand.

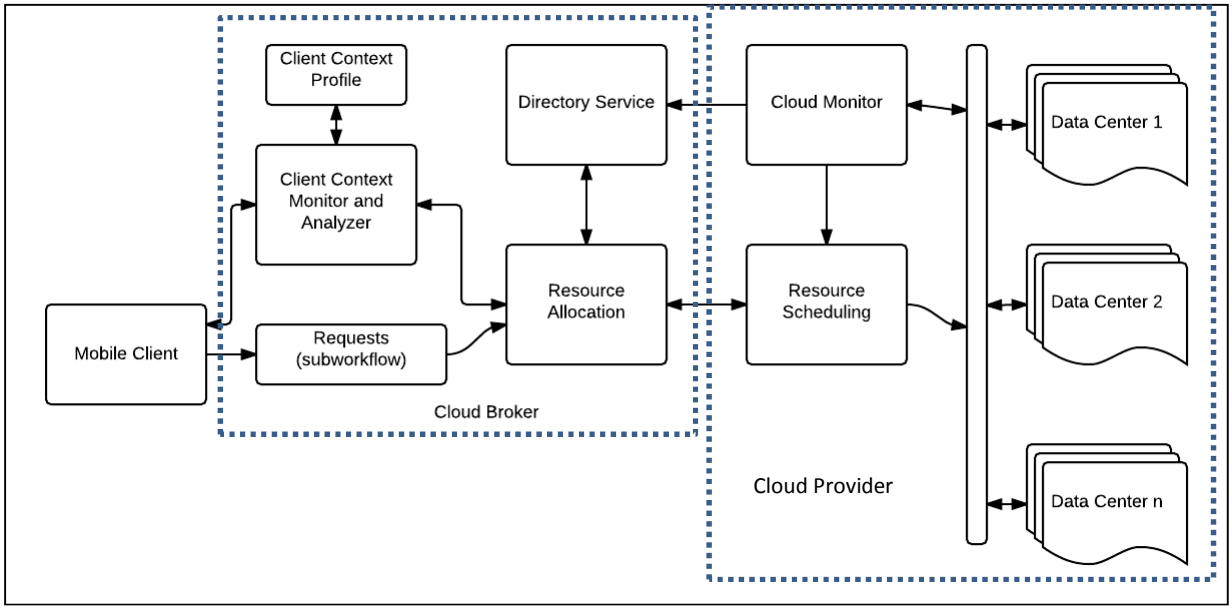

Figure 3.2 Service Model of the Context Aware Cloud-Based System.

In this work it is assumed that the requests are already collected from mobile clients. These requests constitute a job that contains one or more tasks which are indexed to show their order in the job. The indexed tasks are paired with their job IDs.

A job consists of more than one tasks. The job type defines the way the tasks in a job are executed. Sequential-task job type means that a job contains tasks that are executed one by one and that the execution of the tasks never overlaps. The second type is the batch-task job

where the tasks are executed in parallel and one job could not contain less than two tasks. The mix-mode type is a combination of sequential-tasks and batch-tasks. In addition to the list of the jobs to be executed, the requests are embedded with user context information as mentioned previously. This information includes user's current location, power level of mobile device, expected response time (deadline), network information such as connection type, delay tolerance.

3.2. Proposed System Model

Mobile devices are resource constrained, and they have limited computing and storage capabilities. By offloading computation intensive applications to cloud, mobile devices can use the unlimited resources offered by cloud providers. The cloud services offered to the mobile users could be categorized as applications-specific services known as PaaS and mobile computation offloading services known as IaaS. The objective is to enhance the mobile user experience by reducing computation time and battery consumption and offering services to enhance the mobile device capabilities.

The most important issues for mobile users using the cloud are network connectivity, amount of data transmitted, mobile device energy, and bandwidth. Mobile devices connect to the cloud via internet either by using the cellular/satellite network connections through base stations, or by using Wi-Fi connections through access points.

It is the responsibility of the application developer to design the cloud-based mobile application and decide about partitioning of the application such that some parts are executed locally in the mobile device and some parts are executed remotely in the cloud. The cloud-based mobile application workflow could be modeled in a method or module granularity [27].

In [28], the application partitioning is achieved by modeling the application at the functional level rather than at the class or object level, while in [29] methods and/or classes that are expected to be offloaded to the server as *remotable* methods which are the methods that could be executed remotely without disrupting the functionality of the application.

One scheme for the developers to estimate the application requirements and measure their CPU requirements is through the simulator presented in [30].

Mobile application partitioning is beyond the scope of this research and we assume that the application is already partitioned and the mobile cloud API sends the request as an indexed task graph associated with a unique Job ID to be executed in the cloud. We also assume that the cloud provider has a copy of the code to be executed; which is downloaded once when the application is launched for the first time. Hence, it involves onetime communication to the cloud to reduce size of the request. Once the request arrives in the cloud service broker, the unique job ID is linked with the appropriate code, and the code along with the data submitted with the request is launched on a suitable virtual machine.

3.2. 1. Mobile Application Model

The cloud based mobile application consists of a set of tasks to be executed in the mobile device and another set to be executed in the cloud. It consists of a combination of sequential and concurrent tasks.

The mobile application workflow W = (T, E) is modeled as a Directed Acyclic Graph (DAG), where T = {t1, t_2, t_3,… t_n} is the set of tasks and E is a set of directed edges. An example of DAG is shown in Figure 3.3. If there is a data dependency between t_i and t_j, then an edge $e_{ij} \in E$ between these two tasks indicates this dependency, and t_i is said to be the parent of the child t_j.

The data output size of task t_i is denoted by DS_{out}^{ti} . In addition, each workflow is associated with a deadline called W_d based on the user's desired QoS. The workflow execution time is constrained by W_d. The notation T represents one job to be offloaded and executed in the cloud and the tasks are indexed as *i* = {1, 2, …. n}.

3.2.2. Mobile Client Context Model

The context manager in the mobile device collects data from different sensors, packages them into a context, and transfers the mobile context to the CSB. It also monitors the context information and updates the cloud broker if any change in the context occurs. In our scheme, the context includes energy level of the device, its location, and network type (data rate).

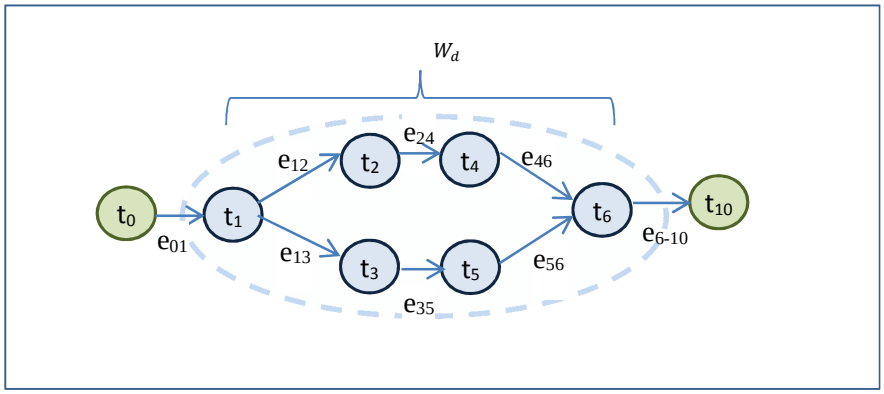

Figure 3.3. Example of a Task Directed Graph of a Sub Workflow.

Mobile Device Connection and Data Rate Model:

Modeling of connection type could be done by using some measures to model the quality of the channel from the user's context perspective. These measurements include: data rate (Bandwidth), SNR and request round-trip times. Two connection types are used: 3G/4G cellular connection and Wi-Fi (WLAN) connection. Typical data rates for 3G interface DR_{3G} is 24 Mbps whereas that for Wi-Fi DR_{wifi} is 54 Mbps.

Mobile Device Energy:

The mobile device measures its energy level $\mathcal{E}_{local}$ periodically and the user context manager on the API embeds this information in its context transfer to the CSB. The energy level sent to the cloud represents the residual energy of the device at the time of transfer.

Mobile devices consume energy while transmitting and thus the energy will decrease after the context is transferred to the CSB. An estimation of the mobile energy consumption could be calculated in the cloud to estimate the expected mobile energy level. The current mobile energy level is then calculated as:

$$\mathrm{E}_c = \mathrm{E}_{device} - \mathrm{E}_t \quad (1)$$

Where E_{device} is the energy level of the mobile device and E_t is the energy consumption due to transmission.

Mobile Location:

The physical space in which Mobile clients and cloud resource are located is divided into *k* zones $\{z_1, z_2, \ldots., z_k\}$. One zone is defined as an area where a cloud data center is located at the center of the zone and mobile clients are spread around the data center. Each service request is assigned to a zone according to the location it is originated from.

$$L = \{l_{z_1}, l_{z_2}, \ldots\ldots, l_{z_k}\} \quad (2)$$

3.2.3. Mobile Client Request Model

A set of mobile users $U = \{u_1, u_2, \ldots \ldots u_x\}$ receives services from the cloud offering services. Each user makes a service request, which is a job that consists of a set of tasks, T = { t_1, t_2, t_3,… t_n}. Where *n* is the total number of tasks contained in one job submitted by the user.

Each job includes a CPU and memory requirements of each task. These requirements could be a measured value or defined a priori by the application developer. We assume that the requirements are available to the mobile application API and they are sent with the request. The cloud provider has to ensure that the task is assigned to a resource that fulfills its resource requirements. In addition to the computing resource requirements, the request contains time of submission, unique user and job identifications. The job of a user u is modeled as:

$$J_u = (\tau \ , u_{ID} \, , Job_{ID}, T) \tag{3}$$

Where task *i* of T is modeled as: $t_i = (t_i^{size}, DS_{out}^{ti} \, , CPU, MEM)$ (4)

In addition, the context of user u_i , as specified below, is periodically monitored and sent to the broker:

$$Context_u = \ (\omega, L_u, E_{device}, \lambda) \tag{5}$$

Where:

t_i	Task *indexed i*
τ	Job Submission time
CPU	The requested number of CPU cores
MEM	The requested amount of memory
W_d	The Job's whole duration
t_i^{size}	Size of the task *i* in millions instructions in task *i*
DS_{out}^{ti}	Task output data size
L_u	Location of user *u*
ω	Time the user context information is logged
E_{device}	Energy level of mobile device
λ	Data Rate of the mobile device: Wi-Fi connection or 3G connection

The context engine in CSB periodically monitors context information from user devices, which is shown in Figure 3.4 as { C_0 , C_1 , C_2 ,C_3 …..}. The figure also shows the job submission time when the cloud receives requests from users to launch their jobs.

Figure 3.4 shows the two timelines are aligned, that is the job submission time related to context period. . Each job *J* is associated with a context *C,* which refers to the context that is used by the broker to allocate and schedule the required resources for the job. For example, starting at time τ_0 , job J_0 is associated with context C_0. Similarly, at τ_1, job J_1 is associated with context C_2 and so on. Only the context attached with the jobs are used in our allocation and scheduling scheme.

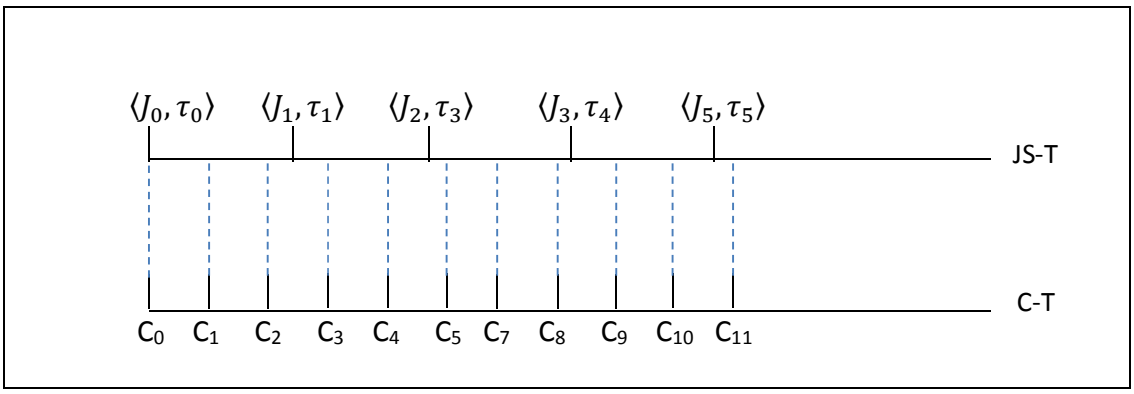

Figure 3.4 User Jobs Schedule Timeline (JS-T) vs. User Context Timeline (C-T).

3.3. Cloud System Model

The cloud service provider offers Infrastructure as a Service. The hybrid cloud service, H, is composed of private cloud services S_{prv} and public cloud services S_{pblc}.

$$H: S_{prv} \cup S_{pblc} \tag{6}$$

The cloud services and associated resources are identified by Virtual Machines of different types. A VM_i is a virtual machine of type *i* that has a computing capacity of P_{VMi} , memory

capacity of μ_{VMi} and cost per unit of time C_{VMi} . We assume that the VMs have sufficient memories to execute mobile applications workflows.

3.3.1 Execution time of a task

The size of task *i* submitted by a user to the clouds is assumed in terms of MI (Million Instructions). The execution time of task *i* on VM_j is calculated using the task size t_i^{size} as shown in the equation below, where deg_{VMj} is a performance degradation percentage used to model performance variation of the computing capacity of VM_j.

$$Ex_{ti}^{VMj} = \frac{t_i^{size}}{P_{VMj} * \left(1 - deg_{VMj}\right)} \tag{7}$$

s.t $\sum_i^n Ex_{ti}^{VMj} \leq W_d ; \forall\, t_i \in W(T, E)$ (8)

3.3.2. Data Transmission Time

The data transfer or transmission time between two tasks t_i and t_j is calculated using the equation:

$$TT_{e_{ij}} = \frac{DS_{out}^{ti}}{\beta} \tag{9}$$

Where β is the bandwidth between the VMs when calculating the transfer time between two tasks t_i and t_j executing on two different VMs.

3.3.3. Total Processing Time

Finally, the total processing time of a task in a VM is the sum of its execution time on a VM and the total transmission times it takes the task to transmit its output data to all its children tasks.

$$PT_{ti} = Ex_{ti} + \left(\sum_1^p TT_{e_{ij}} * \varphi_p\right) \tag{10}$$

Where: *p* is the number of tasks dependent on t_i, and φ_p is 0 whenever t_i and t_j run on the same VM. And the total processing time of all the tasks is $\sum_{i=1}^n PT_{ti}$.

3.3.4. Turnaround time of Request TAT_R

The turnaround time of a service request launched from the mobile device is defined as the duration from the submission time of the request to the time when the results are back to the device from the cloud provider. It is also referred to as latency [7].

If DR is the data rate of the mobile device at the submission of the job to the cloud, and t_j is the last task executed in the cloud, then the transmission time of the result to the mobile device is:

$$TT_{Mob}^{Cl} = \frac{DS_{tj}^{out}}{DR} \quad (11)$$

The total turnaround time of the job is the sum of the data transmission time from the mobile to the cloud, the total execution time of all the tasks in the cloud, and the transmission time of the result to the device.

$$TAT_R = \sum_{i=1}^{n} PT_{ti} + TT_{Mob}^{Cl} + TT_{Cl}^{Mob} \quad (12)$$

The data rate that a mobile device varies due to mobility and wireless channel condition. For example, a mobile device may move from one network type (e.g. cellular) to another (e.g. WiFi), or it may move from one cell to another within the same network. When the mobile gets high data rate connection, the task's execution could be slowed down without major noticeable change in user experience. This can offer cloud provider an opportunity to reduce its cost by exploiting the elongated deadline. Alternatively, when the data rate is low, execution times in the cloud have to be faster to make up for the lost time due the communication delay. This concept is facilitated by introducing the slack time [33], as explained and computed below.

3.3.5. Slack Time (float time)

The workflow has a deadline W_d that needs to be met when executing the workflow tasks in the cloud. The turnaround time of the workflow *TAT* should be equal to or less than W_d. The difference between W_d and *TAT* is called slack time T_{slack} . If the slack time is small, than the workflow needs to be executed closer to the deadline.

$$T_{slack} = W_d - TAT \tag{13}$$

Alternatively, if the slack time is large, than the cloud can slow down the execution to reduce computing and resource usage cost. The slowdown process is further explained in section 3.4.

3.3.6. Total Processing Cost

A task is executed in one VM, and the user is billed on bases of VM usage time.

Assume C_{VMi} is the cost of using VM_i for a time unit Θ . The cost of executing t_j for a period θ_{t_j} time units on VM_i is :

$C_{t_j} = \left(C_{VM_i} * \theta_{t_j}\right)$ Where $\theta \leq \Theta$ and is calculated in multiples of Θ.

Hence, the total cost for executing *n* of tasks is:

$$C_{Total} = \sum_{j=1}^{n} C_{t_j} = \sum_{j=1}^{n} (C_{VM_i} * \theta_{t_j}) \tag{14}$$

$s.t.\ C_{Total} \leq C_{U\_Budget}$; Where C_{U_Budget} is the price the consumer pays to the cloud provider.

3.4. Problem Statement

The objective of this work is to exploit the slack time in reducing the cost of execution of and resource allocation to a job in the cloud. The slack time is computed based on the changes in the user context, more specifically energy level and connection data rate. We develop an adaptive approach to integrated resource allocation and scheduling in cloud offering IaaS. The

slack time provides adaptive control. First, we focus on finding a schedule S for all the tasks in the workflow to execute on cloud servers such that the total execution cost is minimized while meeting the workflow deadline W_d. The CSB periodically monitors user context information, more specifically the connection data rate, the energy level of the mobile device and the location of the user. It defines thresholds for context data and updates the cloud controller if the data exceeds the threshold. The cloud controller may revise resource allocation and scheduling.

The schedule is defined similar to *Rodgriguez et al.* [20]. The schedule S = (R, M, C_{Total}, TET, TAT) contains the best suitable set of resources R ={r_1, r_2, …r_k} to execute all the tasks in T. The schedule defines this assignment by a set of maps M where each map m in M is $m_{ti}^{rj} = (t_i, r_j, ST_{ti}, ET_{ti})$, where ST_{ti}, ET_{ti} are the expected start execution time and expected end execution time respectively for the task t_i. Furthermore, schedule S also includes the workflow's total execution cost C_{Total}, total execution time *TET*, and the Turnaround time *TAT*.

Each resource in *R*, is defined as $r_i = \left(VM_{r\,i}, C_{VM_{ri}}, LST_{r\,i}, LET_{r\,i}\right)$. The terms, $LST_{r\,i}, LET_{r\,i}$ are the lease start time and the lease end time of resource $r_{i,}$, respectively. Also, for each task assigned to a resource, the expected start time is the lease start time of the resource if the task has no parent tasks. If the task has one or more parent tasks then the start time of the task is the maximum of the largest end time of all the parents' tasks and the lease start time for the resource. In addition, the expected end time is defined using PT_{ti} in the equation below.

$$ET_{ti} = PT_{ti} - ST_{ti} \tag{15}$$

The total execution time *TET* is the maximum of the expected end times of all the tasks' end times in the workflow.

$$TET = \max\{ ET_{ti} : t_i \in T\} \quad (16)$$

The problem could be defined as finding a schedule S such that:

$$Minimize\ C_{Total} \quad (17)$$

$$s.t.\ TAT \leq W_d \quad (18)$$

The schedule is updated if a new slack time is calculated when the user context changes, for example the new energy level of the mobile device, which recalculates a new deadline.

3.5. Resource Allocation and Scheduling

Each workflow that a user sends contains a number of tasks to be allocated resources for their execution. Resource allocation is a process of locating the most suitable set of resources (VMs) from a pool of resources for the specified number of tasks. The selection of the resources is done such that the total cost of the resources allocated is minimized and the deadline of executing the workflow is met. The algorithm used for resource allocation in this work is based on the Simulated Annealing Algorithm.

3.5.1. Resource Allocation Using Simulated Annealing Algorithm (SA)

Simulated annealing is based on the idea of "melting" a system with a very high temperature then cooling the system gradually until the point of "freeze" [34]. The system starts with a very high temperature and the cooling is done in small steps of lowering the temperature. The pseudo code of SA algorithm is shown in Figure 3.5 [35]. The first step is to initialize the system parameters; in this case the set of tasks to be assigned and the initial set of resources residing in the resource pool, which are defined using a Vector M known as the map. The vector M defines a mapping of resource to task such that the row index indicates the task index and the columns value is the resource ID. In the algorithm, the initial map is generated by selecting

resources randomly for each task. The constants t_{max} and t_{min} are defined. The variable τ is the temperature that cools down until it reaches the freezing temperature t_{min}.

3.5.2. Task schedule generating algorithm

A schedule specifies the start time and the end time of the execution of each task on the resource allocated for the task. It also specifies the start lease time and the end lease time of each resource allocated to a task. We use Earliest Deadline First scheduling algorithm that is based on the work of *Rodriguez et. al.* [20]. Figure 3.6 shows the pseudo code for the schedule generating algorithm. The algorithm starts with a set of tasks T of a workflow and the corresponding set of resources R_{init}. It keeps track of resource allocation to tasks in a pool of maps, and the allocated resources in a pool of resources.

1. Initialization: initial map M= M_0 (randomally generated);
Initial temperature τ = t_{max}(50), end temperature t_{min}(1).
2. While (τ > t_{min}) do loop:

    a) Get a new solution M_1 by the following way:
        i. Select two different resources r_1 and r_2 randomly.
        ii. Select a task t_1 scheduled to r_1 in M_0 randomly.
        iii. Map the task t_1 to resource r_2.

    b) Calculate the probability p= Math.min (1.0, exp (- (f(V1) -f(V0)/t)).

    c) Generate a random number *Nmbr* from 0.0 to 1.0(not included)
        if *Nmbr* <*p* then M = M_1

    d) Modify the temperature τ = 0.9* τ.

3. Return M

Figure 3.5. Pseudo Code for Simulated Annealing Algorithm[35] **used for resource allocation in Figure 3.7.**

The algorithm also calculates two matrices: task execution matrix and data transfer matrix. The execution matrix shows the execution time of the task on the resource such that the rows

indices are the task IDs and the columns indices are the resource IDs. The data transfer matrix shows the transmission time between two tasks when they are scheduled in two different VMs. The task with no parent starts as soon as the resource becomes ready. The task with a parent starts when both the parent finishes its execution and the resource is ready for the task. Hence, its start time is after whatever takes longer, the execution time of the parent or time when the VM becomes ready. The data transfer time between the task and its parents is calculated only when the two tasks are assigned to VMs on different physical machines. Then, the processing time is calculated as the sum of the transfer times and the execution time of the task. This processing time is used to calculate the end time of the task as in step number 4.5. Once all the information needed for a task are calculated, a map is constructed that includes the task, resource to execute the task, start time of the task and end time of the task. The resource is added to the pool of resources if it is not in the pool, and the lease end time of the task is calculated as in 4.8. The final two steps of the algorithm calculates the total cost of executing all the tasks that are assigned resources TEC, total execution time of all the tasks in the workflow TET, and the total turnaround time TAT. The simulated annealing algorithm choses the resources that satisfy the TAT constraint while minimizing the total cost of execution. Figure 3.4 shows the flow chart of the modified simulated annealing algorithm that uses the deadline based task scheduling algorithm. The objective function is calculated according to the TEC obtained from the scheduling algorithm. The constraint TAT < W_d has to be satisfied to choose a schedule otherwise the schedule is discarded.

Adapting the system to the new slack time:

- The slack time is calculated periodically after receiving a change in context reading, such that the $T_{slack} = W_d - TAT$.
- If the slack time increases by a margin Δ, then the start time of the task could be delayed. This may release a resource that can be given to a long awaited task belonging to another workflow. This arrangement can work only for the workflows that are executed in the same datacenter.

Adapting the system to the new energy level:

- If the context received from a mobile device shows that it is running low on energy, which is not sufficient for the mobile to receive the result back from the cloud, then the scheduler can put the user jobs on hold. The user context is periodically monitored and when the user regains the necessary energy level, then the workflow will be resumed and scheduled. Descheduling a task frees up the resource assigned to the task that can be used to reduce the wait time of another workflow.

// Algorithm Schedule (M)
Input: M
Output: Schedule S = (R,M,TEC,TET,TAT)

Where R= $\{r_1, r_2 \ldots . r_k\}$, r =$\{ r_j$, LST $_{rj}$, LET $_{rj}$ $\}$

M = $\{ m_{t1}^{rj}, , \ldots \ m_{tn}^{rk}\}$, $m_{ti}^{rj} = \{t_i, r_j, ST_{ti}, ET_{ti}\}$

1. Initialize all schedule variables
2. Calculate $ExeTime[|T| * |R_{int}|]$
3. Calculate $TrasnferTimes[|T| * |T|]$
4. **For** each task $t_i \in$ T assigned to resource $r_j \in R_{int}$ from M
   - **4.1. If** t_i is independent (no parent)
     $$ST_{ti} = LET_{rj}$$
     else
     $$ET_{max} = maxEndTime_{tp}(ET_{tp} : t_p \in parent(t_i))$$
     $$ST_{ti} = max(ET_{max}, LET_{rj})$$
     End If
   - **4.2.** exe = $ExeTime[t_i][r_j]$
   - **4.3. for each** child $\boldsymbol{t_c}$ of $\boldsymbol{t_i}$
     $$transfer = TrasnferTimes[t_i][t_c]$$
     end for each
   - **4.4.** $PT_{ti}^{rj} = exe + transfer$
   - **4.5.** $ET_{ti} = PT_{ti}^{rj} - ST_{ti}$
   - **4.6.** $m_{ti}^{rj} = (t_i, r_j, ST_{ti}, ET_{ti})$
   - **4.7. if** $r_j \notin R$
     $$LST_{rj} = max(ST_{\boldsymbol{ti}}, bootTime)$$
     $$R = R \cup \{r_j\}$$
     End If
   - **4.8.** $LET_{rj} = PT_{ti}^{rj} + LST_{rj}$
5. Calculate TEC
6. Calculate TET, TAT
7. Schedule S =(R, M, TEC, TET, TAT)

Figure 3.6 Schedule Algorithm

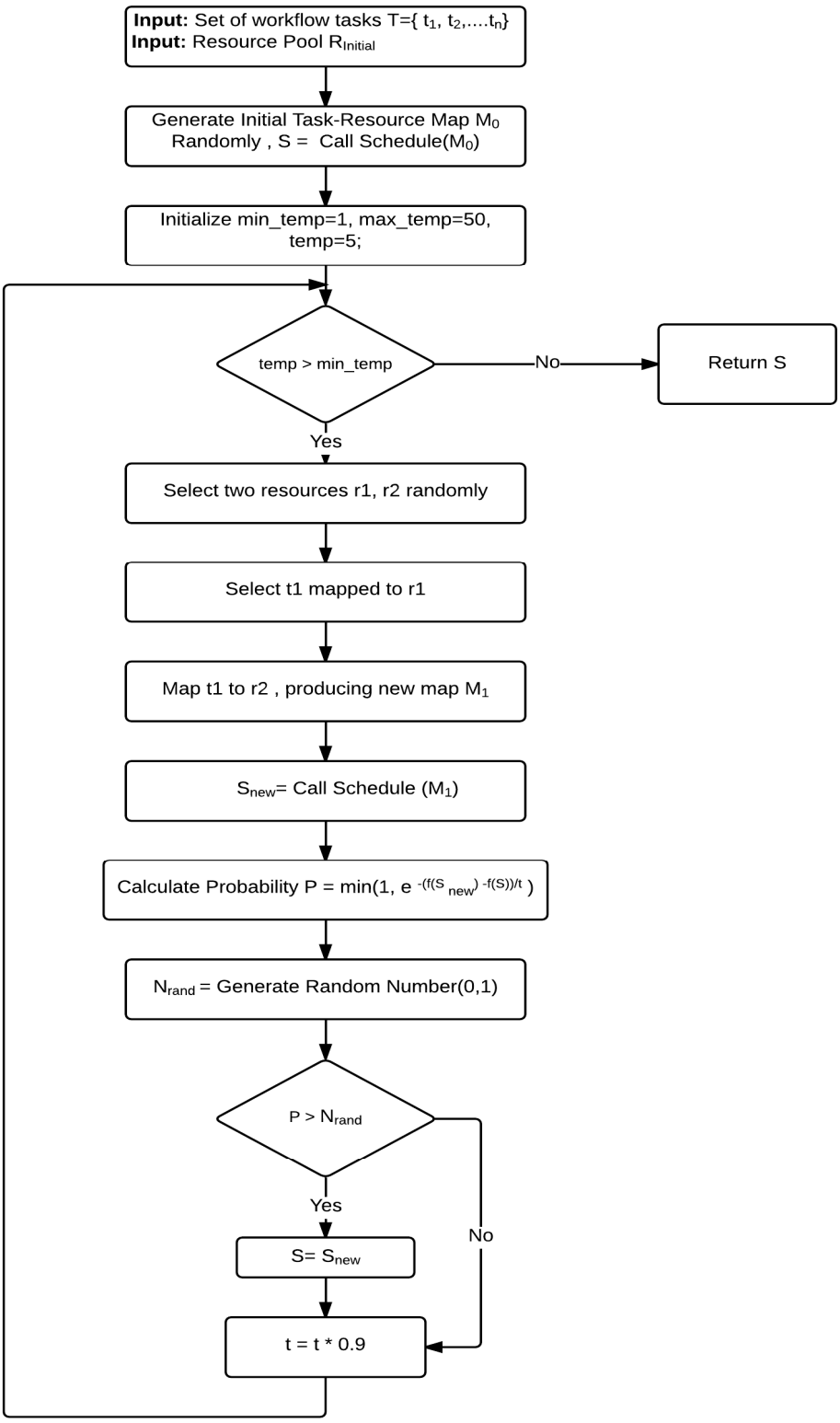

Figure 3.7 Flow Chart of the Simulated Annealing- Deadline Based Algorithm (SA-DB).

Chapter 4
Performance Evaluation

In this chapter, we discuss simulation-based performance study of our proposed scheme. First, the cloud workload used in the experiments is discussed, which is the trace of workloads running on Google compute cells provided by Google Inc. The traces are data from a cell of 12,000 machines over about a month-long period in May 2011 [36]. These traces are first studied and then filtered to extract the required fields to be used in the system. A Java program is written to extract the information. The tables were read and the related fields were extracted and written into new tables to be used in our experiments of the proposed system.

The second section is about the simulation of the proposed cloud system. The simulation was done using a discrete event simulator called CloudSim. CloudSim Toolkit software is developed by The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia and it is released as open source under the LGPL license. It is a java library that provides a simulation framework for enabling the modeling, simulation, and experimentation of cloud computing infrastructures and cloud application services.

4.1 Cloud Workload

4.1.1. Google cluster load

Google Inc. released a load trace [3] of more than 12000 machines, for a period of one month of May 2011. The data is available online in the form of Google buckets that can be downloaded using *gsutil* tool or manually, it consists of 39 GB of trace data. There are 6 folders each containing file(s) in the csv format. The folders are machine events, machine attributes, job events, task events, task constraints and task usage folder. All the folders contain 500 files

except for the machine attribute folder and the machine event folder, both of which contains only one file. A detailed explanation of each table is provided by Google in a schema csv file[36]. Table 4.1 shows the fields for the tables in each folder. The fields in gray are the fields we used to extract the cloud workload that will be used in this thesis simulation.

Table 4.1. Machine Events Table
1. Time Stamp
2. Machine ID
3. Event Type
4. Platform ID
5. Capacity: CPU
6. Capacity: Memory

Table 4.2. Machine Attributes Table
1. Time Stamp
2. Machine ID
3. Attribute Name
4. Attribute Value
5. Attribute Deleted

Table 4.3. Job Events Table
1. Time Stamp
2. Missing Info
3. Job ID
4. Event Type
5. User Name
6. Scheduling Class
7. Job Name
8. Logical Job Name

Table 4.4. Task Constraints Table
1. Time Stamp
2. Job ID
3. Task Index
4. Attribute Name (Machine Attribute)
5. Comparison Operator
6. Attribute Value

Table 4.5. Task Events Table
1. Time Stamp
2. Missing Info
3. Job ID
4. Task Index – within the job
5. Machine ID
6. Event Type
7. User Name
8. Scheduling Class
9. Priority
10. Resource Request for CPU cores
11. Resource Request for RAM
12. Resource Request for Disk Space
13. Different Machine Constraints

4.1.2. Google Cluster Trace Analysis and Extraction

Table 4.2 shows the machine attributes table. There are 1048578 fields entries in the machine attribute table and the fields titles are as follows: time stamp, machine ID, machine attribute, machine value, attribute deleted. There are 12583 machine attributes that starts with the unique name of "**GK**AYWlOFlntxaxF". In an analysis of the Google trace, *Reiss et. al* [37] concluded that the GK attribute is used as a machine location identifier. There are 253 of

the users who specified constraints on the machines and 17% of those users used the GK attribute in their constraint [37].

In this work, the users that have specified the GK attribute were extracted from the tables and collected in one table to construct the user set. These are the users who have specified the physical locations of the machines they want their jobs and tasks to be placed in.

Table 4.4 shows the task constraint table. The fields are: Time stamp, Job ID, Task index, Comparator operator, Attribute Name, Attribute Value. Records that contained "GK" in their attribute name field were extracted from the 500 task constraint tables. The next step was extracting the records with unique pairs of Job ID and Task Index field values, resulting in 500 tables that were then combined and the repeated fields were omitted.

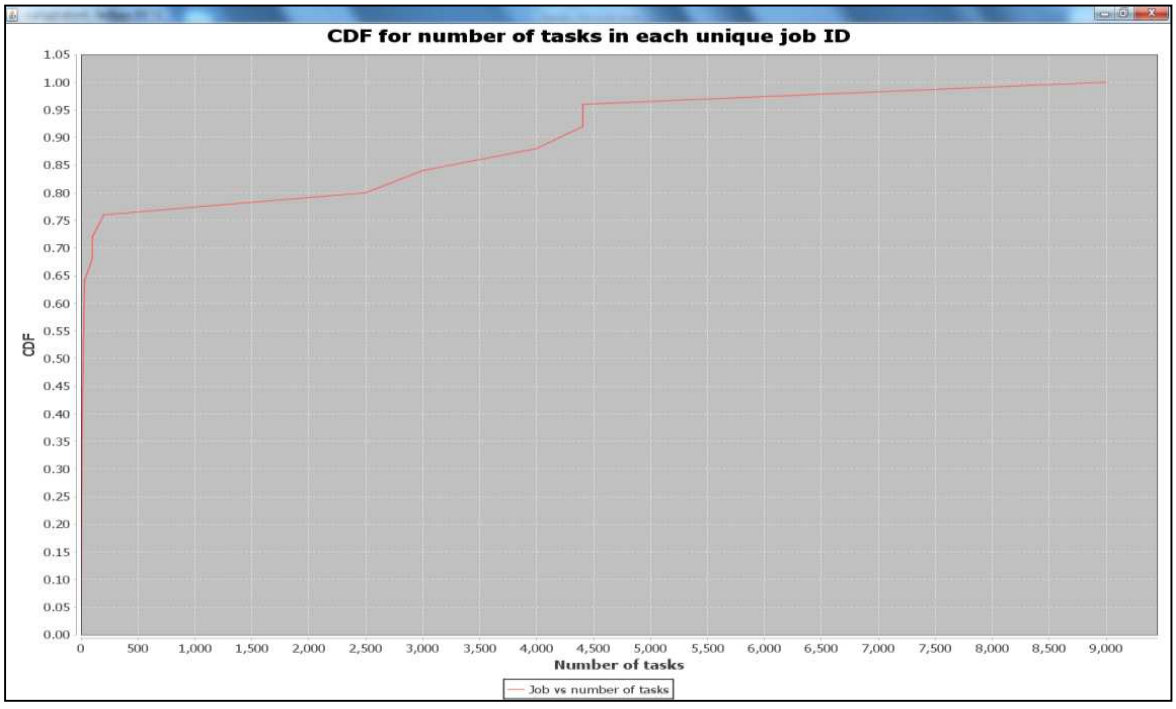

Figure 4.1. Cumulative Distribution Function (CDF) for number of tasks in each unique Job ID.

In Google load, a user launches a set of jobs while each job is a workflow that includes one more tasks. We count number of jobs a user has launched by extracting the unique job Id and task index pairs from the task constraints tables, which in turn is accomplished by searching the maximum value of the task index field for each unique Job ID. The results show that the number of unique job ids is 402. Also it shows that the number of combined job task pairs is 94128 pairs. Of these pairs there are 42373 different job task pairs that consist of only one task. Figure 4.1 shows the CDF of the number of tasks in each unique job ID, it shows that 75% of jobs consist of less than 500 tasks. In addition, the job/task events table include any jobs that are active (Running) or eligible to run but waiting to be scheduled (pending) at any point in the trace.

For each unique user name, the total number of requests (tasks) submitted by the user is the total number of unique entries in the task events table. The requests per user are counted and Figure 4.2 shows the Cumulative Distribution Function CDF of tasks per user. It shows that more than 87% of users have 200 tasks or less. According to this statistic, the tasks selected for the cloud workload are the tasks that belong to users that requested less than 200 tasks. The total number of users that submitted jobs with 200 tasks or less is 42.

Figure 4.3 (a) shows the number of jobs for each of the 42 user and Figure 4.3 (b) shows the number of task for each of the 42 users. After the tasks were extracted to be used as workloads, the unique users are separated each with their own table that contains all the tasks requested by the user. Table 4.6 shows the field for each user table that was cross referenced and extracted from the Google cluster trace as explained previously. Furthermore, three fields are added in the task table, namely Task length, Task input size, Task output size.

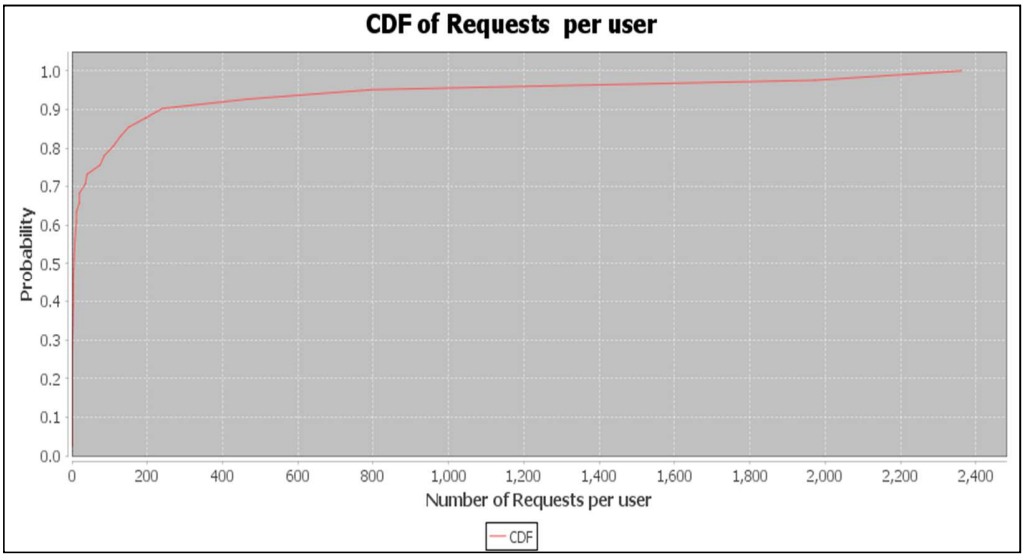

Figure 4.2. Cumulative Distribution Function for the number of tasks per each unique user.

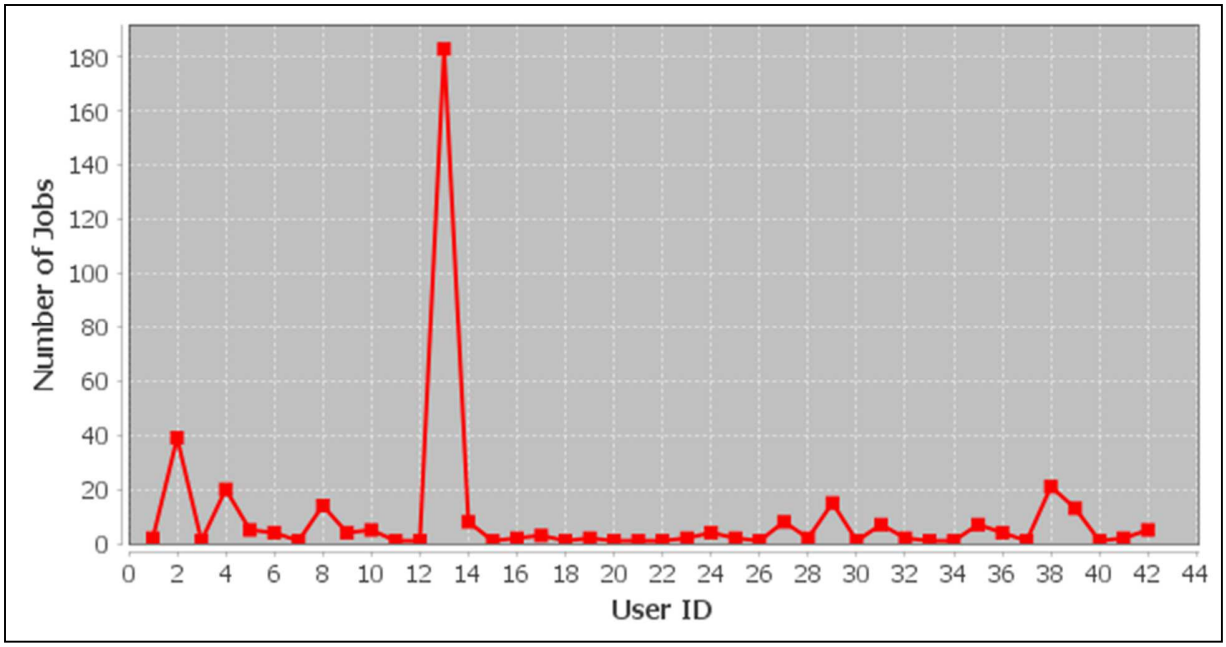

Figure 4.3 (a) Number Jobs per Each Unique User.

The new fields are generated randomly for each user as shown in Figure 4.8. The task length is the number of instructions in millions. It takes into account when generating the new data

that for computation offloading to be beneficial in terms of energy efficiency the workload needs to perform more than 1000 cycles of computation for each byte of data [12].

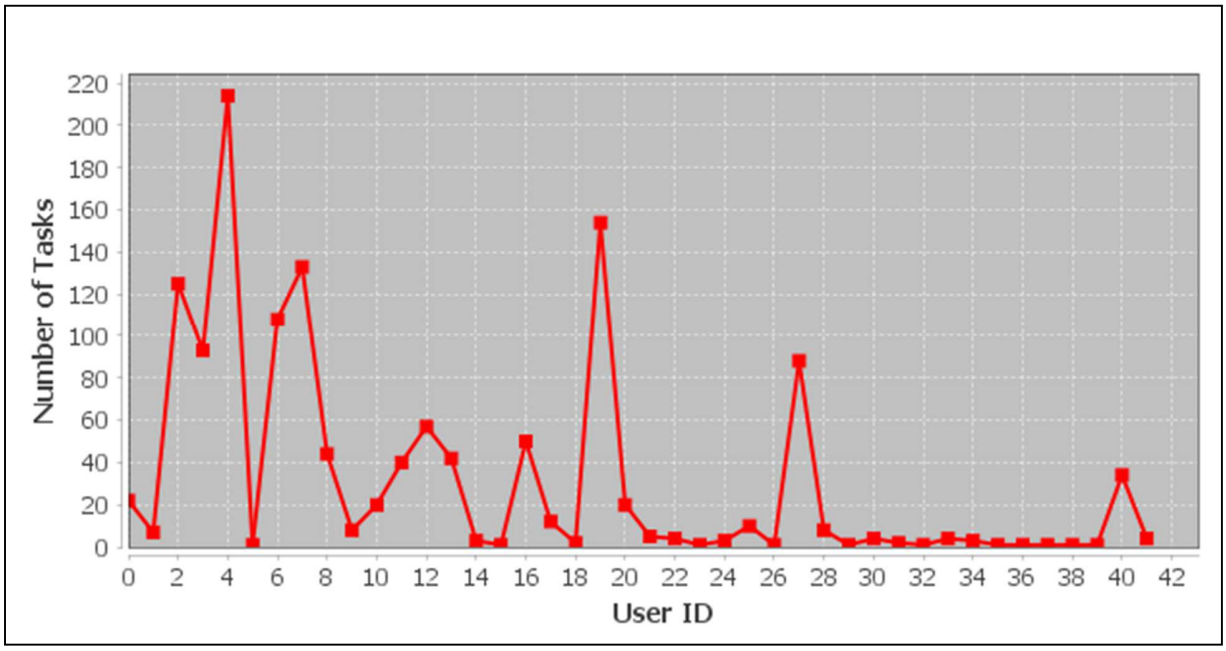

Figure 4.3 (b) Number Tasks per Each Unique User.

The number of cycles per instruction depends upon its type, for example, load (store) takes 6, arithmetic takes 4, and other types take 3 cycles. Hence, the length should be 160 – 330 instructions for each byte of data transmitted. A random function was used to generate the tasks input and output data sizes in the range 100-400 MB and the task lengths in the range 1000-3000 MIs.

4.1.3. Mobile Context Information Generation

The user tables that are discussed in the previous section are related to the users' requests generated from one location due to the fact that the tasks constraints specified the location of the machines to be used. The data rate of the network used by the mobile devices is set

according to the type of the connection: 3G or Wi-Fi. An 802.11g Wi-Fi device in close proximity to a network router will often connect at 54 Mbps. Wi-Fi network equipment use dynamic rate scaling and the data rate defined ratings are 54Mbps, 48Mbps, 36Mbps, 24Mbps, 18Mbps, 12Mbps, 9Mbps and 6Mbps. As for the 3G network the data rate ranges are: 24Mbps, 16Mbps, 8Mbps and 4Mbps. The energy level is considered in the simulation as a binary state; either the mobile has energy above the threshold (1), or below the threshold (0), where a threshold is defined as the minimum energy level required for the mobile device to carry out the job.

Table 4.6. User Table's Fields
1. Job ID
2. Task Index – within the job
3. Scheduling Class
4. Resource Request for CPU cores
5. Resource Request for RAM
6. Resource Request for Disk Space

Table 4.7. New User Table's Fields
1. Job ID
2. Task Index – within the job
3. Scheduling Class
4. Resource Request for CPU cores
5. Resource Request for RAM
6. Resource Request for Disk Space
7. Task Length in MI
8. Task Input Size in MB
9. Task Output Size in MB

4.2. Proposed Cloud System Simulation

The cloud system was simulated using CloudSim software framework [38]. CloudSim is a discrete event simulator used to model cloud system architecture and application services. It is a java library that was used in this work on NetBeans. The CloudSim multi-layered architecture is shown in Figure 4.4.

The CloudSim employs layered architecture. The top layer is the user code layer that defines the cloud scenario in terms of number of hosts, number of VMs and broker scheduling policies. In addition, the application configuration defines the number of tasks, their requirements, and

the number of users. The CloudSim layer is used to model and simulate cloud-based datacenters. The datacenter environment in CloudSim provides management interfaces for VMs, memory, storage, and bandwidth.

Figure 4.5 shows CloudSim class diagram. A list of some of the classes follows to discuss the usage and utilization of each class when modeling the cloud and the cloud based application services.

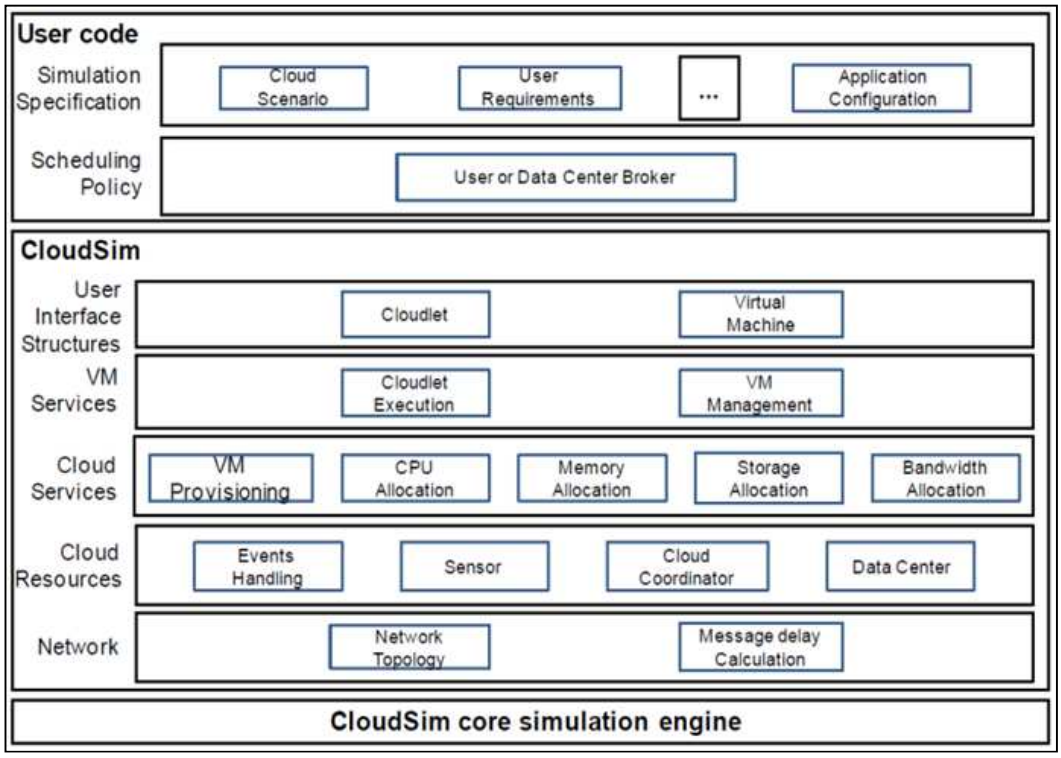

Figure 4.4. Layered CloudSim Architecture [39]**.**

- Cloudlet: It models the cloud-based application service that is executed in the datacenters. Specifically, cloudlets are used in the experiments of this thesis to model the tasks. Each cloudlet has a set of characteristics that are used to describe the task in terms of its unique ID, task length in Million Instructions (MI), CPU (number of cores), task file size, task output size, utilization model for the CPU, memory, and network bandwidth

(BW). The CloudletScheduler is also defined in the cloudlet and it refers to the type of scheduler used to schedule the cloudlets on PE (Processing Elements or cores of the CPU). Two types of schedulers are available for Cloudlet in CloudSim: TimeShared-CloudletScheduler and SpaceShared-CloudletScheduler.

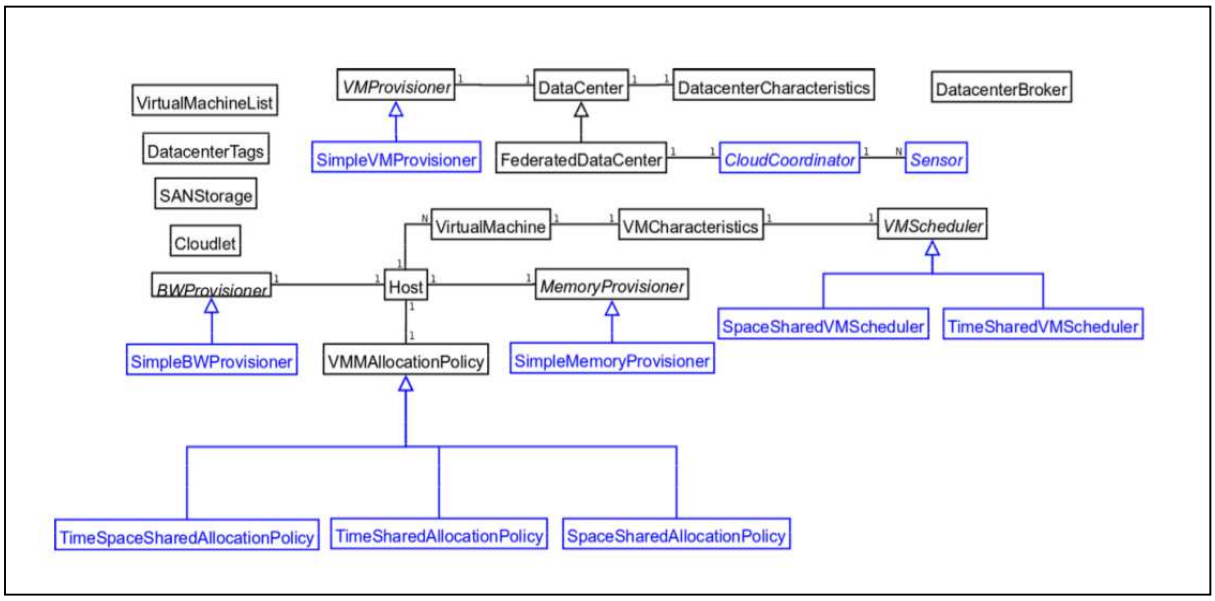

Figure 4.5. Shows Cloudsim Class Design Diagram [6].

- DataCenter and DataceterCharacteristics: A Datacenter contains many hosts. The first step in creating a datacenter is to create the hosts by calling the Host class and assigning the properties of each host in the call function. The Host class defines the host ID, the provisioning policy for both the host memory and the host bandwidth. It also defines the size of the storage for the host, number of PE units, and the type of the VM scheduler (more VM scheduler in the VMScheduler class). The second step is to define the Datacenter-Characteristics object that stores the following properties of a data center: architecture (e.g. x86), OS (e.g. Linux) , list of hosts created, the type of hyper visor used by the datacenter to virtualize the hosts (e.g. Xen), time zone where the datacenter is

located, and cost of using the datacenter. The last step is create the datacenter by calling the datacenter class and specifying the datacenter name, characteristics, the allocation policy of VMs on the hosts, and the storage list for the SAN devices.

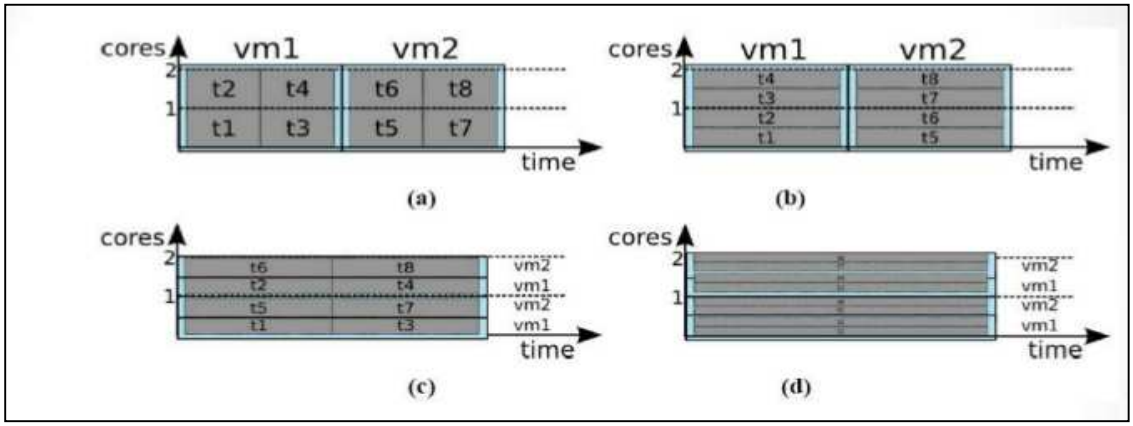

Figure 4.6. The Different Provisioning Policies for VMs and tasks [39]**. (a) Space-Shared Provisioning for VMs and tasks. (b) Space-Shared Provisioing for VMs and Time-Shared for tasks. (c) Time-Shared Provisioing for VMs and Space-Shared Provisioing for tasks. (d) Time-Shared Provisiong for both VMs and tasks.**

- DataCenterBroker: The datacenter broker is an intermediatry between users and service providers. The broker uses the Cloud Information Service (CIS) to find suitable resources for the users that meet their QoS requirements. This class is used in our simulation to implement the Simulated Annealing allocation algorithm and the deadline based scheduling algorithm. The interface between the user and the cloud communicates with this class and passes the user tasks that needs to be assigned suitable VMs.
- VirtualMachine (VM) and VmAllocationPolicy: VM class implements the Virtual Machine in CloudSim. This class defines a method for creating the VM. The specification for the VM defines the Million Instructions Per Seconds (MIPS) for each VM, image size in MB, memory or RAM in MB, bandwidth in Mbps, the number of PEs, and the type of

hypervisor. The values for VM specification are defined in our simulation according to the machine specification set by the cloud provider, Google Compute Cloud [40]

Scheduling policies for both VM and cloudlets can be further explained using the example in Figure 4.6 from the CloudSim documentation [39]. In this example, there is a machine with two CPU cores that is shared between two VMs. Also, there are eight tasks to be executed on this machine. In Figure 4.6.(a), SpaceSharedScheduling of VMs and tasks shows that for each time unit only one task unit executes on one VM that is given one core. Figure 4.6.(b) shows the SpaceSharedScheduling of VMs and TimeShared-Scheduling for tasks. In this case,only one VM is assigned to one core and two tasks for each VM in each core for every time unit. Figure 4.6. (c) shows the TimeShared-Scheduling of VMs and SpaceShared-Scheduling for tasks. In this case, two VMs are assigned to one core and each VM runs two task units for every time unit. Finally, in Figure 4.6 (d) TimeShared-Scheduling of both VMs and tasks shows that for each time unit, each core is assigned two VMs and each VM is assigned two tasks. For the tasks in the workflow used in our experiments to be dependent and execute sequentially, the last VMs and tasks allocation policy is used, that is TimeShared-Scheduler of VMs and TimeShared-Scheduler for tasks. Then, a delay is introduced to each task that needs to wait for its previous task to complete its execution. The delay is equal to the end time of the previous task as it is obtained from the scheduling algorithm. After reviewing the basic functions of CloudSim, we discuss the performance of the system evaluation in the next section.

4.3. Performance Evaluation

4.3.1 Experimental Setup

Four different types of VMs are used as shown in table 4.8. A datacenter is created with 10 hosts each has eight cores. There are 42 users each has a different number of jobs consisting of indexed tasks. The tasks are read from the csv file sequentially, which are then submitted for mapping and scheduling.

Table 4.8 Different types of VMs used in the experiments.

Type/Charc .	*MIPS*	*RAM(MB)*	*Number of CPU Cores*
1	*40000*	*512*	*1*
2	*50000*	*1024*	*2*
3	*60000*	*2048*	*4*
4	*70000*	*4096*	*8*

- Task- VM Mapping: The set of tasks are collected from all the users, then the set of VMs are created to run those tasks, and later both sets are submitted to the broker for executing them in the simulator. The broker calls the mapping and scheduling functions that we programmed using the algorithms in chapter 3. Only sequential tasks are considered in our simulation. The mapping and scheduling functions bind each task with a suitble VM and returns the map with a schedule for each task and VM. The VMS are submitted to the hosts and the tasks are executed following the start and end time specified in their schedule. The experiments were repeated 10 times and the results are averaged for every point in a graph.

4.3.2. Performance Parameters

- *Execution Time:* the estimated execution time of all the tasks submitted by the user calculated from the task-VM map. Each task is assigned to a VM using the simulated annealing algorithm and the execution time is calculated as follows:

Estimated Task Execution time = Task Length (MI)/ VM Capacity (MIPS)

Total estimated execution time for a user is the sum of the estimated execution time of all the tasks submitted by that user.

The actual total execution time of all the tasks of a user is obtained after all the tasks are executed. The execution time for each user's workflow is the maximum finish time of all the tasks belonging to that user:

Actual Execution Time of All Tasks = Max (finish times of all tasks)

- Cost: User cost of executing a workflow is calculated by first calculating the cost of using the VM multiplied by the time period the VM is used. Adding all the costs incurred by different types of VMs in executing the workflow for a user gives the total cost for each user. Another cost is the provider cost. Each data center has an associated cost in CloudSim.
- Average CPU utilization: CPU utilization of the cloud system physical machines was monitored for the period of the workload execution (execution of 42 user jobs). The total execution time for all the user jobs is divided into 10 seconds intervals and the average CPU utilization of the datacenter's machines is calculated for each time interval.
- Deadline Based Simulated Annealing algorithm (SA-DB) is used as the base algorithm for the resource allocation and scheduling sub-system. The Context-Aware Deadline Based Simulated Annealing algorithm (CASA-DB), is the algorithm that implements the context aware resource allocation and scheduling. These two algorithms are used to compare the cloud system performance for the context free and context aware scenarios.

4.4. Simulation Results and Analysis

4.4.1. Calculating Workflow Execution Deadline

An experiment prior to introducing the deadline of the workflow to the execution shows that all of the users' workflows actual execution times exceeded the expected execution times. Therefore, a deadline for the tasks constraining the system to choose the VMs that meet the workflow execution deadline was calculated. The deadline is then increased gradually while monitoring the violation percentage.

The execution deadline is calculated in two steps. First, the minimum expected execution time, *EET*, is calculated for each user's workflow by assigning the fastest VM to all the tasks of that user. Secondly, *EET* is multiplied by 2^k, where k =1,2,3, generating three different deadlines for each user's workflow as shown in table 2.8.

Figure 4.7. shows different deadlines in comaprison with the actual execution times of each user's workflow. Table 4.9 shows the total number of workflows for all 42 users violating each of the deadlines and their percentages. 8. It is evident that relaxing the deadline results in fewer violations of the deadline and a strict deadline results in high number of violations. Hence, for remaining simulations the deadline is set to 8* Min ET, which shows the least number of violations.

Table 4.9. Different Deadlines and the number and percentage of Violations.

Deadline	# Violations (total # users=42)	% Violations
2 * Min EET	35	83.33
4* Min EET	15	35.71
8 * Min EET	5	11.90

4.4.2. Effect of Varying Task Lengths on the Execution Time of Workflows

The execution time of a task running on a VM is the task length in MI divided by the VM capacity in MIPS. It increases by increasing the task length. Figure 4.8 shows the execution

times for workflows of all the users by varying the task length from 10MI to 10,000MI. The workflows with the task length equal to 10 MI complete executions within 4/10th of a second, which is very small. Hence, as mentioned before, it is not energy efficient to offload small tasks to the cloud as the communication cost far exceeds the gain in the execution time. The high execution times are observed for the workflows with task lengths 1000MI and 10,000MI, which take maximum values of 0.5 minute and 1.3 minutes respectively. For the remaining simulations we select task lengths in the range of 1000-3000 MI, and they are generated randomly from this range.

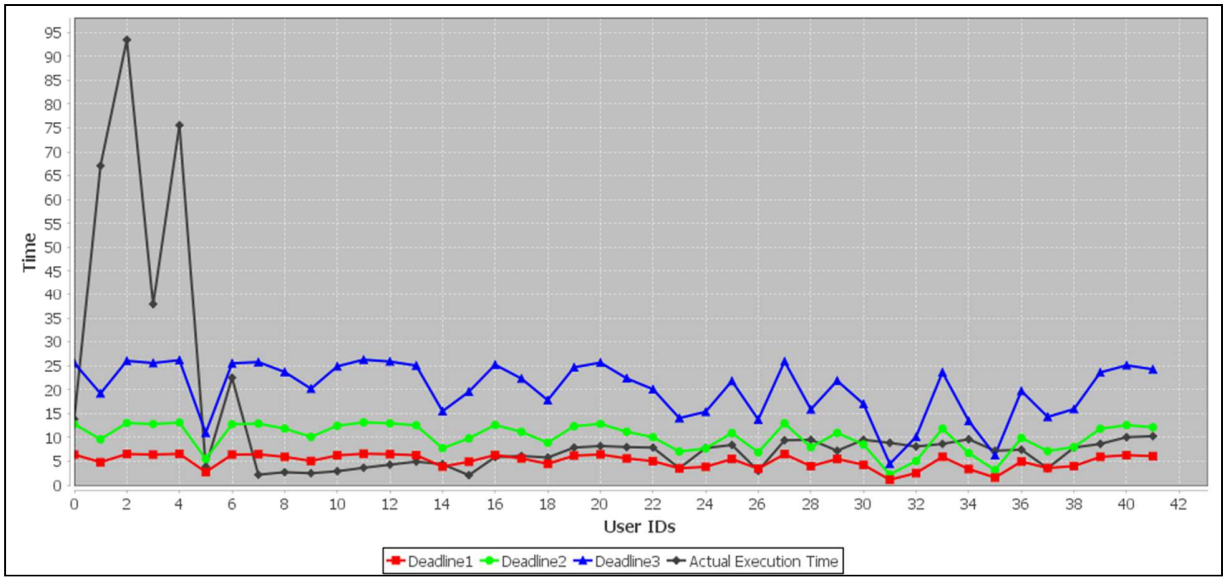

Figure 4.7. Different Workflow Deadline vs. Actual Execution Times of Workflows.

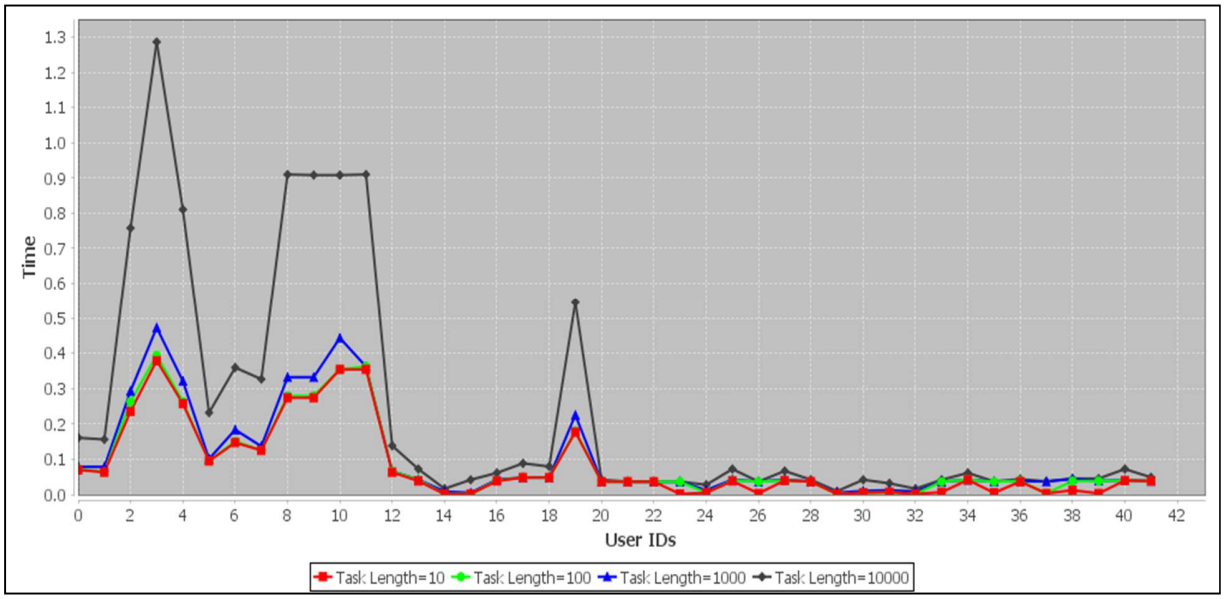

Figure 4.8. Execution Times of Workflows (in seconds) by Varying the Task Lengths for all 42 Users.

4.4.3 Context Aware Provisioning

Mobile Energy: Both context aware provisioning and scheduling take into account the energy level of mobile devices. A mobile user sends a job (request) to the cloud, which also receives the mobile context from the CSB that in turn periodically reads from the mobile device. The CSB defines a threshold for energy level that is the minimum energy level required by a mobile device to send data and receive results with the tasks in the cloud. It launches the workflow if the energy level is above the threshold for the user to be able to receive the results of the workflow. If the energy level is below the threshold the user workflow is delayed and the user context is monitored until the mobile device replenishes its energy and is able to receive the result. The workflow is then scheduled for execution. If a mobile device doesn't replenish within 10 minutes, then the workflow is discarded after 10 minutes. The energy replenish time is simulated as a delay in the workflow execution, which is a random value between 60 sec – 300 seconds. Figure 4.9 shows the execution times of the context free cloud system (SA-DB) and the

context aware cloud system (CASA-DB). In this simulation, workflows for the users with user ID 18 to user ID 42 are delayed due to their low energy level. The benefit to the cloud provider lies in the fact that the context aware scheduling was able to delay execution of the workflows until the users' mobiles are able to get the results back. Alternatively, in the context free scheduling the workflow is scheduled and executed, but the results are discarded as the connection with the mobile device fails due to low energy level of the mobile device. The context aware cloud system provides a cost effective mechanism to avoid wasting resources and it gives the users' mobile devices the opportunity to resume their workflow execution after regaining their energy level.

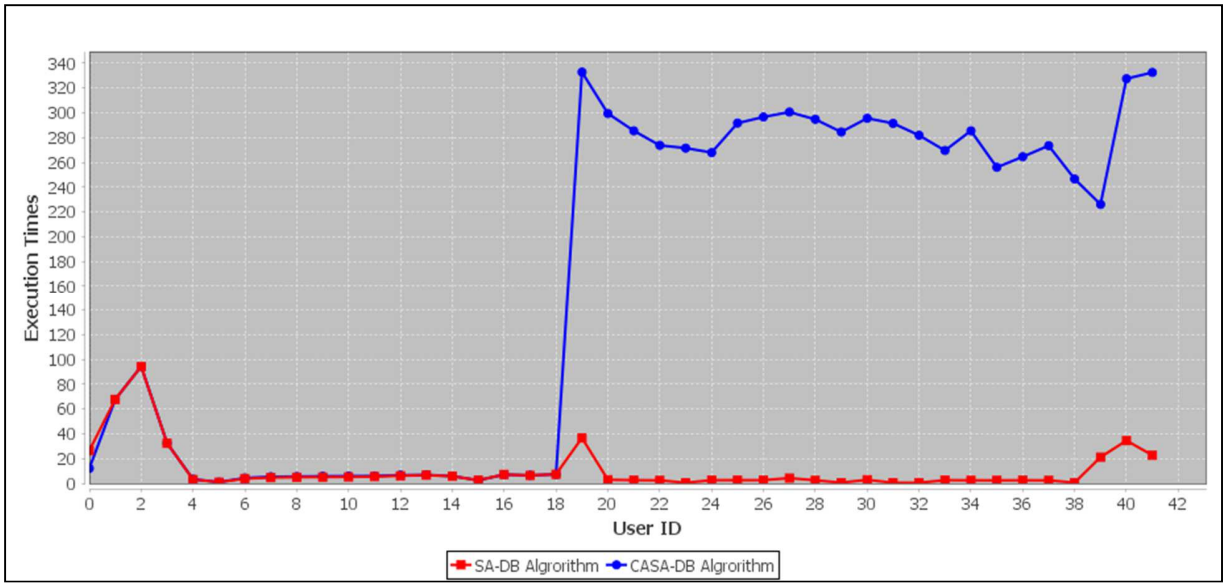

Figure 4.9 Execution Times (In Seconds) Of The Context Free Cloud System (SA-DB) And The Context Aware Cloud (CASA-DB) System Considering Only Energy Level Context.

Mobile Data Rate and Slack Time: Mobile devices experience different data rates depending on the connection type and communication quality of the wireless network. The data rate also varies for the same connection type with the location of the device. The mobile device context includes, as mentioned before, the data rate in units of Mbps. Transmission times *TT* between

the mobile and the cloud for both directions are calculated using the data rate and the data size of the task to be transferred. From the cloud's perspective, the transmission time from the cloud to the mobile TT_{cm} is calculated using the mobile download link data rate and the output data size of the last task in the workflow. Alternatively, the transmission time from the mobile to the cloud TT_{mc}, is calculated using the upload link data rate of the mobile device and the input data size of the first task of the workflow that is delegated for execution in the cloud. As shown in Figure 4.10, two scenarios are plotted for the transmission time in minutes. The first scenario is the dominant LTE, where 70% of mobile devices are set to use LTE connection while 30% of the mobile devices are set for Wi-Fi connection. The second scenario is the Wi-Fi dominant scenario, which is set such that 70% of mobile devices are configured for Wi-Fi connection while 30% of the mobile devices are configured for LTE connection.

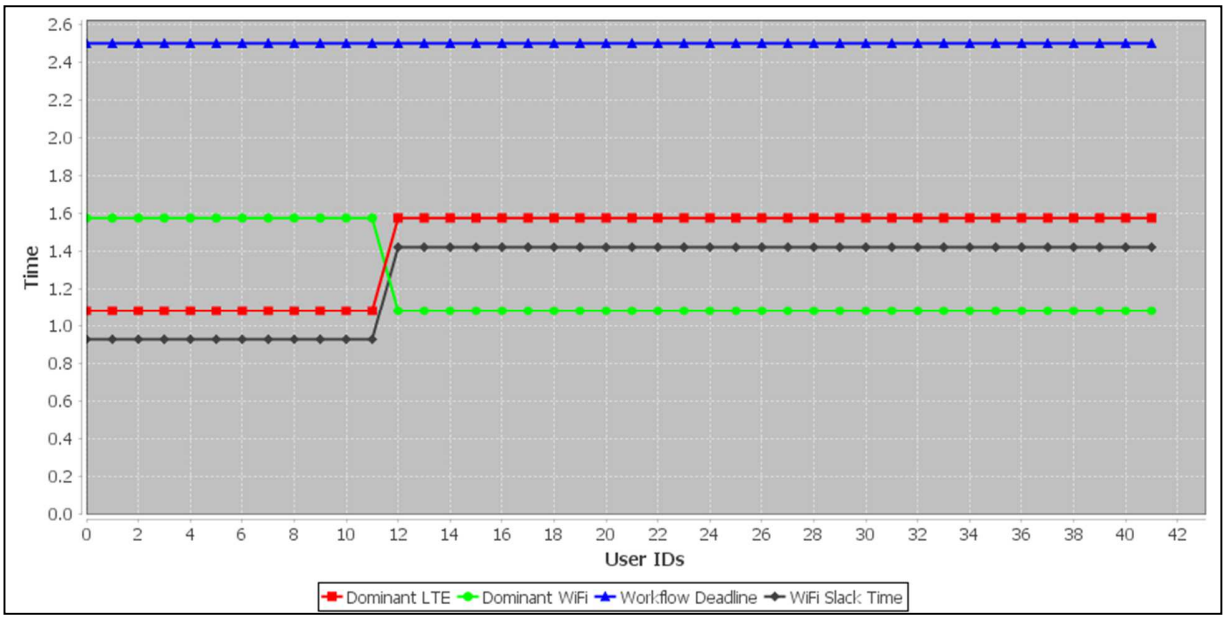

Figure 4.10. Transmission Times Using Different Types of Connection Shown In Comparison With the Workflow Deadline and the Slack Time Gained In the Wi-Fi Dominant Scenario.

In addition to transmission times, we also calculated deadline W_d for each workflow, which we assumed comes with the user's request to the cloud. It is calculated using the slowest data rate that the user might experience, i.e. the lowest rate for LTE connection, which is 4Mbps. Furthermore, the slack time is calculated using the workflow deadline W_d and the dominant Wi-Fi connection scenario. The cloud system exploits the slack time gained by the mobile device using the high speed Wi-Fi connection, as opposed vs. the LTE connection that the deadline W_d was based upon.

- We calculated the slack time of the Wi-Fi dominant scenario. The increased slack time is a time gain that the broker uses to its advantage. The broker delays the execution of the workflows of the mobile users that are experiencing higher speed connections by a factor of slack time. The delay in these workflows will not affect the SLA agreement as the deadline is met regardless due to the decreased transfer time for the mobile device. Figure 4.11 and Figure 4.12 show the execution times of workflows of each user when the delay is set to be equal to the slack time and delay is equal to 10% of slack time. Two cloud systems are compared in both cases. The context free cloud system using the SA-DB and the context aware cloud system using CASA-DB algorithm. The time gained by using the extra slack time and delaying the workflow is used by the broker, to the cloud provider's advantage, by mapping slower VMs for the same workflow in the context aware scenario. Slower VMs achieve higher execution times and cost less to operate. The cost of using the datacenter is calculated as the cost of machine multiplied by the time period it is used. An example of provider cost is the energy cost of using the machine. Machines operating with less PEs and slower processors consume less energy [33].

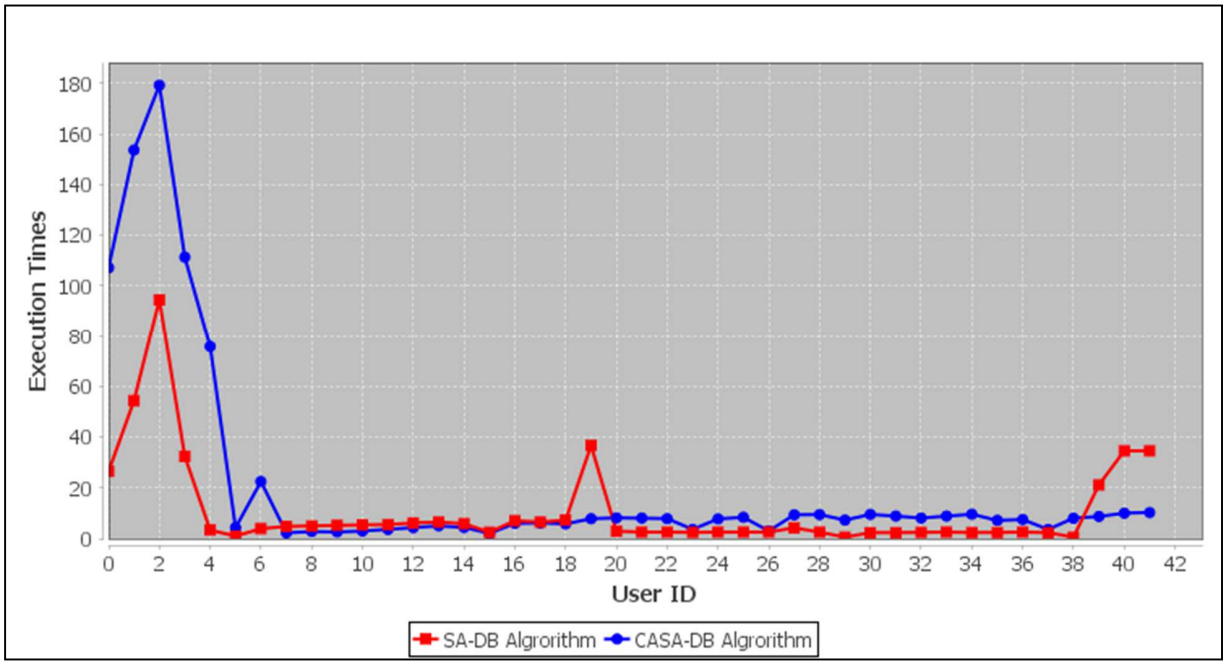

Figure 4.11. Execution Times Of Users' Workflow When They Are Delayed By Slack Time (CASA-DB) Vs. Execution Times When They Are Not Delayed (SA-DB).

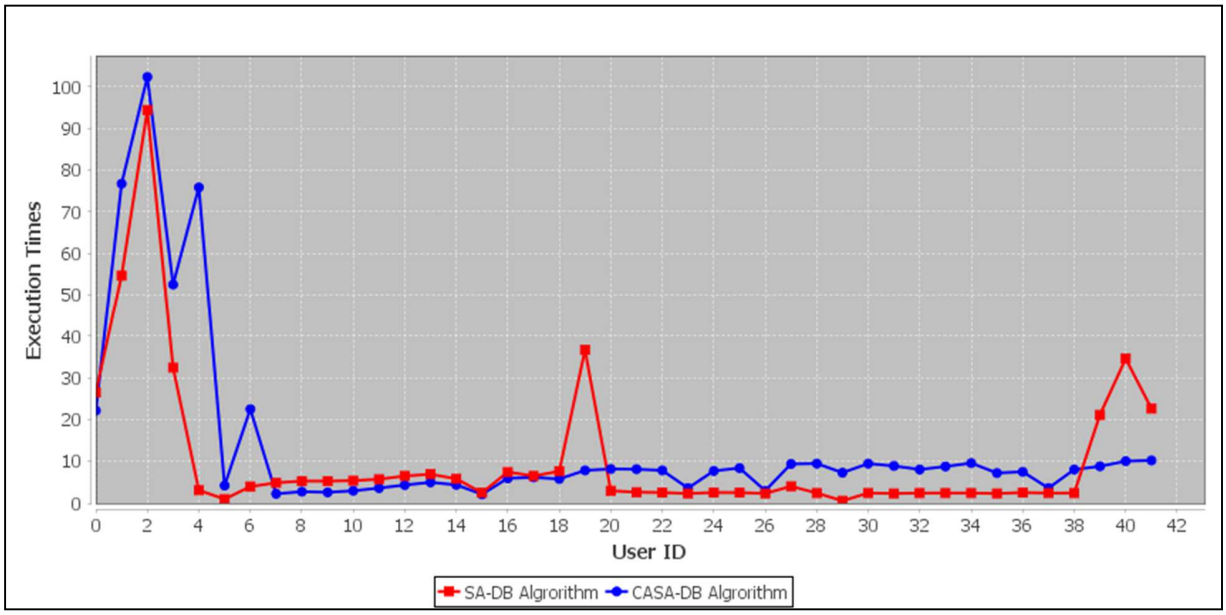

Figure 4.12. Execution Times Of Users' Workflow When They Are Delayed By 10% Of Slack Time (CASA-DB) Vs. Execution Times When They Are Not Delayed (SA-DB).

Average CPU Utilization: The average CPU utilization of datacenter is calculated to show the advantage of using the context aware system. The datacenter consists of 10 hosts. For each

host, the CPU utilization was calculated by first logging the execution times known as the ActualCPUTime () in CloudSim for each cloudlet that was executed in that particular host. The execution start time and execution end time of each cloudlet in each host was also logged. The logged period is divided into 10 seconds intervals. For each interval, the CPU time is the summation of CPU times of all tasks executing in that interval (The CPU time of each interval is calculated by adding up the ActualCPUTime of each task starting at an interval, to the interval's CPU time). If the ActualCPUTime is greater than the interval, then the CPU Time will be 10 seconds for that interval and the residual of the ActualCPUTime is calculated and distributed among the following interval CPU times. The residual of the execution time equals *ActualCPUTime -10*, will be added to the following interval's CPU Time. If the residual is greater than 10, then residual -10 will remain and it will be added to the following interval and so on until the residual of the *ActualCPUTime* is distributed among the intervals. For example, if the ActualCPUTime of a task is equal to 22 seconds and the task starts executing in interval 1, then only 10 seconds will be added to CPU time of interval 1. The residual 22-10 = 12 seconds, is greater than 10 thus only 10 seconds will be added to interval 2's CPU Time. The new residual 12 -10 = 2 seconds will be added to interval 3's CPU time.

Each interval of 10 seconds has its corresponding CPU time collected from the tasks that were executed on the ten hosts of the data center during that interval. The CPU time calculated for each interval is accumulated for the ten hosts. Hence, the CPU utilization calculated from *the accumulated* CPU time represents the average value of the CPU utilization of the data center in each interval. The average CPU utilization is calculated using the maximum CPU time of all the intervals as follows:

$$Average\ CPU\ Utilization\ of\ an\ Interval = \frac{CPU\ Time\ of\ Interval}{Maximum\ CPU\ Time\ of\ all\ Intervals}$$

Figure 4.13 shows the CPU utilization for the context-aware system CASA-DB and the context free system SA-DB for intervals from 1-14, for a total of 140 seconds. It has to be noted that the actual CPU time does not take into account the wait each tasks incurs due to scheduling and delay. CASA-DB in this case considers the energy context only, as mentioned before, the delay is introduced to a number of users. From intervals 11 onward, only the context aware system will remain executing the tasks, thus the CPU utilization for that period is the for the context aware system only. The figure shows that for the first 11 time intervals, the CPU utilization for context aware system (CASA-DB) is better in 70% of the intervals than the CPU utilization of the context free system (SA-DB).

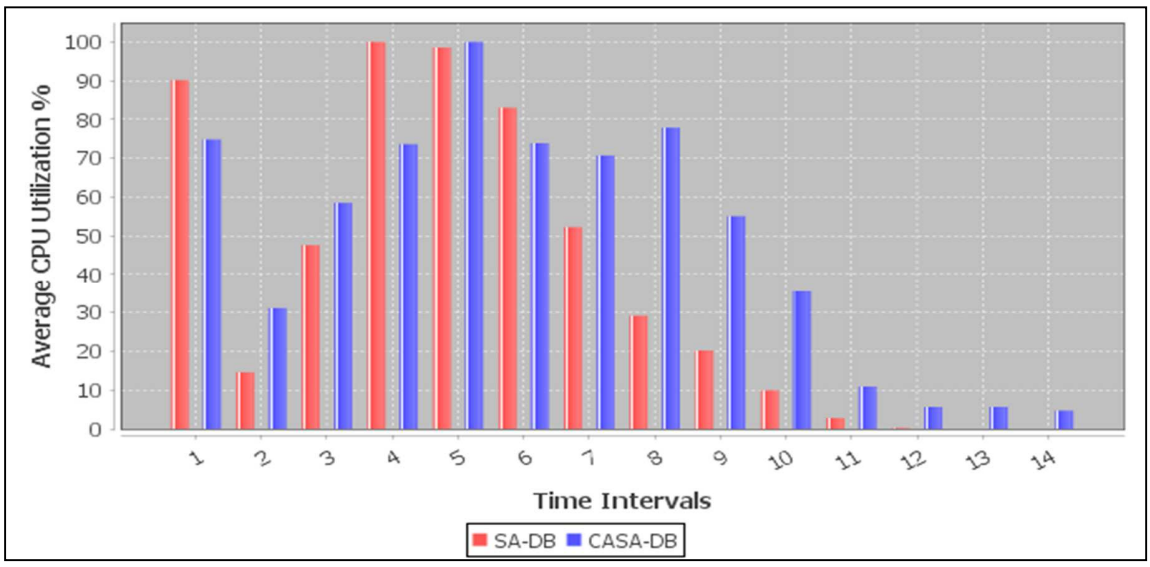

Figure 4.13 Average CPU Utilization Of SA-DB vs. Average CPU Utilization of CASA-DB System Considering Only Energy Level Context.

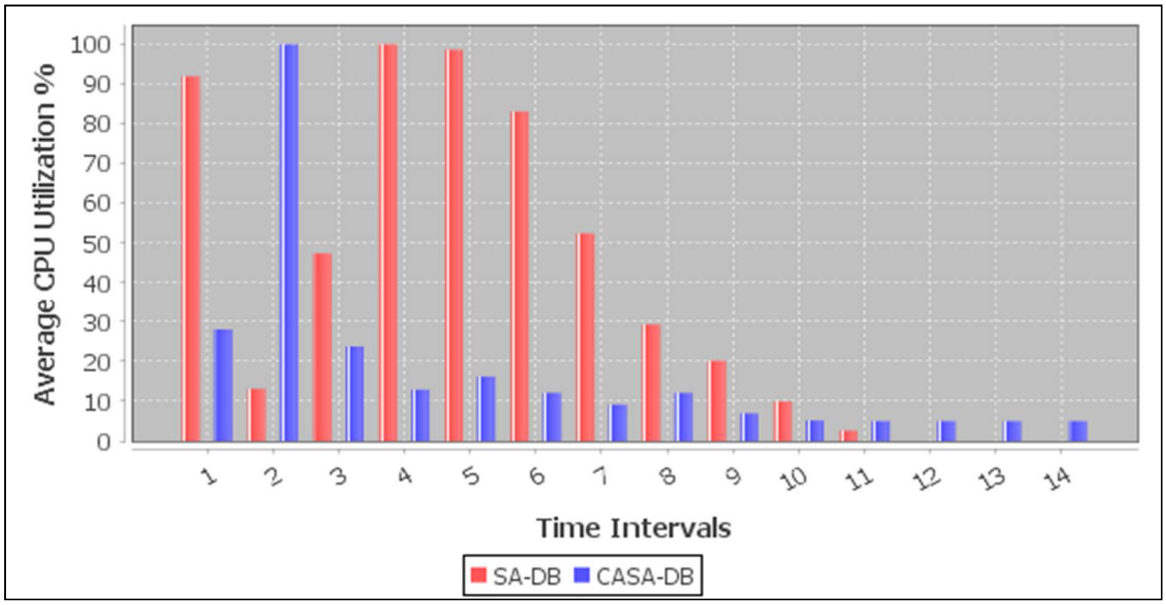

Figure 4.14 Average CPU Utilization of SA-DB vs. Average CPU Utilization of CASA-DB System considering only Communication Context (Slack Time).

Figure 4.14 shows the average CPU utilization for the context aware system (CASA-DB) considering the communication context vs. the CPU utilization of the context free system (SA-DB). The figure show that the context free system SA-DB, performs better in terms of CPU utilization than the context aware system CASA-DB.

Chapter 5

Conclusion and Future Work

In this thesis we propose a context aware cloud system that monitors and uses the mobile context information to make intelligent cloud resource allocation and scheduling decisions. The cloud resource allocation and scheduling scheme's objective is to minimize cost of execution whilst meeting the mobile user jobs' deadline. The rational of using a context aware cloud system is validated through simulation, which shows that the context aware system could reduce the cost of executing the mobile user jobs on the cloud.

Mobile cloud users present their own challenges when acquiring services from the cloud. The inherent characteristics of the mobile user devices and their mobility induce these new set of challenges. The context aware system considers the following user context information: location, mobile device energy level and the data rate the mobile device is experiencing. Mobile user jobs are allocated to data centers that are located in the same location zone where the mobile device resides to reduce any additional latency that the user may encounter using distant datacenters. Furthermore, in our scheme the context aware cloud broker monitors the mobile device energy level that may drop below the threshold required for communication resulting in data loss. The broker make a decision of either executing the mobile user job on the cloud if the energy level is above the threshold, *or* hold the execution of the job if the energy level is below the threshold. Once the mobile device regains the level of energy required for successful communication, the cloud broker will execute the mobile user job. In contrast, in the context free scenario, if the mobile device energy is low beyond the required energy for

receiving the cloud result and the mobile user launches their job on the cloud then the job will be executed regardless of the user context and the user will not be able to receive the results from the cloud. In this case the cost for the user is the cost of transmitting the job and its data to the cloud in addition to the cost of executing the job on the cloud. The user will incur this cost even when the user is not able to receive the results. Alternatively, in the energy context aware scenario, the mobile user avoids incurring this cost because the context aware broker will make the intelligent decision of delaying the execution of the job until the mobile device regains the energy required to receive the result from the cloud. In addition to saving the user the extra cost, our implementation of the cloud system shows better CPU utilization of resources in the context aware scenario than in the context free scenario. Simulation shows that the CPU utilization of the energy context aware system performs better in 70% of the time intervals of the monitored period than the context free system.

In addition to the energy context, the mobile device's data rate was considered in the context aware system. The mobile device network connection type and/or data rate changes affect directly the communication between the cloud users and the cloud provider. The change in communication time is exploited to adapt the resource allocation and scheduling in the cloud. A *slack time* is calculated to reflect the change in the data rate the mobile device is experiencing. Slack time is the difference between the user deadline (W_d) and the turnaround time for a user job which encapsulates the transmission time of the user jobs. The slack time has to be more than zero for the cloud system to meet the deadline (W_d). If the data rate the mobile device is experiencing becomes faster thus reducing the transmission time, then the slack time will be increased which gives the cloud system additional time to delay executing the

job. The delay of job execution could be used to increase the system internal execution deadline when allocating the resources for the jobs of the user who is experiencing higher data rates. Thus the cloud system could allocate slower VMs to the user jobs. In the future, we plan to design a model for the calculation of the delay in job execution introduced by the context change.

The use of context aware system provides better cloud system utilization and potentially better user experience. However, a design of a 'context aware' SLA agreement between the cloud user and the cloud provider is needed to take advantage of the user context. The SLA negotiation has to include the terms of which the mobile user and the cloud provider will agree upon when using the user context information. These terms should include an agreement on changes in costs and/or user defined deadlines that might take effect in the context aware system. This topic will also be explored in our future work.

Furthermore, we plan to study the effect of using a context engine and monitoring the user context information on the context aware *cloud system scalability*. Cloud mobile users' numbers are growing rapidly and a context-aware cloud system has to provide the extra processing and storage incurred from managing the context information of cloud mobile users.

References

[1] The_Centre_for_Energy-Efficient_Telecommunications, "The Power of Wireless Cloud," *Whitepaper*, 2013.

[2] K. Bratanis, D. Kourtesis, I. Paraskakis, and S. Braun, "A Research Roadmap for Bringing Continuous Quality Assurance and Optimization to Cloud Service Brokers," *Proc. eChallenges*, 2013.

[3] J. Wilkes and C. Reiss, "Details of the ClusterData-2011-1 trace," 2011. [Online]. Available: https://code.google.com/p/.

[4] B. Snaith, M. Hardy, and a. Walker, "Emergency ultrasound in the prehospital setting: the impact of environment on examination outcomes," *Emergency Medicine Journal*, vol. 28. pp. 1063–1065, 2011.

[5] D. C. Marinescu, *Cloud Computing: Theory and Practice*. 2013.

[6] R. Buyya, J. Broberg, and A. Goscinski, *Cloud Computing: Principles and Paradigms*. 2011.

[7] A. U. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Commun. Surv. Tutorials*, vol. 16, pp. 393–413, 2014.

[8] M. Proebster, M. Kaschub, T. Werthmann, and S. Valentin, "Context-Aware Resource Allocation for Cellular Wireless Networks," *EURASIP J. Wirel. Commun. Netw.*, vol. 2012, no. 1, p. 216, 2012.

[9] M. R. Rahimi, N. Venkatasubramanian, S. Mehrotra, and A. V. Vasilakos, "MAPCloud: Mobile applications on an elastic and scalable 2-tier cloud architecture," in *Proceedings - 2012 IEEE/ACM 5th International Conference on Utility and Cloud Computing, UCC 2012*, 2012, pp. 83–90.

[10] M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, "MuSIC: Mobility-aware optimal service allocation in mobile cloud computing," in *IEEE International Conference on Cloud Computing, CLOUD*, 2013, pp. 75–82.

[11] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic service placement in geographically distributed clouds," *IEEE J. Sel. Areas Commun.*, vol. 31, pp. 762–772, 2013.

[12] A. P. Miettinen, "Energy efficiency of mobile clients in cloud computing," *Energy*, p. 4, 2010.

[13] S. Di, D. Kondo, and W. Cirne, "Host load prediction in a Google compute cloud with a Bayesian model," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, 2012.

[14] J. Heo, K. Terada, M. Toyama, S. Kurumatani, and E. Y. Chen, "User demand prediction from application usage pattern in virtual smartphone," in *Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010*, 2010, pp. 449–455.

[15] P. Saripalli, G. V. R. Kiran, R. R. Shankar, H. Narware, and N. Bindal, "Load prediction and hot spot detection models for autonomic cloud computing," in *Proceedings - 2011 4th IEEE International Conference on Utility and Cloud Computing, UCC 2011*, 2011, pp. 397–402.

[16] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and T. Yimwadsana, "Predictability of web-server traffic congestion," in *Proceedings - WCW 2005: 10th International Workshop on Web Content Caching and Distribution*, 2005, pp. 97–103.

[17] M. Andreolini and S. Casolari, "Load prediction models in web-based systems," in *1st International Conference on Performance Evaluation Methodologies and Tools*, 2006.

[18] D. Yuan, Y. Yang, X. Liu, and J. Chen, “A data placement strategy in scientific cloud workflows,” *Futur. Gener. Comput. Syst.*, vol. 26, pp. 1200–1214, 2010.

[19] M. Ferber, T. Rauber, M. H. C. Torres, and T. Holvoet, “Resource allocation for cloud-assisted mobile applications,” in *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, 2012, pp. 400–407.

[20] M. Rodriguez Sossa and R. Buyya, “Deadline based Resource Provisioning and Scheduling Algorithmfor Scientific Workflows on Clouds,” *IEEE Trans. Cloud Comput.*, p. 1, 2014.

[21] C. Mei, D. Taylor, C. Wang, A. Chandra, and J. Weissman, “Sharing-aware cloud-based mobile outsourcing,” in *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, 2012, pp. 408–415.

[22] W. Zhang, Y. Wen, and D. O. Wu, “Energy-efficient scheduling policy for collaborative execution in mobile cloud computing,” *Proc. - IEEE INFOCOM*, pp. 190–194, 2013.

[23] J. Guitart, D. Carrera, V. Beltran, J. Torres, and E. Ayguadé, “Dynamic CPU provisioning for self-managed secure web applications in SMP hosting platforms,” *Comput. Networks*, vol. 52, pp. 1390–1409, 2008.

[24] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, “A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments,” in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 400–407.

[25] L. Guo, S. Zhao, S. Shen, and C. Jiang, “Task scheduling optimization in cloud computing based on heuristic Algorithm,” *J. Networks*, vol. 7, no. 3, pp. 547–553, 2012.

[26] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, “Kingfisher: Cost-aware elasticity in the cloud,” in *Proceedings - IEEE INFOCOM*, 2011, pp. 206–210.

[27] M. Jia, J. Cao, and L. Yang, “Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing,” in *Proceedings - IEEE INFOCOM*, 2014, pp. 352–357.

[28] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, “Calling the cloud: Enabling mobile phones as interfaces to cloud applications,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5896 LNCS, pp. 83–102.

[29] E. Cuervo and A. Balasubramanian, “MAUI: making smartphones last longer with code offload,” *Proc. 8th …*, vol. 17, pp. 49–62, 2010.

[30] M. Shiraz, A. Gani, R. H. Khokhar, and E. Ahmed, “An extendable simulation framework for modeling application processing potentials of smart mobile devices for mobile cloud computing,” in *Proceedings - 10th International Conference on Frontiers of Information Technology, FIT 2012*, 2012, pp. 331–336.

[31] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, “Energy-optimal mobile cloud computing under stochastic wireless channel,” *IEEE Trans. Wirel. Commun.*, vol. 12, pp. 4569–4581, 2013.

[32] M. Zafer and E. Modiano, “Minimum Energy Transmission Over a Wireless Channel With Deadline and Power Constraints,” *Autom. Control. IEEE Trans.*, vol. 54, pp. 2841–2852, 2009.

[33] P. Balakrishnan and C. K. Tham, “Energy-efficient mapping and scheduling of task interaction graphs for code offloading in mobile cloud computing,” in *Proceedings - 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013*, 2013, pp. 34–41.

[34] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing.,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[35] M. Wang and W. Zeng, “A comparison of four popular heuristics for task scheduling problem in computational grid,” in *2010 6th International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2010*, 2010.

[36] C. Reiss, J. Wilkes, and J. Hellerstein, “Google cluster-usage traces: format+ schema,” *Google Inc., …*, pp. 1–14, 2011.

[37] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Heterogeneity and dynamicity of clouds at scale,” in *Proceedings of the Third ACM Symposium on Cloud Computing - SoCC ’12*, 2012, pp. 1–13.

[38] R. Buyya, R. Ranjan, and R. N. Calheiros, “Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities,” in *Proceedings of the 2009 International Conference on High Performance Computing and Simulation, HPCS 2009*, 2009, pp. 1–11.

[39] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Softw. - Pract. Exp.*, vol. 41, no. 1, pp. 23–50, 2011.

[40] “Google Cloud Platform.” [Online]. Available: https://cloud.google.com/compute/docs/machine-types.