A TASK RESOURCE PROBABILISTIC SCHEDULING ALGORITHM

By

SRAVYA NUTALAPATI

Bachelor of Technology in Information Technology

Acharya Nagarjuna University

Guntur, AP, India

2009

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2011

A TASK RESOURCE PROBABILISTIC

SCHEDULING ALGORITHM

Thesis Approved:

Dr. Johnson P Thomas

Thesis Adviser

Dr. Subhash Kak

Dr. Michel Toulouse

Dr. Sheryl A. Tucker

Dean of the Graduate College

.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

A sensor detects some physical phenomenon. A Wireless sensor network (WSN) is a network of spatially distributed sensors to monitor physical events.

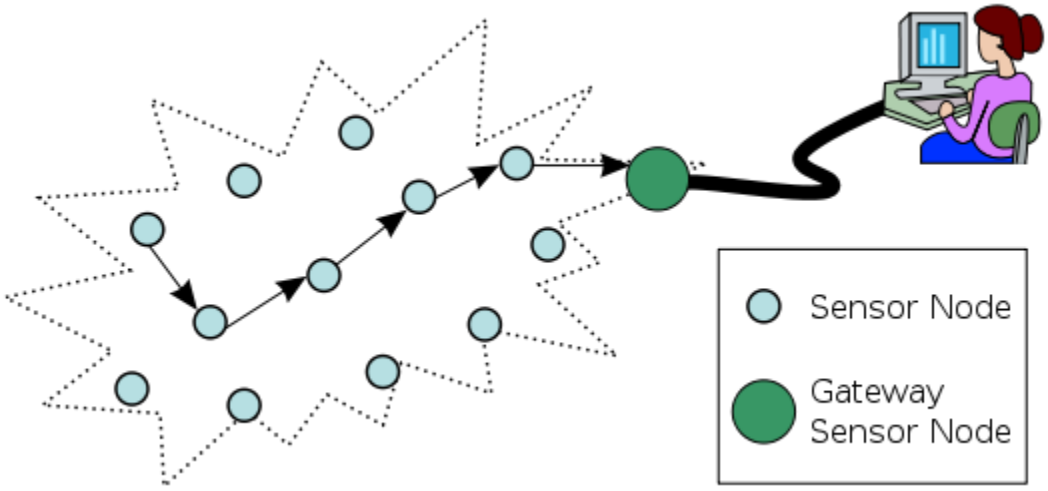

Figure 1: Wireless sensor network

There are many applications of wireless sensor network. Some of these include air pollution monitoring, forest fires detection, greenhouse monitoring and health monitoring.

.

1.1 Biomedical Sensors

In this thesis we concentrate on health monitoring using biomedical sensors. The condition of a patient is derived by measuring parameters such as temperature, blood pressure, heart beat etc using biomedical sensors attached to the patient's body. This information is then analyzed to determine for monitoring, diagnostic, treatment and other reasons. In this thesis we assume a patient who may not be at the hospital, but may be at home or carrying on with his normal activities. In order to monitor a patient remotely a set of biomedical sensors are attached to his body. The data from the sensors may have to be analyzed and processed in real-time such as in an emergency situation, whereas in other cases, time is of less importance. The signals from the sensors are sent to a cell phone. Once the cell phone receives the signal from a sensor it forwards the signal to a filter, which is located between a set of patients and cloud. A filter will take the input data such as temperature, blood pressure, heart beat etc and analyze them and process them if a real-time response is required. The response is sent to a hospital management system, for example, in the case of a patient's emergency condition. The parameters which do not require a real-time response are sent over to the cloud. The Cloud provides software to measure the parameters and provides secure storage for large amount of patient records as well as processing power.

Many tasks from different patients may arrive at the same time to a filter. Hence we need a scheduling algorithm to schedule the tasks based on the resources (memory, CPU etc) available. The scheduling algorithm will ensure that all time constrained tasks get scheduled and later executed within the required deadlines. If we are not able to schedule a task on one filter we need to schedule the task on other filters.

1.2 Cloud computing

Computing based on the internet using shared resources is called cloud computing. Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort. The underlying concept of cloud computing is the separation of applications from the operating systems and the hardware on which they run.

Cloud computing deliver applications via the internet, which are accessible from web browsers and desktop and mobile apps, while the software and data are stored on servers at a remote location.

In the past, many of us worried about losing our documents, photos and files if something bad happened to our computers, like a virus or a hardware malfunction. Today, our data is migrating beyond the boundaries of our personal computers and all our data would still safely reside on the web, accessible from any Internet-connected computer, anywhere in the world because of cloud computing.

1.3 Proposed Approach

In this thesis, we assume biomedical sensors are attached to the patient's body. The signals from the sensors are sent to a cell phone. Once the cell phone receives a signal from the sensors it forwards them to a filter, which is located between set of patients in a geographical area and the cloud. A filter is installed for the geographical group of patients in a network. The filter will measure, analyze and process health related parameters such as temperature, blood pressure, heart beat, etc. The sensor data may indicate that the patient needs a timely response. In such cases the data is sent to the hospital management system for appropriate action and a copy of the data will be stored in the cloud. The parameters which do not require a real-time response are sent directly to the cloud only for inclusion in the patient's medical record. If a medical professional wants to access the record of a particular patient he can access it from the cloud.

In this thesis we propose a scheduling mechanism that runs on the filters to meet the real-time requirements of patient sensor data. The scheduling algorithm schedules the sensor data based on the resources available in the cloud. The scheduling algorithm ensures that the time constrained parameters, get scheduled and executed before the deadline. If a parameter cannot be scheduled on one filter we need to look for other filters. The parameters which do not require a real-time response are scheduled on any one of the filters based on a probabilistic function.

Chapter II gives the literature review. The system architecture is described in Chapter III. Chapter IV gives more details about design and implementation of the algorithm. The simulation work and results are explained in Chapter V.

CHAPTER II

LITERATURE REVIEW

2.1 Cloud computing services

Delivering the computing resources such as operating system, hardware, software as a service rather than a product is called a cloud computing service. Cloud computing contains many different services which are mentioned below.

2.1.1 Infrastructure-as-a-service (IaaS)

In this service, the service provider shares infrastructure resources to support operations done by the end-user. The examples for infrastructure are CPU, memory, network, server etc.

2.1.2 Platform-as-a-service (PaaS)

This service is used to complete the life cycle of building and delivering web applications which are available over the internet. The developers of this service are concerned only with web-based development and do not care about the operating system. Anyone who has the internet can do this and deploy over the internet. Some examples of PaaS model are eBay, Google, iTunes, YouTube. Basically platform-as-a-service provides support for user interface by using HTML, JavaScript, and some rich internet applications. This service is used to make software for customers. We can make combinations of web services using this service.

2.1.3 Software-as-a-service (SaaS)

In the past we used to buy software and install that software in our local computer, but by using SaaS service, a service provider can made software available to end-users over the internet. In this type of service end-users pay for what they use. Many types of software will suit to this service model. For example the types of software that suits this model are email software, video conferencing etc. The prices of the SaaS applications are based on the number of users using that software over the internet.

Wei-Tek, et al. [1] presented an overview survey of current cloud computing architectures, discussed issues that current cloud computing implementations have, and proposed a Service-Oriented Cloud Computing Architecture (SOCCA) so that clouds can interoperate with each other.

Meiko, et al. [2] provided an overview on technical security issues of cloud computing environments. Starting with real-world examples of attacks performed on cloud computing systems (e.g. Amazon EC2 service), Meiko, et al. [2] gave an overview of existing and upcoming threats to cloud computing security. Along with that, Meiko, et al. [2] also briefly discussed appropriate countermeasures to threats.

Cong, et al. [3] proposed an effective and flexible distributed scheme with explicit dynamic data support to ensure the correctness of users' data in the cloud. They rely on erasure-correcting code in the file distribution preparation to provide redundancies and guarantee the data dependability. This construction drastically reduced the communication and storage overhead as compared to the traditional replication-based file distribution techniques.

The below figure [15] shows the layered architecture of IaaS, PaaS, SaaS, and applications.

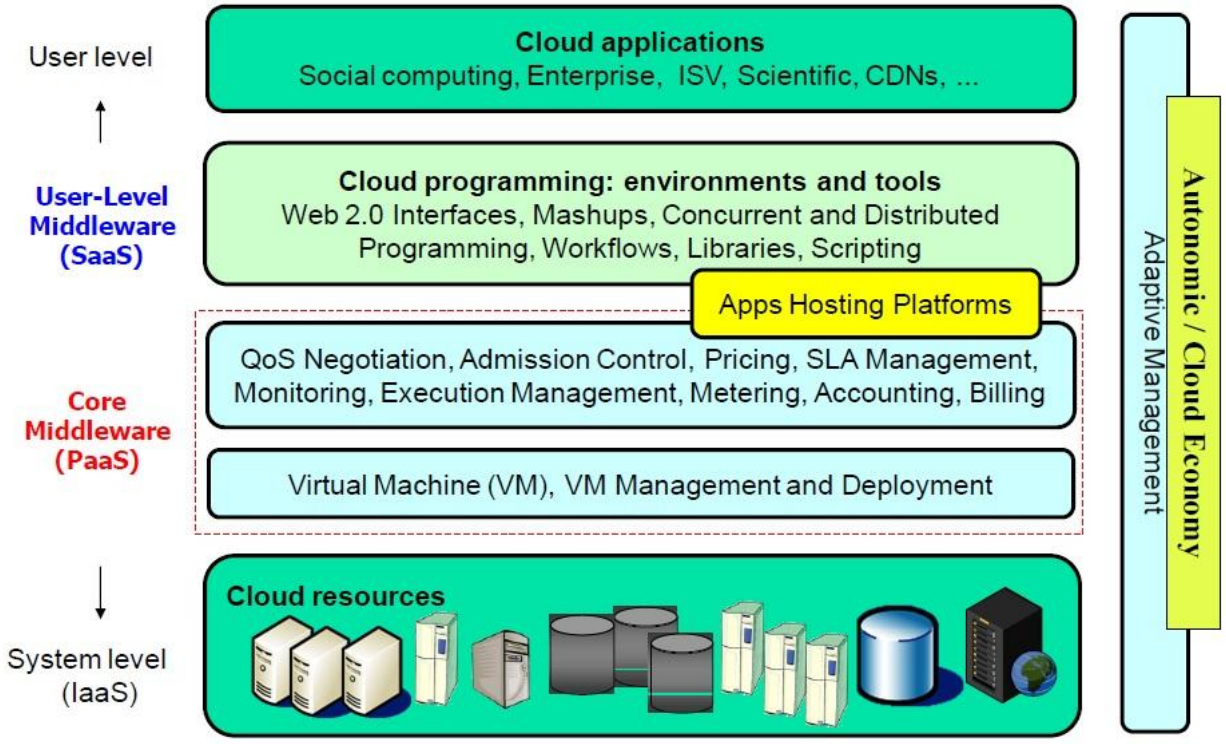

Figure 2: Layered cloud computing architecture

2.2 Sensor-based healthcare

Zhou et al. [4] present a pervasive medical supervision system in a three layer network structure. Different wearable wireless bio-sensor nodes forming the first network layer sample the physiological conditions. Any of those nodes could be set up using a self-organizing or manual configuration as a gateway node for routing sensing data from other nodes. Nodes in the second layer are responsible for reliable data transmission as backbone sensor nodes using mobile phones/PDA depending on two kinds of transmission modes, home mode and nomadic mode. The hospital medical data center acts as the third layer supporting personal services and aggregates patient health data from the other layers. Kang et al. [5] proposed a wearable context aware system for ubiquitous healthcare, composing of two types of wearable sensor systems: a watch type sensor system and a chest belt type sensor system. The context aware system in distributed networks is configured around wearable sensors, wearable computers (e.g. PDA), and internet-based healthcare services.

Fei et al. [6] designed a practical hardware/software platform supporting a telecardiology sensor network (TSN) under a typical healthcare community with many elderly patients. The TSN network performs real-time healthcare data collection and supports medical privacy to secure ECG data transmission in wireless channels. This network adopted the Skipjack-based symmetric crypto for one-hop secure ECG data transmission with low energy and low overhead to preserve the patients' medical privacy. For multiple patient cases, the network acted in a cluster structure to reduce the patient-to-doctor routing overhead and key management with few one-hop hash key chains.

S. Dagtas, et al. [7] designed a wireless health monitoring system in a smart environment, which consists of different sets of sensors; one set of sensors attached to a patient's body (body sensors are used to sense vital sign data such as ECG for performing real-time health monitoring of mobile patients) and another set of sensors are placed over a zigbee network. Each patient has a personal wireless hub to collect sensor signals from patient's body and store them in local server via Zigbee network. Zigbee network name comes from the zigzagging path which is shown in figure 3. The local server analyses the signals from PWH and transmits the data to central server. A doctor can access the information from the central server.

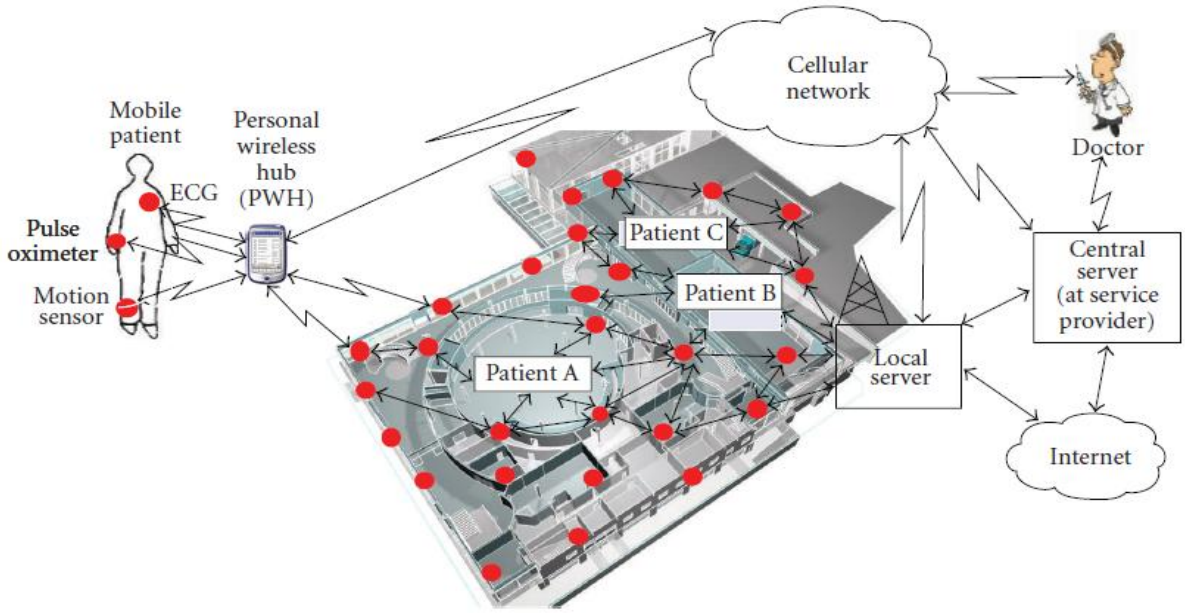

Figure 3: Wireless health monitoring in a smart environment

Sensors fixed over the network have their own limitations, such as installation costs and lack of mobility. Moreover, the central server does not have the needed capacity to store more data from the patients.

In place of sensors we are proposing today's ubiquitous and pervasive mobile phones to play the role of sensors. Instead of having a central server, we are going for cloud storage because the cloud can store more data.

2.3 Scheduling

Scheduling means how tasks are assigned to run on the available resources. It is a key concept in multi-tasking and real-time implementations. There are different scheduling algorithms available. For example, First-in-First-out, Shortest-job-first, Priority-based scheduling etc.

F. Dong, et al. [8] discussed various grid scheduling algorithms from different points of view. In static scheduling [8], information regarding all resources, as well as all the tasks in an application are assumed to be available by the time the application is scheduled. In the static mode, every task comprising the job is assigned once to a resource. After assigning a job to a resource, an estimate of the cost of the computation can be made in advance of the actual execution. Cost estimates based on static information is not adaptive to situations such as when one of the nodes selected to perform a computation fails, becomes isolated from the system due to network failure, or is so heavily loaded with jobs that its response time becomes longer than expected. Unfortunately, these situations are quite possible and beyond the capability of a traditional scheduler running static scheduling policies.

Dynamic scheduling [8] is useful when it is impossible to determine the execution time, direction of branches, and number of iterations in a loop, as well as in the case where jobs arrive in a real-time mode.

The advantage of dynamic load balancing over static scheduling is that the system need not be aware of the run-time behavior of the application before execution. It is particularly useful in a system where the primary performance goal is maximizing resource utilization, rather than minimizing runtime for individual jobs. If a resource is assigned to too many tasks, it may invoke a balancing policy to decide whether to transfer some tasks to other resources and which tasks to transfer.

Scheduling decisions change dynamically according to the previous, current, and/or future resource status in Adaptive scheduling [8].

There are three kinds of adaptive scheduling:

Resource Adaptation: In this available resources are grouped first into disjoint subsets according to the network delays between the subsets. Then, inside each subset, resources are ranked according to their memory size and computational power. Finally, an appropriately-sized resource group is selected from the sorted lists.

Dynamic Performance Adaptation: Dynamic performance adaptation means changing scheduling policies, finding proper number of resources, and work-load distribution.

Application Adaptation: In application adaptation, each scheduler is application specific, and they can communicate through interfaces. In most realistic scheduling situations, the information available to execute a process is uncertain. Factors such as time to execute a task, the time resources are available are all uncertain. Hence a probabilistic scheduling algorithm is required.

W. Zaho, et al. [9] describes a heuristic approach to dynamically schedule tasks in a real-time system where tasks have deadlines and resource requirements. The basic approach assumes that each node of the system contains a local scheduler, a bidder and a dispatcher. Simulation studies of W. Zaho, et al. [9] paper show that the algorithm is close to optimal.

There is another algorithm based on heuristics to schedule tasks (that have deadline) which was developed by K. Ramamritham, et al. [10]. This algorithm mainly focused on focused addressing and bidding algorithms to schedule tasks which are not locally scheduled. The results of this algorithm show it is very efficient in hard real-time environment.

I. Rao, et al [11] developed a probabilistic scheduling algorithm in a Grid system where the scheduling of a task depends on current resource utilization and number of jobs in the queue which is dynamic information. This algorithm consists of independent and indivisible jobs.

However, the algorithms mentioned by W. Zaho, et al. [9], . Ramamritham, et al. [10] , I. Rao, et al [11] are not tested in the cloud environment which are going to be tested in this thesis using CloudSim simulation tool kit.

CHAPTER III

SYSTEM ARCHITECTURE

In this work we concentrate on remote healthcare monitoring which makes use of emerging technologies to fundamentally change the relationship between health care providers and patients. The proposed architecture has the potential for improving mobile health care system with minimal investment and can meet real-time requirements.

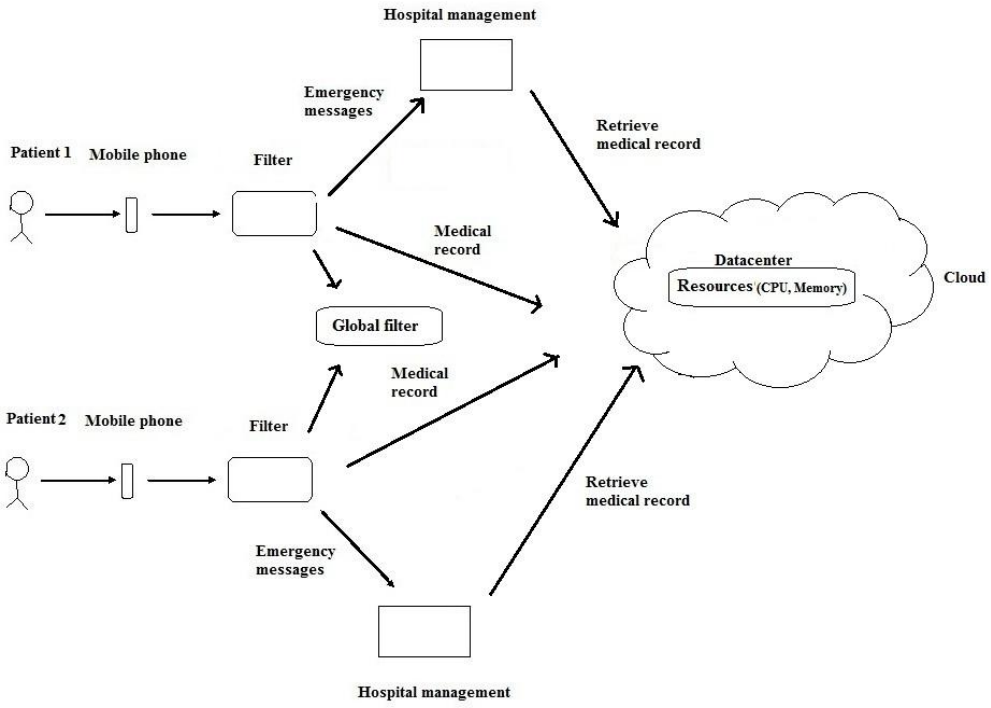

Figure 4: Real-Time and mobile health care architecture

In the above architecture a set of biomedical sensors are attached to the patient's body. As described earlier, some responses to the sensed environment require a real-time response whereas in other cases, the sensed data does not call for a real-time response. The signals from the sensors are sent to a cell phone. Once the cell phone receives a signal from the sensors it forwards them to a filter, which is located between a set of patients in a geographical area and the cloud. A filter is installed for a group of patients in a network. The filter will measure, analyze and process parameters such as temperature, blood pressure, heart beat, etc. The response is sent to the hospital management about the patient's emergency condition and a copy of the response will be stored in the cloud. The parameters which do not require a real-time response are analyzed on any one of the filters and the medical record will be sent over to the cloud. If a doctor wants to access the record of a particular patient he can access it from the cloud.

The outline of the proposed model is shown in figure 5. From the filter the signals will go to the hospital management system and to the cloud. If there is an emergency situation the signal will go to the hospital management and a copy of the patient information will be stored in the cloud. If the situation is normal, a copy of the patient information will be stored in the cloud for later access.

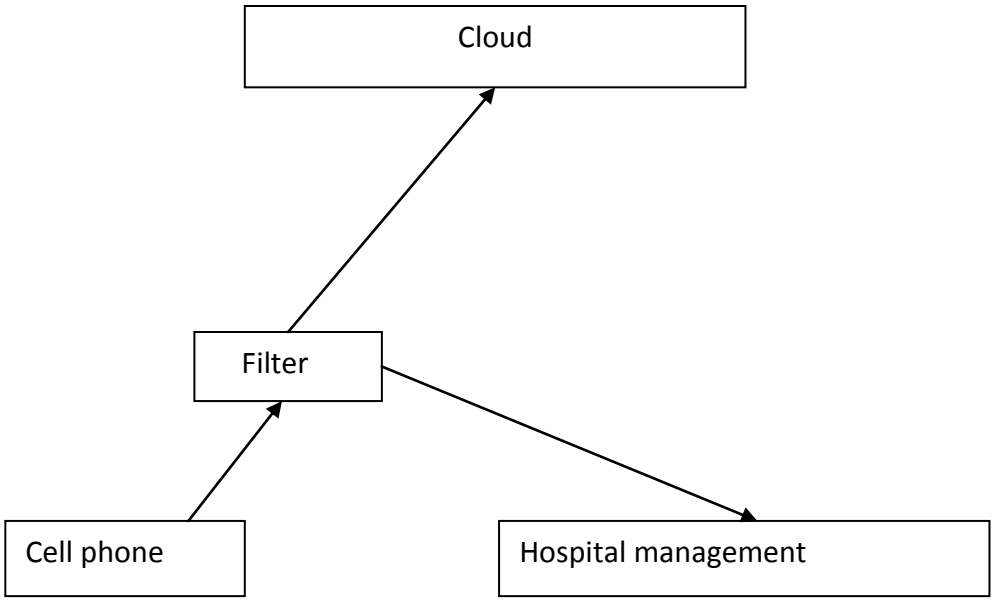

Figure 5: Simplified architecture

3.1 Problem Description

Many tasks from different patients arrive at the same time to the filters. A scheduling algorithm schedules the tasks based on the resources (memory, CPU etc) available. The scheduling algorithm ensures that the time constrained tasks get scheduled and executed before the deadline. If a task cannot be scheduled on one filter we need to look for other filters. The tasks which are not time constrained are scheduled on any one of the filters based on a probabilistic function.

CHAPTER IV

DESIGN AND IMPLEMENTATION

4.1 Introduction

The objective of our "task resource probabilistic scheduling algorithm" (which runs on the filters) is to find the target filter for every task coming from a cell phone based on the criticality of the patient, so that for critical patients, the timing deadlines are met. Sensor data (set of tasks) from a cell phone reach the filter (called Broker in CloudSim). The filter tries to schedule tasks based on the resources available. Resources (e.g. memory, CPU) are available at the cloud datacenter. A Datacenter is a class which will carry all infrastructure level resources (e.g. CPUs). Through the filters at each location, the virtual machines will be created on the datacenters.

The tasks (called cloudlets in CloudSim) arriving from cell phones will be scheduled on virtual machines of the datacenters through the filters. Globalfilter is a class, which is placed in a geographical area between all local filters and the cloud or it can be on a local filter, will be connected to all the local filters. If a cloudlet cannot be scheduled on a local filter due to lack of resources, in order to schedule that cloudlet, the information about the cloudlet will be sent over to the Globalfilter from the local filter.

The Globalfilter will send bids to all local filters. A bid is a policy to select a local filter, which can meet the cloudlet's deadline. Each filter responds to the bid. A local filter with the lowest bid value is selected by Globalfilter to schedule the cloudlet. The tasks which do not have any deadline can be scheduled on any of the local filters. Although the filters do the scheduling, the tasks are executed on the cloud. The selection process is explained detail later in this chapter.

4.2 Probabilistic based scheduling

Abbreviations and definitions:

$ST(t_i)$ = Starting time of cloudlet t_i - Time at which the cloudlet t_i should start execution

t_{ia} = Arrival time of cloudlet t_i

$E(t_i)$ = Execution time of cloudlet t_i

$D(t_i)$ = Deadline time of cloudlet t_i

EAT = Earliest available time of CPU (initially it is equal to the time when the simulation gets started, as there only one CPU per datacenter).

NewEAT (t_i) = this will replace the current EAT if a current cloudlet t_i is scheduled.

DRDR(t_i) = Dynamic Resource Demand Ratio, indicates the likelihood of the cloudlet t_i to be scheduled on the local virtual machine.

$H(t_i)$ = The objective of this heuristic function $H(t_i)$ is to select a cloudlet t_i to schedule next on the virtual machine, among a set of cloudlets.If we have a set of cloudlets, the cloudlet t_i, to be scheduled first will be the one with the minimum $H(t_i)$.

info(t_i) – cloudlet information for cloudlet t_i. Each cloudlet has cloudlet ID, arrival time, deadline time, execution time.

VIM(t_i) – virtual machine assigned to cloudlet t_i (if a local filter cannot schedule a cloudlet t_i, we need to select another virtual machine through other filters to schedule).

$PX(t_i)$ = probability that starting time of cloudlet t_i does not conflict with previously scheduled cloudlet t_{i-1}'s execution time.

$PY(t_i)$ = probability that no previously scheduled cloudlet t_i has starting time that conflicts with newly arrived cloudlet t_{i+1}'s starting time.

$F(v_k)$ = A virtual machine v_k having minimum $F(v_k)$ is selected as a target node for scheduling cloudlet t_i. $F(v_k)$ is a function to select which virtual machine can schedule the cloudlet t_i to meet the deadline.

Bidvalue(v_k) = A virtual machine v_k having minimum *Bidvalue*(v_k) is selected as a target node for scheduling. *Bidvalue*(v_k) indicates the likelihood, that the cloudlet can be guaranteed to meet the deadline on the virtual machine. It is calculated using *EAT* of CPU in a virtual machine and execution time of a cloudlet t_i.

When cloudlets $\{t_1, t_2, \ldots, t_n\}$ arrive at each filter, calculate the starting time of each cloudlet t_i using $ST(t_i) = EAT$. The job of the filter is to schedule the cloudlets on the virtual machine. Initially the schedule $\langle s_1, s_2, \ldots, s_m \rangle$ is empty, where each s_i is a scheduled cloudlet. The scheduling is done based on the resources (CPU) available at each virtual machine. A filter calculates the likelihood of the cloudlets to be scheduled on the local virtual machine, using

$$DRDR(t_i) = E(t_i) / (D(t_i) - EAT)$$

If $DRDR(t_i)$ is less than or equal to one, then the algorithm schedules the cloudlet on the local virtual machine. If we have a set of cloudlets, the cloudlet t_i, to be scheduled first will be the one with the minimum $H(t_i)$ and be selected using

$$H(t_i) = DRDR(t_i) + [D(t_i) - (ST(t_i)+E(t_i))] + E(t_i) = DRDR(t_i) + D(t_i) - ST(t_i).$$

Weights can be associated with each term to form $w_1 DRDR(t_i) + w_2 D(t_i) - w_3(ST(t_i))$. In our case $w_1 = w_2 = w_3 = 1$. This equation selects cloudlet t_i to schedule next on the virtual machine, among a set of cloudlets. This equation selects the cloudlet with the earliest deadline. If other factors such as the cloudlet with the least computation time is to be selected, the weights need to be modified.

If $DRDR(t_i)$ is greater than one, the information of cloudlet t_i, is passed to the GlobalFilter. The GlobalFilter send bids to all local filters, to determine which virtual machine can schedule the cloudlet t_i to meet its deadline. The $Bidvalue(v_k)$ for all virtual machines is calculated using $Bidvalue(v_k) = EAT / E(t_i)$. The virtual machine having the minimum $Bidvalue(v_k)$ is selected as a target machine for the cloudlet t_i.

If a cloudlet t_i does not have any deadline, it can be scheduled on any of the filters. Here we use probability based scheduling unlike deterministic scheduling for cloudlets with deadlines, because deterministic scheduling is more time consuming. The probabiliy based scheduling gives only the estimation of the appropriate target filter and cloudlets without deadlines get scheduled on estimated target filter. The selection of a target filter depends on two factors, one is probability $PX(t_i)$, where $PX(t_i) = (EAT) / t_{ia}$, which is the probability that the starting time of one cloudlet does not conflict with a previously scheduled cloudlet execution time. The second one is probability $PY(t_i)$, where $PY(t_i) = (1 – E(t_i)) / EAT$, which is the probability that no previously scheduled cloudlet has a starting time that conflicts with a newly arrived cloudlet starting time. For each cloudlet, the product $F(v_k) = PX(t_i) * PY(t_i)$ is obtained at each filter. The cloudlet is scheduled for execution on the filter that gives the minimum product value $F(v_k)$.

The figure below shows our proposed model:

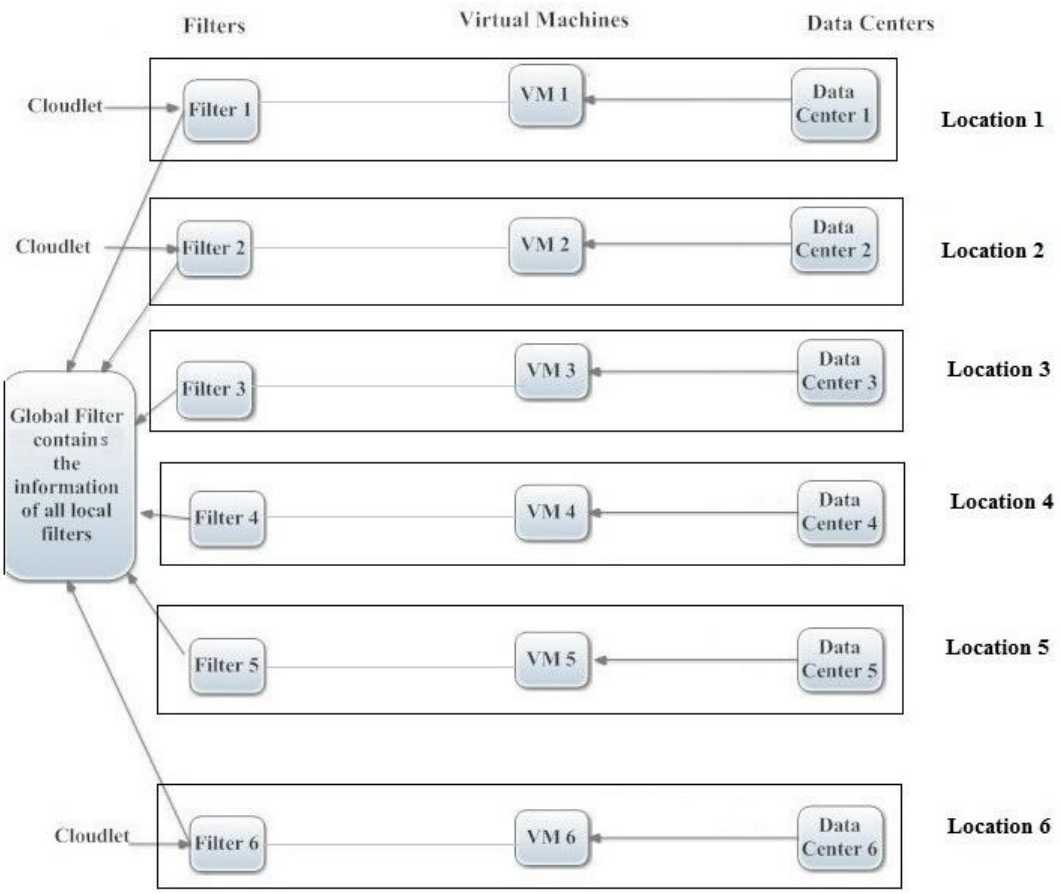

Figure 6: Proposed model

4.3 Proposed probabilistic scheduling algorithm

if (cloudlet t_i has a deadline $D(t_i)$)

goto part 1;

if (cloudlet t_i does not have any deadline, that is, $D(t_i) = \infty$)

goto part 3;

Part 1:

When cloudlets arrive at each filter, the job of the filter is to schedule the cloudlets on the virtual machine. The scheduling is done based on the resources available at each virtual machine. A filter calculates the likelihood of the cloudlets to be scheduled on the local virtual machine, using $DRDR(t_i)$. If $DRDR(t_i)$ is less than or equal to one, then the algorithm schedules the cloudlet (if we have a set of cloudlets, the cloudlet t_i, which needs to be scheduled first will be selected using $H(t_i)$) on the local virtual machine. If $DRDR(t_i)$ is greater than one, the information of cloudlet t_i is passed to the GlobalFilter.

schedule = $\langle s_1, s_2, \ldots, s_m \rangle = \langle\ \rangle$ //schedule is empty
cloudlet-set = $\{t_1, t_2, \ldots, t_n\}$ where t_i is cloudlet i
while cloudlet-set ≠ {} **do**
begin
$ST(t_i) = EAT$ //calculate starting time for each cloudlet in cloudlet-set//
$DRDR(t_i) = E(t_i) / (D(t_i) - EAT)$ // Calculate DRDR for each cloudlet//
if ($DRDR(t_i) > 1$) **then** // cloudlet cannot be scheduled locally//
begin
info(t_i) → GlobalFilter // Send cloudlet information to GlobalFilter//
goto part2.
end
else begin
for i = 1 to n **do** //for all the cloudlets in cloudlet-set//
$NewEAT(t_i) = ST(t_i)+E(t_i)$
//NewEAT is the earliest time when the CPU will be available//
$H(t_i) = DRDR(t_i) + [D(t_i) - (ST(t_i)+E(t_i))] + E(t_i)$
end forloop
select t_i such that t_i = **min**($H(t_1), H(t_2), \ldots, H(t_n)$) //schedule cloudlet with min($H(t_i)$) //
begin
cloudlet-set = $\{t_1, t_2, \ldots, t_n\} - \{t_i\}$ // remove cloudlet t_i from the cloudlet-set //
schedule = $\langle$ schedule | t_i $\rangle$ // | stands for the operation of appending//
$EAT = NewEAT(t_i)$
end
end
end

Part2:

Note: Accessing virtual machines can be done only through respective filters

If the cloudlet t_i cannot be scheduled locally, GlobalFilter send bids to all local filters, to see which virtual machine is most feasible to schedule the cloudlet t_i. The *Bidvalue*(v_k) for all virtual machines is calculated using *EAT* of CPU in every virtual machine, and execution time of cloudlet t_i. The virtual machine having the minimum *Bidvalue*(v_k) is selected as a target machine for the cloudlet t_i.

begin
for k = 1 to n **do** //for all the virtual machines in GlobalFilter //
Bidvalue(v_k) =*EAT* / *E*(t_i)
end forloop
select v_k **such that** v_k = **min(***Bidvalue*(v_1), *Bidvalue*(v_2), …, *Bidvalue*(v_n)) //select virtual machine with min(*Bidvalue*(v_k) //
begin
VIM(t_i) = v_k // assign selected virtual machine v_k to cloudlet t_i
cloudlet-set = {t_1, t_2, …, t_n} − {t_i} // remove cloudlet t_i from the cloudlet-set //
schedule = ⟨ schedule | t_i ⟩ // | stands for the operation of appending//
EAT = *ST*(t_i)+*E*(t_i)
end
end

Part3:

After a local filter receives a cloudlet t_i which does not have any deadline, it can be scheduled on any of the filters. The selection of a target filter depends on two factors, one is probability $PX(t)$, which is the probability that the starting time of one cloudlet does not conflict with a previously scheduled cloudlet execution time. The second one is probability $PY(t)$, which is the probability that no previously scheduled cloudlet has starting time that conflicts with newly arrived cloudlet starting time.

For each cloudlet, the product $PX(t)* PY(t)$ is obtained at each filter. The cloudlet is scheduled for execution on the filter that gives the minimum product value $F(v_k)$.

begin
for k = 1 to n **do** //for all the virtual machines in Globalfilter

$F(v_k) = PX(t_i) * PY(t_i)$
$PX(t_i) = (EAT) / t_{ia}$
$PY(t_i) = (1 – E(t_i)) / EAT$
end forloop
select v_k **such that** v_k = **min{** $F(v_1), F(v_2), \ldots, F(v_n)$ } //select virtual machine with min($F(v_k)$) //
begin
$VIM(t_i) = v_k$ // assign selected virtual machine v_k to cloudlet t_i
cloudlet-set = $\{t_1, t_2, \ldots, t_n\} – \{t_i\}$ // remove cloudlet t_i from the cloudlet-set //
schedule = ⟨ schedule | t_i ⟩ // | stands for the operation of appending//
$EAT = ST(t_i)+E(t_i)$
end
end

CHAPTER V

SIMULATION AND RESULTS

5.1. Introduction

The objective of the simulation is to determine the performance of the proposed probabilistic scheduling approach for both emergency and non-emergency tasks in a mobile health care system in a cloud environment. Simulation is done on a system having six Filters (Brokers) at six different locations. The CloudSim simulation tool [15] is used for the simulation, which is a tool kit for modeling and simulation of cloud computing environments. It supports behavior modeling of cloud system components, such as data centers, virtual machines, and resource provisioning policies. CloudSim is being used to investigate Cloud resource provisioning and energy efficient management of data center resources [15].

The reason behind choosing CloudSim is that, it allows cloud providers to test their services in a repeatable and controlled environment free of cost and to tune the performance bottlenecks before deploying on real Clouds. On the simulation side, the simulation environment allows different kinds of resource allocation under different load distributions. CloudSim provides a good simulation framework for emerging cloud computing applications, so that researchers can investigate different design issues without getting concerned about low level details.

5.2 CloudSim simulation tool

The layered architecture of CloudSim is shown below

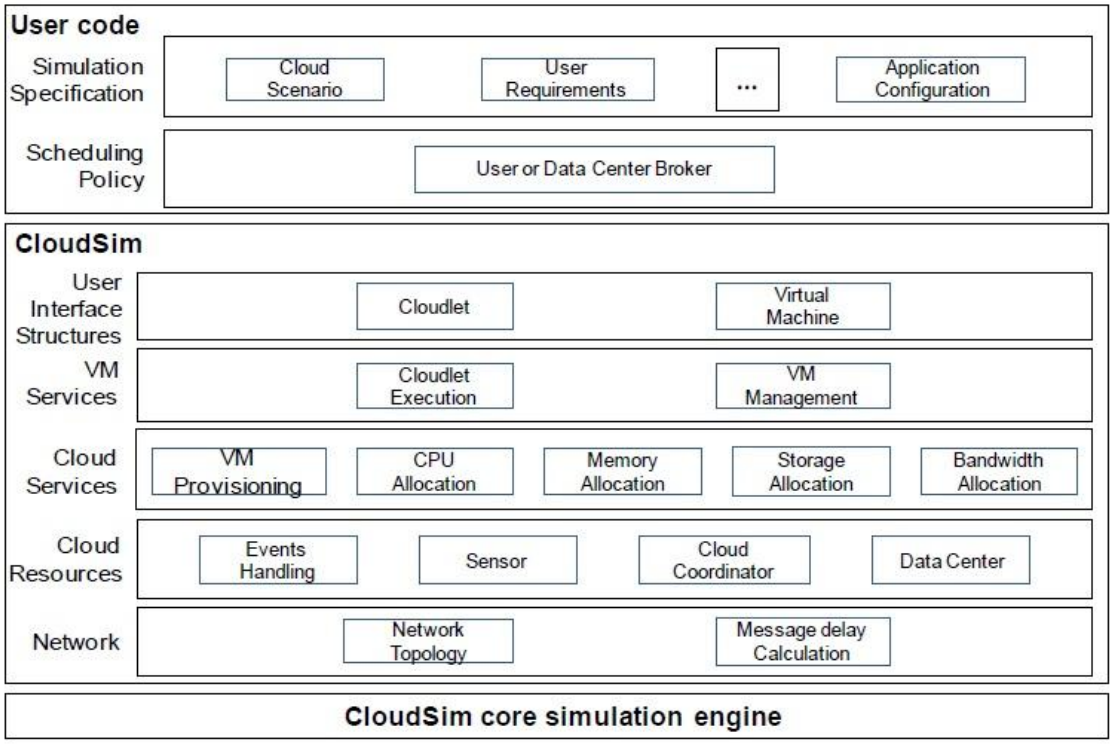

Figure 7: Layered CloudSim architecture

Novel features of CloudSim are:

1) Providing simulation for cloud computing environments like datacenters, virtual machines, etc.
2) Providing platform allocation policies, on resources.
3) Providing simulation for network connections.

Unique features of the CloudSim are:

1) Managing independent, co-hosted virtualized services on a data center node.
2) Easy to change between space-shared and time-shared allocation of tasks on resources.

In CloudSim the fundamental classes which are building blocks of the simulation in this thesis are BwProvisioner, CloudCoordinator, Cloudlet, CloudletSchedular, Datacenter, DatacenterBroker, etc.

Cloudlet: Cloudlet is a task arriving from outside to be scheduled or to be executed.

Datacenter: This class models the core infrastructure level services.

DatacenterBroker: This class negotiates between cloud providers and resources. This broker negotiates on behalf of the providers to select which resource is most suitable for the providers.

Vm: This class models a VM, which is managed and hosted by a Cloud host component.

Vmschedular: This is an abstract class implemented by a Host component that models the policies (space-shared, time-shared) required for allocating processor cores to VMs. The functionalities of this class can easily be overridden to accommodate application-specific processor sharing policies. Figure 8 show the overall class diagram of CloudSim.

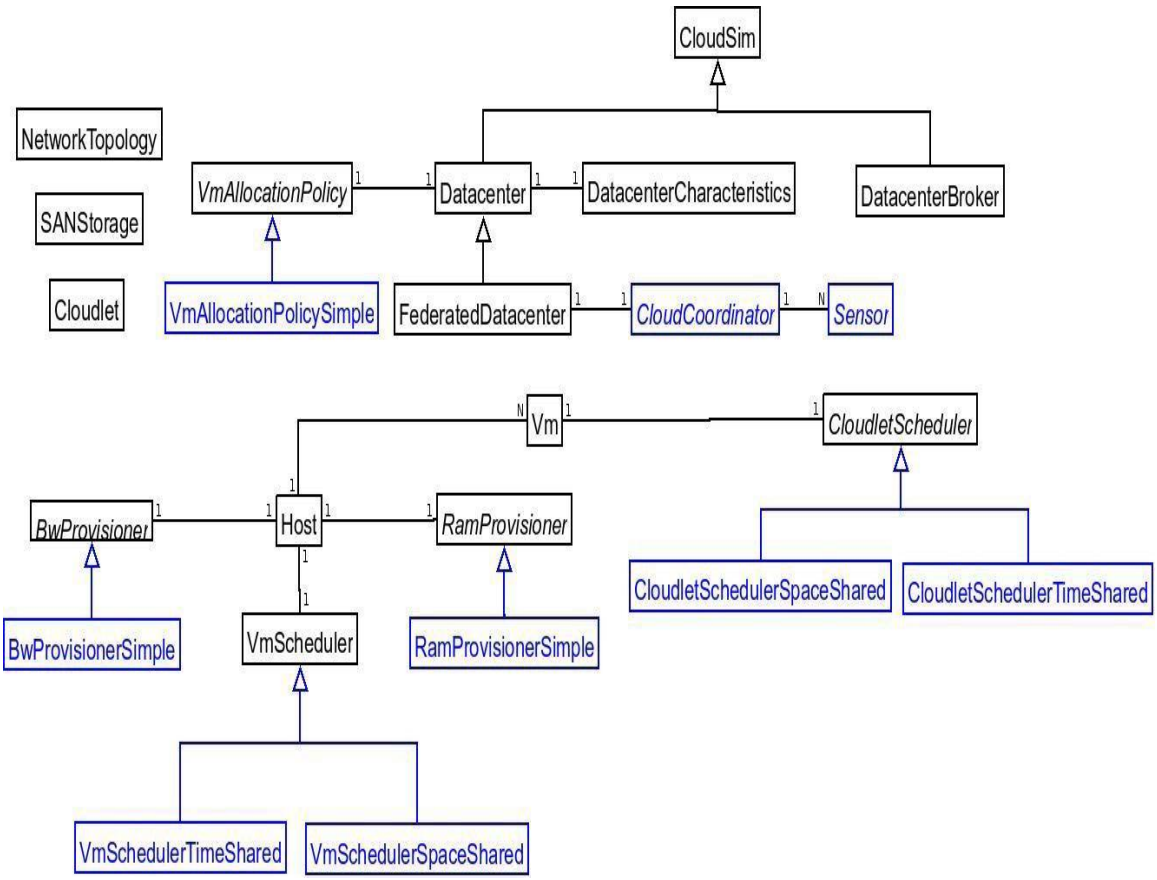

Figure 8: CloudSim class diagram

CloudSim: This is the main class, which is responsible for managing event queues and controlling sequential execution of simulation events. Every event that is generated by the CloudSim entity at run-time is stored in the queue called future events. The events which are scheduled are removed from the future events queue and transferred to the deferred event queue.

SimEntity: This is an abstract class, which represents a simulation entity that is able to send messages to other entities and process received massages as well as fire and handle events.

5.3 Simulation model

Our simulation consists of a system having six Filters (Brokers) at six different locations. Every filter gets access to one virtual machine (a virtual machine is created on a datacenter). The first step of the simulation is creating six datacenters at six different locations. Each datacenter has a name. After the creation of datacenters, we create filters at each location. All these filters are maintained by a GlobalFilter.

Through the filters, we create a virtual machine on a datacenter at each one of the locations. In CloudSim, there are two ways to create virtual machines (VMs) on a datacenter; we can create types of VMs in advance and use VMallocation policy to allocate VMs to a datacenter, or a VM can be statically allocated to a desired datacenter. In this thesis we chose to statically create a VM on a particular datacenter. For example, VM1 is created in Datacenter1 through Filter1 in figure 9(a). Likewise the remaining five datacenters, filters, and VMs are created. Note that every VM creation on a datacenter is done through the particular filters at that location

The simulation model is shown in figure.9(a)

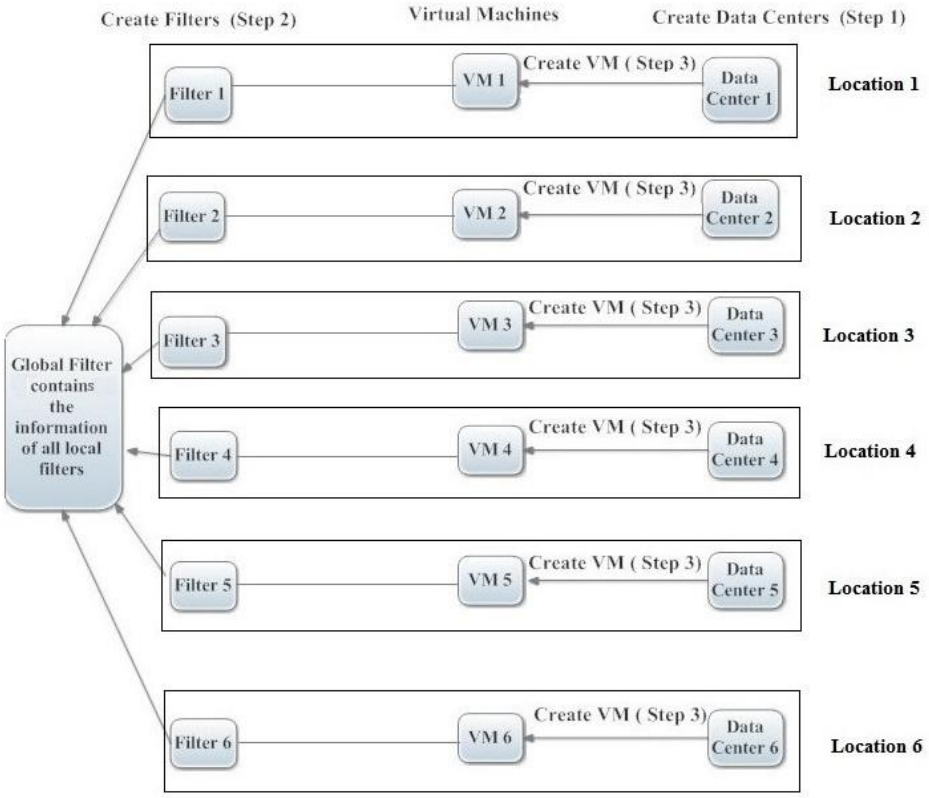

Figure. 9(a): Simulation model

The simulation model shown above reflects our proposed architecture. Cloudlets will arrive in a random fashion at each filter. The job of the Filter is to schedule the Cloudlets on a deadline basis.

The scheduling of the cloudets is shown in figure. 9(b)

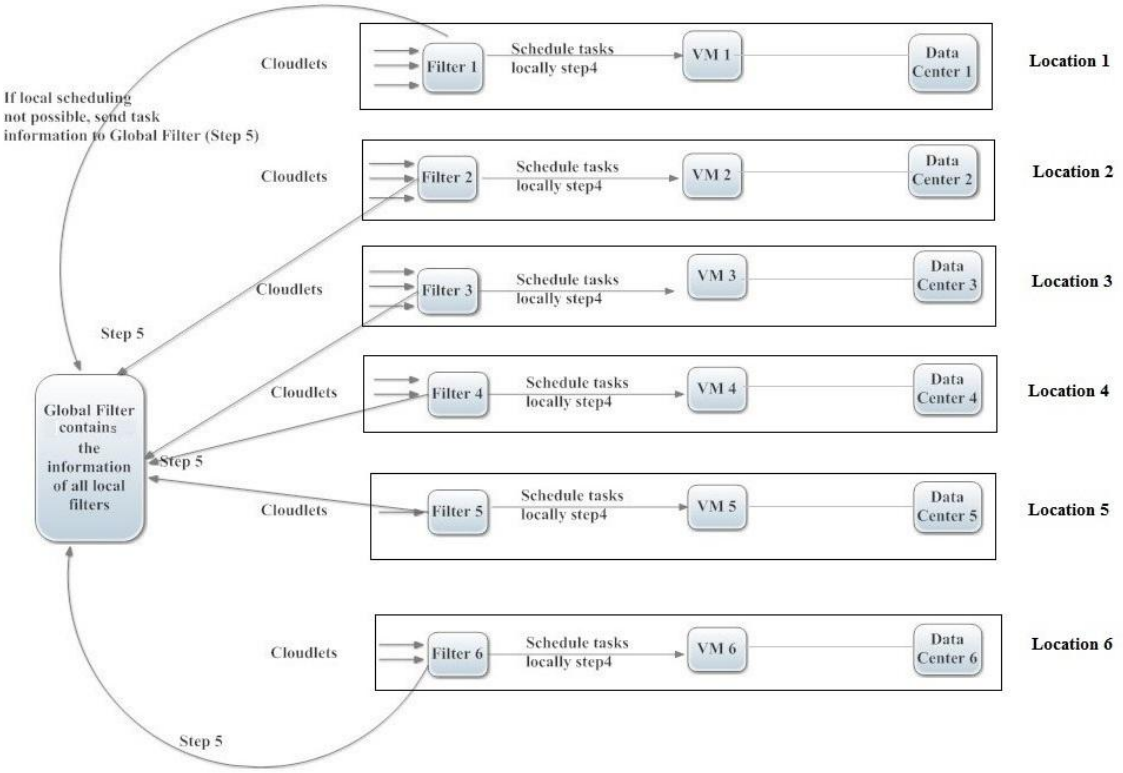

Figure. 9(b): Scheduling the cloudlets

If Cloudlets arrive at different filters we need to calculate the starting time of each Cloudlet using function $ST(T) = EAT$

After calculating the starting time we need to make sure the Dynamic Resource Demand Ratio (DRDR) must be less than or equal to 1. This DRDR will guide in scheduling the jobs without any overlap.

If that ratio is less than or equal to 1, then calculate the New Earliest Available Time (*NewEAT*) (addition of initial earliest available and execution time of the particular Cloudlet) for each Cloudlet. This will replace the current Earliest Available Time (*EAT*) if that Cloudlet is scheduled. The next step is to calculate the heuristic function for every Cloudlet which will identify a Cloudlet which needs to be scheduled first. The Cloudlet with the minimum heuristic function value will be scheduled and the process will continue if there are more Cloudlets in the Cloudlet set.

If a local filter cannot schedule the Cloudlet, then information of that Cloudlet will be sent over to the GlobalFilter. Then GlobalFilter will send the Cloudlet details to all local filters. The Cloudlets with no deadline are probabilistically scheduled using probability function ($PX(t_i)*PX(t_i)$).

5. 4 Percentage of cloudlets that meet deadline at both local and non-local filters

5.4.1 Simulation parameters

For every five seconds, 2 cloudlets arrive at each filter. The execution time for each cloudlet is fixed (30 seconds). The deadline time is 50 seconds initially and the deadline time for each cloudlet increases by 20 for every other cloudlet that come in.

5.4.2 Results

If we repeat the iteration for 50 iterations, from the below figure we can see that if the average number of cloudlets at each filter increases then the percentage that will meet the deadline decreases in a six node system. If the average number of cloudlets at each filter are 80 then the percentage that meet the deadline are only 2.5% of 80 cloudlets, at both local and non-local filters. On the other hand, if there are only 5 cloudlets at a time at each filter, 60% of them meet their deadlines.

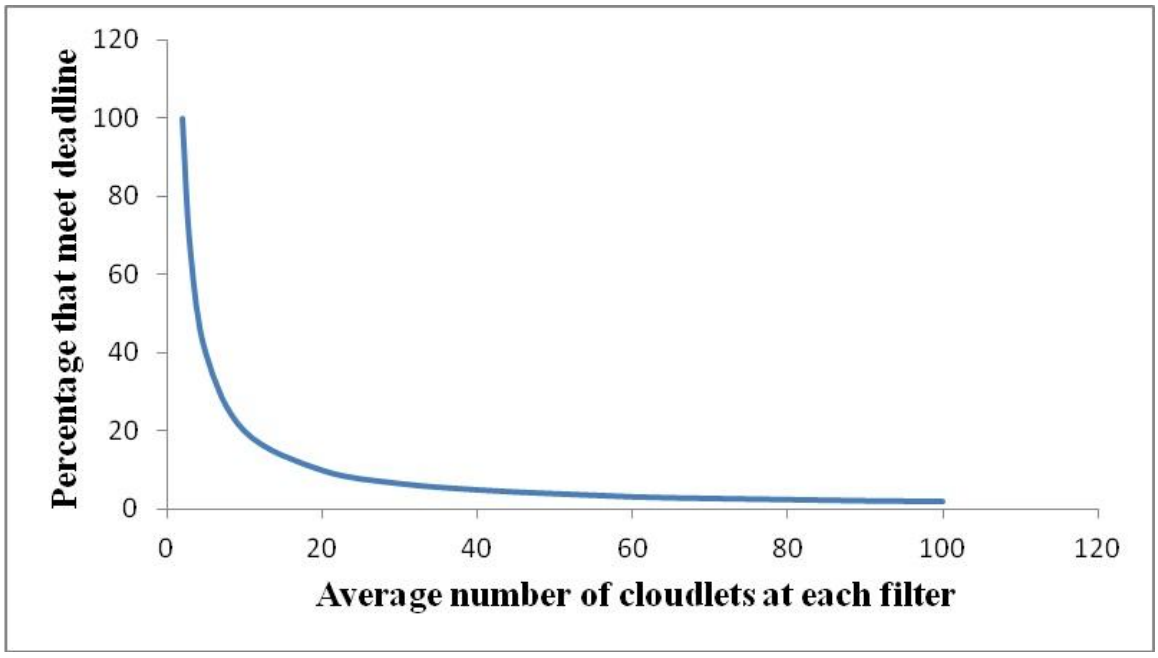

Figure 10: Relationship between number of cloudlets and percentage that meet deadline

5.5 Impact of increasing deadline on fixed number of cloudlets

5.5.1 Simulation parameters

In this simulation there are a fixed number of cloudlets at each filter with a fixed deadline and a fixed execution time .The execution time is fixed for all (10 seconds). The cloudlets all arrive at the same time to each filter.

5.5.2 Results

In the below figure we can see there is a linear relationship between the deadline and the percentage of cloudlets that will meet the deadline. As the number of cloudlets at each filter increase, the percentage of cloudlets that will satisfy the deadline decrease. Increasing the deadline for a fixed number of cloudlets also increases the percentage of cloudlets that will meet the deadline

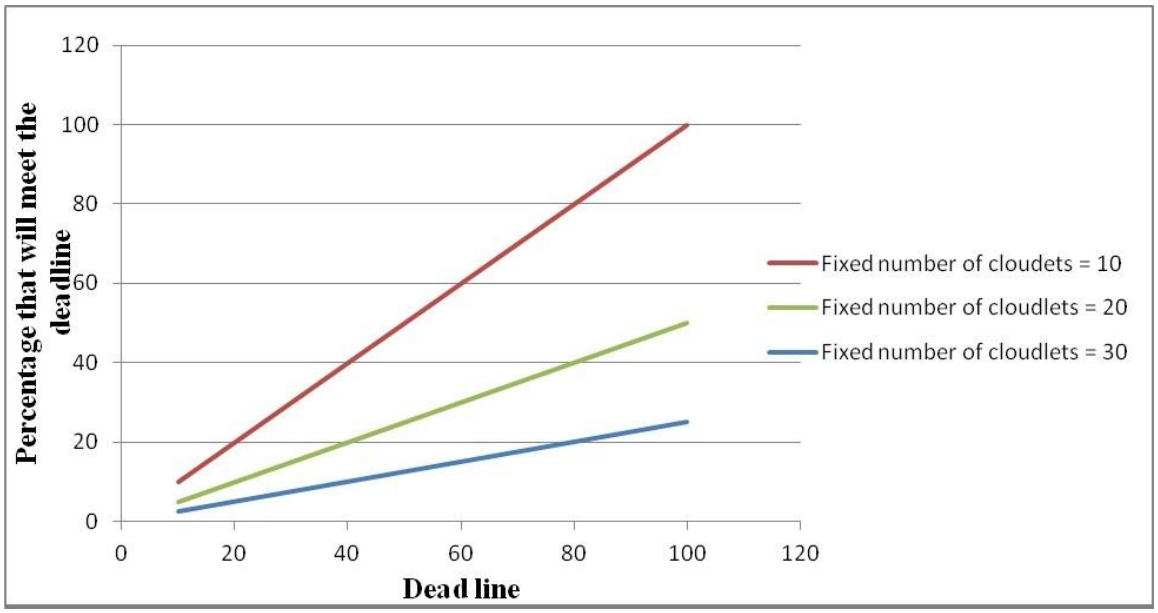

Figure 11: Comparison between fixed number of cloudlets with increasing deadline

5.6 Non-emergency cloudlets scheduling probability

5.6.1 Simulation parameters

We next consider a fixed number of non-emergency (i.e. deadline=∞) cloudlets in a group of tasks. In this case also, the cloudlets arrive at the same time to each filter. We simulated total number of tasks 30, 40 and 50 with 10, 20 and 30 tasks in each of these cases to be tasks with no deadlines. Hence, at one extreme case scenario, 100% of the tasks had no deadlines and at the other extreme only 20% of tasks had no deadlines. The execution time for these tasks were fixed with 30 seconds

5.6.2 Results

In the below figure, the y-axis show only the tasks that do not have a deadline that will be scheduled at the local filter. If the number of cloudlets with deadline= ∞ are 10 out of a total of 30 cloudlets then only 60% of tasks will be scheduled on the local filter. On the other hand if all the 30 cloudlets have a deadline= ∞ then 100% of the tasks will be scheduled on the local filter.

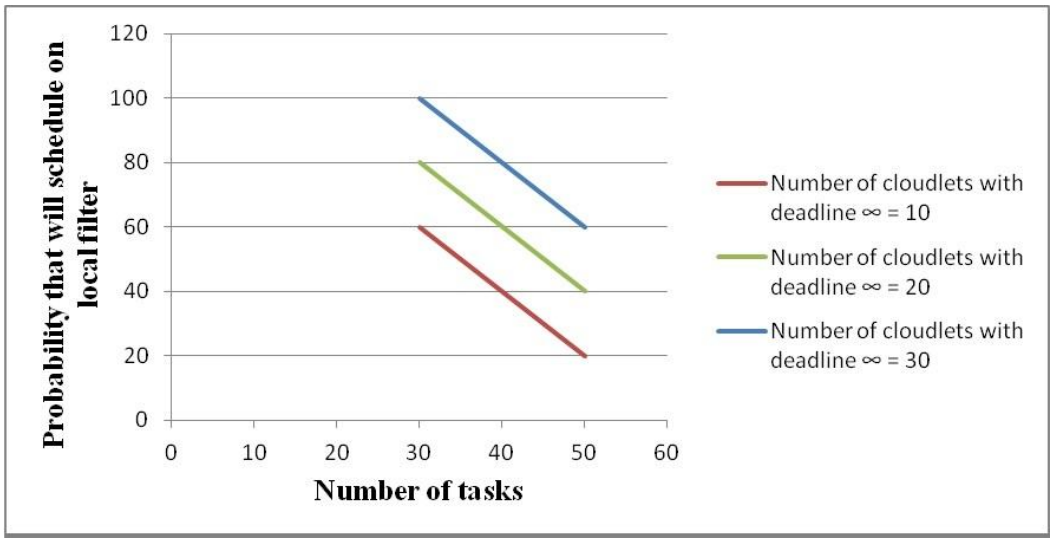

Figure 12: Comparison between different number of cloudlets whose deadlines are infinity

Simulation results of the scheduling algorithm show that each filter can handle multiple tasks at a time and with an increase in the average number of tasks at a filter, the percentage of tasks that will meet the deadline decrease. Furthermore, with an increase in the deadline of tasks, the percentage of tasks that will meet the deadline also increases. As non-emergency tasks do not have a time constraint, they will always find a filter to execute.

CHAPTER VI

CONCLUSION

In this thesis we have proposed a filter-based architecture for scheduling real-time tasks in a cloud system. We propose a probabilistic scheduling algorithm for real-time tasks in a mobile health care system, using the CloudSim simulation tool kit. Our simulations were done on a 6 node filter system. The proposed algorithm deals with both emergency tasks and non-emergency tasks. Emergency tasks have deadlines, so they should execute before the deadline and hence they should find a filter to be scheduled before reaching the deadline. Non-emergency tasks do not have any deadline; they can be scheduled on any of the filters.

Simulation results of the scheduling algorithm show that with an increase in the average number of tasks at a filter, the percentage of tasks that will meet the deadline decrease. Furthermore, with an increase in the deadline of tasks, the percentage of tasks that will meet the deadline also increases. As non-emergency tasks do not have a time constraint, they will always find a filter to execute.

The future of mobile health monitoring with the combination of cloud computing is expected to see lot of technological advances. The focus on mobile health monitoring in a cloud environment will diminish the need of the physical presence of a patient and provide emergency healthcare. The CloudSim toolkit can only simulate the services and algorithms in a repeatable and controlled environment. Future work may include real time implementations. This includes determining the optimized location of filters and the number of filters in such a system. One big research area is the security of such systems as it deals with patient information. Improved scheduling algorithms are also needed. The division of tasks between the filters and virtual machines in the cloud is another topic for further research.

REFERENCES

[1] Wei-Tek Tsai, Xin Sun and Janaka Balasooriya. "Service-Oriented Cloud Computing Architecture". *Seventh International Conference on Information Technology,* pp. 684-689, 2010.

[2] Meiko Jensen, J¨org Schwenk, Nils Gruschka, et al. "On Technical Security Issues in Cloud Computing". *IEEE International Conference on Cloud Computing*, pp. 109-116, 2009.

[3] Cong Wang, Qian Wang, and Kui Ren, et al. "Ensuring Data Storage Security in Cloud Computing". *Quality of Service, 2009. IWQoS. 17th International Workshop*, pp. 1-9, 2009.

[4] B. Zhou, C. Hu, M. Q. H. Meng, et al. "A wireless sensor network for pervasive medical supervision" in *Proc. ICIT '07 IEEE Int. Conf. Integration Technology*, pp. 740-744, 2007.

[5] D. O. Kang, H. J. Lee, E. J. Ko, et al. "A wearable context aware system for ubiquitous healthcare," *in Proc. 28th IEEE EMBS Annual Int. Conf*, pp. 5192-5195, 2006.

[6] F. Hu, M. Jiang, M. Wagner, et al. "Privacy-preserving telecardiology sensor networks: toward a low-cost portable wireless hardware/software codesign" *IEEE Trans. Inform. Technol. Biomed.*, vol. 11, pp. 619-627, 2007.

[7] S. Dagtas, G. Pekhteryev, Z. Sahinoglu, et al. “Real_time and secure Wireless Health Monitoring”. *International Journal of Telemedicine and Applications*, pp. 1-10, 2008.

[8] Fangpeng Dong and Selim G. Akl. “Scheduling Algorithms for Grid Computing: State of the Art and Open Problems”. Microsoft Technical Report *2006-504*, pp. 1-55, 2006.

[9] W. Zhao, K. Ramamritham and J. A. Stankovic. “Scheduling Tasks with Resource Requirements in Hard Real-Time Systems”. *IEEE Transactions on Software Engineering.*, Vol. 5, pp. 564-577, 1987.

[10] K. Ramamritham and J. A. Stankovic. “Distributed scheduling of Tasks with Deadlines and Resource Requirements*”. IEEE Transactions on Computers*., Vol. 38, pp. 1110-1123, 1989.

[11] Imran Rao and Eui-Nam Huh. “A probabilistic and adaptive scheduling algorithm using system-generated predictions for inter-grid resource sharing”. *J Supercomput, Vol.* 45, pp 185–204, 2008.

[12] R. Buyya and M. Murshed. “GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing”. *Concurrency and Computation: Practice and Experience*, Vol. 14, pp. 13-15, 2002.

[13] A. Legrand, L. Marchal, and H. Casanova. “Scheduling distributed applications: the SimGrid simulation framework”. *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 138-145, 2003.

[14] C. L. Dumitrescu and I. Foster. “GangSim: a simulator for grid scheduling studies”. *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, Vol. 2, pp. 1151-1158, 2005.

[15] R. N. Calheiros, R. Ranjan, A. Beloglazov, et al. “CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms”. *Cloud Computing and Distributed Systems (CLOUDS) Laboratory Department of Computer Science and Software Engineering The University of Melbourne, Australia.*

[16] CodeBlue project, http://www.eecs.harvard.edu/ mdw/proj/codeblue/. (Date last accessed May 6th, 2011).

[17] CloudSim, http://www.cloudbus.org/cloudsim/. (Date last accessed October 18th, 2011)

VITA

Sravya Nutalapati

Candidate for the Degree of

Master of Science

Thesis: A TASK RESOURCE PROBABILISTIC SCHEDULING ALGORITHM

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December 2011.

Completed the requirements for the Bachelor of Technology in Information Technology at Acharya Nagarjuna University, Guntur, AP, India in 2009.

Experience:

Graduate Research Assistant in the Department of Computer Science, Oklahoma State University, Stillwater, OK October'11 – December'11

Graduate Research Assistant in Oklahoma Animal Disease Diagnostic Lab, Oklahoma State University, Stillwater, OK August'10 - September'11

System Analyst in the Department of Entomology and Plant Pathology, Oklahoma State University, Stillwater, OK January'10 - July'10

Web Developer in the Department of Zoology, Oklahoma State University, Stillwater, OK January'10 - July'10

Name: Sravya Nutalapati Date of Degree: December, 2011

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: A TASK RESOURCE PROBABILISTIC SCHEDULING ALGORITHM

Pages in Study: 41 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study:

The future of mobile health monitoring combined with cloud computing is expected to see a lot of technological advances. In this thesis we assume that to remotely monitor a patient, biomedical sensors are attached to the patient's body. The signals from the sensors of different patients are sent over to a six filter system through the patient's cell phone. The filter is located between a geographical group of patients and the cloud. The filter will measure, analyze and process the sensor data such as temperature, blood pressure, heart beat, etc. If timely action is required, the data is sent to the hospital management system and a copy of the data will be stored in the cloud. The parameters which do not require a real-time response are sent over to the cloud.

Findings and Conclusions:

The proposed scheduling algorithm is validated using CloudSim, which is a tool kit for modeling and simulation of cloud computing environments. The scheduling algorithm, which runs on the filters, schedules the tasks arriving from different patients, based on the resources available in the cloud. The scheduling algorithm ensures that time constrained tasks get scheduled and executed before the deadline. The tasks which are not time constrained are scheduled on any one of the filters based on a probabilistic function. Simulation results of the scheduling algorithm show that with an increase in the average number of tasks at a filter, the percentage of tasks that will meet the deadline decrease. Furthermore, with an increase in the deadline of tasks, the percentage of tasks that will meet the deadline also increases. As non-emergency tasks do not have a time constraint, they will always find a filter to execute. Future work will seek to improve on the above approach in order to effectively schedule emergency situations and also identify approaches to place the filters in appropriate locations to optimize performance.

ADVISER'S APPROVAL: JOHNSON P THOMAS