University of Washington

Abstract

Energy-aware Workflow Scheduling and Fuzzy Logic Based Capacity Provisioning in Cloud Environment

Yao Pan

Chair of the Supervisory Committee:
Ph.D. Yan Bai
Institute of Technology

The increasing need of large-scale data centers has brought new challenges to the development of energy-efficiency techniques. Although many optimization techniques have been proposed, most of them neglect the characteristics of tasks and fail to consider the dependencies between tasks. Therefore, more comprehensive optimization is needed. The goal of this thesis is to combine capacity provisioning with scientific workflow scheduling to generate optimization in these two ways. Capacity provisioning is built upon a fuzzy logical mechanism to accomplish constructing computing resources proactively, and workflow scheduling is designed with the purpose of achieving energy efficiency. We aim at achieving optimization both in energy consumption and completion time.

Energy-aware Workflow Scheduling and Fuzzy Logic Based Capacity Provisioning in Cloud Environment

Yao Pan

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science

University of Washington

2015

Committee:

Yan Bai, Chair

Donald Chinn

Program Authorized to Offer Degree:
Computer Science and Systems

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would never have been able to finish my thesis without the guidance of my committee members and help from my family.

I would like to express my deepest gratitude to my advisor, Dr. Yan Bai, for her excellent guidance, patience, and providing me with an excellent atmosphere for doing research. I would like to thank Dr. Donald Chinn, who patiently corrected my technical errors and writing.

I would also like to thank my parents. They were always supporting me and encouraging me with their best wishes.

Chapter 1. INTRODUCTION

With the booming development of service applications and the popular concept of Cloud computing, large-scale virtualized data centers have been built by leading IT companies. Beloglazov and Buyya [1] observed that the rapid development in demand for computational power driven by modern service applications combined with the shift to the Cloud computing model have led to the establishment of large-scale virtualized data centers. Buyya et al. [2] further observed that the Cloud computing model leverages virtualization of computing resources allowing customers to provision resources on-demand on a pay-as-you-go basis. By using the Cloud, customers can outsource their computational jobs and will not need to deal with the maintenance and upgrades of both hardware and software, which can also save a lot of money. At the same time, on the Cloud side, more workload will be added to the data centers. This would result in a significant increase in server side energy consumption. Beloglazov et al. [3] pointed out these data centers consume enormous amounts of electrical energy resulting in high operating costs and high-level carbon dioxide emissions. Furthermore, high temperatures caused by energy consumption lead to some serious problems, such as reduced system reliability and availability, as well as decreased lifetime of devices [4]. All of these indicated that the development of energy-efficient management methodologies and techniques are needed.

Although many research achievements have improved energy efficiency, most of them assumed tasks are independent of each other and neglected the dependent characteristics between them. Currently, many scientific applications, such as Nimbus [5] and Eucalyptus [6], are under the trend of executing in a Cloud computing environment instead of using Grid platform. These

scientific applications are modeled as workflows, which can be abstracted into a Directed Acyclic Graph (DAG). The dependency and parallelism embedded in a workflow requires suitable scheduling algorithms, which can achieve high-level energy saving performance. Under this motivation, this research chooses an approach of workflow scheduling that generates a user-satisfied service quality defined by completion time and simultaneously supplies high-level energy efficiency in the Cloud environment.

Virtual machine (VM) consolidation is another approach that is used to save energy in Cloud environment. VM consolidation occurs when the system places two or more VMs located in different physical servers into one physical server. In order to determine when and how to execute a consolidation, the Cloud environment needs a capacity assessment system. However, in a real-world scenario, computing resource requirements exhibit uncertainty and imprecision that suggests that the capacity assessment system cannot use traditional binary sets where variables only take on true or false values. As a form of many-valued logic, fuzzy logic has been extended to handle the concept of partial truth. Fuzzy logic variables may have truth-values ranging between completely true and completely false rather than fixed and exact [7]. Based on these characteristics, we implemented a fuzzy logic-based capacity assessment system to meet a high level Quality of Service (QoS), and meanwhile, try to save energy for the Cloud environment.

This thesis provides four main contributions: 1) It will consider the dependency between tasks when implementing energy-aware scheduling; 2) extend the performance metrics and add energy consumption when estimating the performance of workflows scheduling algorithms; 3) make comparison on energy consumption and completion time between our implemented

algorithms and the popular Grid one, Heterogeneous Earliest Finish Time (HEFT) [8], in a Cloud environment; and 4) combine Fuzzy-based capacity provisioning with workflow scheduling to generate a dual-direction optimization.

The rest of the thesis is organized as follows. Chapter 2 discusses related work. Chapter 3 explains the details of the scheduling system model, which include the application model, the Cloud model, and the energy model. The MinMin [9], MaxMin [9], and HEFT [8] algorithms are described in Chapter 4. The details of the simulator and the experimental setup, which used to compare the algorithms, are explained in Chapter 5. Chapter 6 proposes a Fuzzy-based capacity assessment system for the Cloud environment. Chapter 7 describes how to use the Fuzzy system to determine capacity provisioning. The final section discusses lessons learned and future work.

Chapter 2. RELATED WORK

Previous work has addressed energy-aware task scheduling in the Cloud environment. Goiri et al. [10] proposed a power-aware task scheduling policy in a virtualized data center. The policy takes many important parameters into consideration, such as reliability or dynamic Service Level Agreements (SLAs) enforcement, and others, but they modeled all tasks as independent and neglected the dependency and control flow between tasks. Therefore, there is potential to take dependencies of scientific workflows into account to develop a better algorithm.

Recently, some research considered the benefits of using Cloud computing for executing scientific workflows. Hoffa et al. [11] compared the performance of running the Montage workflow [12] in a local cluster, against a science Cloud at the University of Chicago. Similarly, Juve et al. [13] did the comparation between in the NCSA's Abe cluster and the Amazon EC2.

Both of them used Pegasus [14] as the workflow management system to execute the workflows. However, these works concentrated only on the feasibility of running scientific workflows in the Cloud environment, and the performance of completion time between these two environments. They did not consider energy-efficiency performance during their research.

Abrishami et al. [15] proposed two algorithms, which they called IaaS Cloud Partial Critical Paths (IC-PCP) and IaaS Cloud Partial Critical Paths with Deadline Distribution (IC-PCPD2). These two algorithms have a polynomial time complexity, which makes them suitable options for scheduling large workflows. However, they aimed to minimize the economic cost of running scientific workflows in a Cloud environment, instead of the energy consumption of the data center.

Cao et al. [16] considered the energy consumption when made scheduling for workflows in a Cloud environment. By using Dynamic Voltage and Frequency Scheduling (DVFS) technique, they proposed a three-step workflow scheduling algorithm, namely Energy-efficient Resource Allocation for workflow Scheduling under Deadline constraint (ERAS-D). This algorithm calculates the optimal CPU frequency of a Virtual Machine (VM) when executing an assigned task and maximizes the resource utilization rate of a server in order to improve the system throughput and reduce the energy cost. However, only optimizing the VM's CPU frequency cannot guarantee the server is under an optimal energy consuming status.

Another category of work focuses on energy-aware dynamic capacity provisioning, which is one of the most promising solutions to reduce the energy cost. Based on look-ahead control, Kusic et al. [17] have proposed a dynamic resource-provisioning framework for a Cloud environment. By estimating the CPU share and workload directed to every VM, the framework

could adjust the number of physical and virtual machines in order to minimize power consumption. However, they mainly focus on the performance of servers rather than the scheduling at the VM level.

As a form of many-valued logic, fuzzy logic has been extended to handle the concept of partial truth. Compared to traditional binary sets where variables only take on true or false values, fuzzy logic variables may have truth-values ranging between completely true and completely false rather than fixed and exact [7]. In a real-world scenario, computing resource requirements exhibit uncertainty and imprecision, traits that can be modeled through use of fuzzy logic. Recently, fuzzy logic has been applied in Cloud computing. Albano et al. [18] categorized the response time and the residual capacity to adjust the CPU CAP (a real number, usually taking values in the range between 0 and 100) to apply to the VM in order to satisfy the reference response time. Lama et al. [19] categorized CPU usage limits as inputs and measured the performance of each application and the average power consumption of the Cloud capacity as outputs. They dynamically modulate the provisioned capacity with respect to the static provision method. Although they tried to provide guarantees on the Quality of Services (QoS) and, at the same time, to minimize the energy consumption, they only took limited inputs or neglected VM consolidation.

To address the issues within current workflow scheduling algorithms, we chose an energy-efficient workflow-scheduling algorithm, which can achieve optimization both in energy consumption and completion time. For the capacity provisioning part, we apply fuzzy logic to our capacity assessment system.

Chapter 3. SCHEDULING SYSTEM MODEL

The proposed scheduling system model consists of the following models:

3.1 THE APPLICATION MODEL

An application is considered to be a distributed computing application, which consists of multiple dependent tasks with data transfer and control flow between each other. Thus, according to this characteristic, we model an application as a DAG $w(T, E)$, where T is a set of n tasks $\{t_1, t_2, t_3, \ldots, t_n\}$, and E is a set of dependences. The weight $w_{i,j}$ on edge $e_{i,j}$ denotes the size of data transferred from t_i to t_j. A task cannot start its processing until all the required data from its preceding tasks have been received. In a given DAG, a task without any parent is called an entry task, and a task without any child is called an exit task. According to the properties of different applications, a DAG can have several entry tasks and several exit tasks.

3.2 THE CLOUD MODEL

Our Cloud model consists of a service provider, which provides services to its clients and is capable of executing applications (e.g. scientific workflows), and an energy monitor, which is responsible for energy-related calculating and controlling.

The service provider offers several VMs, VMs = $\{vm_1, vm_2, vm_3, \ldots, vm_n\}$, with QoS for each task of every workflow. $SVM(t_i)$ denotes the selected VM for task t_i. In general, there are many QoS parameters, like CPU, memory, bandwidth, cost, reliability, security and so on. In this thesis, we consider the most important and related one, CPU, for our scheduling model. We assume that each workflow task, t_i, can be processed by vm_i services with different QoS

parameters. Furthermore, We assume that all tasks use a shared storage service, such as Amazon Simple Storage Service [20], to send and receive the intermediate data. So the average bandwidth between VMs and the shared storage service is roughly equal. With this assumption, the data transfer time of a dependency $e_{i,j}$ only depends on $w_{i,j}$, and it is independent of the VMs that execute them. Therefore, we define $TT(e_{i,j})$ as the data transfer time of a dependency $e_{i,j}$, which is independent of the selected VMs for t_i and t_j. Also, we define $ET(t_i, vm_i)$ as the execution time of processing task t_i on service vm_i.

In order to optimize the energy consumption, the energy monitor component is added to our model. The energy monitor is responsible for server status conversion (e.g., turn on, sleep, and shut down) and VM migration. To achieve this purpose, the energy monitor periodically calculates and updates the CPU Utilization Rate (UR_{CPU}), which is one important attribute of our energy-aware scheduling algorithm.

3.3 The Energy Models

Most power consumption of a Cloud service center is composed of computational processing, disk storage access, communication using intra-network, and cooling systems. In this thesis, we only consider the power consumption by physical servers, which is mostly determined by the CPU, memory, disk storage and network. According to the conclusion from previous work [4], CPU utilization is typically proportional to the overall system load and power consumption. Therefore, we only consider the CPU power consumption for servers and use the power model defined as:

$$P(u) = kP_{max} + (1-k)P_{max}u \tag{3.1}$$

where P_{max} is the maximum power consumed when a server's CPU is fully loaded, k is the fraction of power consumed when a server is in its idle state (the value of k depends on the characteristics of a physical machine), and u is the percentage of CPU utilization.

Equation 3.1 can be further reduced to:

$$P(u) = \alpha + \beta u \tag{3.2}$$

where P is the power consumption of a server at the time point when its CPU utilization is u, α is the static part of the power consumption (i.e., when the server is idle), and β stands for the dynamic part of the power consumption, which is the difference when CPU utilization is between 0 and 100%.

Based on the power model defined above, we can calculate the energy consumption function over a period of time t.

$$E_i = \int_0^t P\big(u(t)\big)dt \tag{3.3}$$

where E_i denotes the energy consumption of server i in a continuous period of time.

To formulate the total data center energy consumption E, we define an equation:

$$E = \sum_{i=1}^{N} \int_0^t P\left(\sum_{j=1}^{M} u_{VM_{j,i}}(t)\right) dt \tag{3.4}$$

where N denotes the total number of running servers in the data center, M denotes the total number of VMs in the i^{th} server, and $VM_{j,i}$ denotes the j^{th} VM in the server i.

Chapter 4. ALGORITHM IMPLEMENTATION

In this section, we introduce the main idea of the algorithms, then describe some basic definitions, and finally elaborate on the MinMin [9], MaxMin [9] and Heterogeneous Earliest Finish Time (HEFT) [8] algorithms.

4.1 MAIN IDEA

Greedy algorithms are widely used in workflow scheduling. Most of these algorithms follow the problem solving heuristic of making the locally optimal choice at each stage [21], in order to minimize the completion time of the entire workflow. We implemented the MinMin and MaxMin algorithms. But instead of only testing the completion time, we aim to test the energy consumption of executing the workflow within an acceptable completion time.

4.2 BASIC DEFINITIONS

For a workflow, a schedule is defined as assigning services to its tasks. For each task, t_i, of an arbitrary schedule, we define its Earliest Start Time, $EST(t_i)$ as the earliest time when t_i can start its computation. It can be computed as follows:

$$EST(t_{entry}) = 0 \tag{4.1}$$

$$EST(t_i) = max_{t_p \in predecessors\ of\ t_i}\{EST(t_p) + ET\left(t_p, SVM(t_p)\right) + TT(e_{p,i})\} \tag{4.2}$$

Based on the definition above, we can further get the Earliest Finish Time of each task t_i, $EFT(t_i)$, which is the earliest time when t_i can finish its computation. It can be computed as follows:

$$EFT(t_i) = EST(t_i) + ET(t_i, SVM(t_i)) \tag{4.3}$$

In consideration of Cloud computing environment changing computing resources and energy consumption quite frequently, our chose algorithms are based on the idea of making local decisions about which task to send to which resource.

4.3 THE MINMIN SCHEDULING ALGORITHM

Originally, MinMin [9] was proposed to minimize the completion time of an application in a Grid environment. We use to test the completion time and energy consumption in our Cloud environment. Algorithm 1 shows the pseudo-code of our implementation. In the first phase, the algorithm updates the ready tasks pool, which tasks' predecessors finished. We define all tasks that in the initial state are unscheduled. In the next phase, the algorithm determines available VMs for each unscheduled task in ready tasks pool. Then it calculates the minimum *EFT* value over all ready tasks. After that, a task with the minimum *EFT* value is scheduled and marked as scheduled. This will be repeated until all the tasks have been scheduled. The intuition of MinMin is to create a local optimal path so as to reduce the overall completion time.

Algorithm 1: ***MinMin scheduling algorithm***

1. **while** *w(T, E)* are not finished **do**
2. Update *readyTasks* = tasks with predecessors finished;
3. *MinMin*(*readyTasks*);

procedure *MinMin*(*readyTasks*)

4. **while** *readyTasks* are not all scheduled **do**
5. determine available VMs for each unscheduled task in *readyTasks;*
6. compute $EFT(t_i)$ for each unscheduled task in *readyTasks;*
7. compute $min(EFT(t_i))$ over all tasks in *readyTasks*;
8. schedule t_i
9. mark t_i as scheduled
10. **end procedure**

4.4 THE MAXMIN SCHEDULING ALGORITHM

The pseudo-code for MaxMin [9] is shown in Algorithm 2, which has a similar idea with MinMin [9]. The only difference with MinMin is that the task with the maximum EFT value is selected for execution. The intuition of MaxMin is to avoid penalty from long running tasks.

Algorithm 2: ***MaxMin scheduling algorithm***

1. **while** *w(T, E)* are not finished **do**
2. Update *readyTasks* = tasks with predecessors finished;
3. *MaxMin(readyTasks*);

procedure *MaxMin*(*readyTasks*)

4. **while** *readyTasks* are not all scheduled **do**
5. determine available VM for each unscheduled task in *readyTasks;*
6. compute $EFT(t_i)$ for each unscheduled task in *readyTasks;*
7. compute $max(EFT(t_i))$ over all tasks in *readyTasks*;
8. schedule t_i
9. mark t_i as scheduled
10. **end procedure**

4.5 THE HEFT SCHEDULING ALGORITHM

HEFT [8] is a popular scheduling algorithm and has been used in the Grid environment. There are three phases for this algorithm: 1) Weighting: based on the average Earliest Finish Time (Average EFT) and average data Transfer Time (Average TT), assign weights to the tasks and dependencies respectively. 2) Ranking: performed traverse the workflow graph upwards and assign a rank value each of the tasks. The rank of a task is the weight of the task plus the maximum over each successor of the successor's rank plus the weight on the edge between the task and that successor. 3) Mapping: by decreasing order of rank value, schedule each task to the VM that has the Earliest Finish Time. The capacity of the Grid environment is relatively "static" when comparing with the Cloud environment. With the trend of executing scientific workflow in the Cloud environment, we aim to test the performance of the traditional scheduling algorithm HEFT, which is popular in Grid environment, in the Cloud environment. We choose this algorithm as the baseline when we evaluating the performance of the algorithms.

Chapter 5. FUZZY-BASED CAPACITY ASSESSMENT FOR CLOUD ENVIRONMENT

As we discussed in Chapter 1, VM consolidation is another major approach for energy saving in Cloud environment. In order to determine when and how to execute a consolidation, the Cloud environment needs a capacity assessment system. This assessment system responsible for determining whether there is an excess or lack of computing resources. Also, it should be able to determine when to trigger a consolidation. However, in a real-world scenario, computing resource requirements exhibit uncertainty and imprecision, which means the capacity assessment system is likely to be not optimal if it uses traditional binary sets where variables only take on true or false values. As a form of many-valued logic, fuzzy logic has been extended to handle the concept of partial truth. Fuzzy logic variables may have truth-values ranging between completely true and completely false rather than fixed and exact [7]. Based on these characteristics, we implemented a fuzzy logic based capacity assessment system to meet a high level Quality of Service (QoS), and meanwhile, try to save energy for the Cloud environment.

To construct a Fuzzy-based capacity assessment model, four major steps are involved. The first step specifies key capacity indicators and defines linguistic variables. We begin by determining problem input and output variables and their ranges in this step. For our problem, there are three key capacity indicators defined as input: (1) the average waiting time for each task. When a task is ready to execute, there is no guarantee that the task can execute immediately. Then the waiting time can reflect the situation of available capacity for our workflows; (2) the residual capacity of VMs, which is in idle status. Although the residual capacity is a good indicator, it cannot exclusively determine whether the available capacity is an

excess or lack of resources. Considering the dependency character of the workflows, we need to combine the two indicators above to determine the capacity situation. Then according to the inputs (1) and (2), the assessment system generates the first output is the VM adjust value, which is how many VMs should change status between power-on and power-off (positive adjust value means turning VMs off); (3) the physical servers' average CPU utilization. This indicator reflects the CPU utilization status of the Cloud environment. But the available capacity of the Cloud environment is a dynamic state, which means we cannot use (3) exclusively to determine whether to trigger VM consolidation. Thus we combine (3) with the first output, the VM adjust value, as the two inputs to generate the second output: the suitability to trigger a VM consolidation.

As mentioned earlier, the inputs and the outputs constitute vague estimates rather than crisp values; such vague estimates defined general categories, which is a range value as opposed to fixed collections. Valid ranges of the inputs are considered and divided into classes, or fuzzy sets. For example, the average waiting time can range from "low" to "high." The residual capacity can also range from "low" to "high," which is same with the physical servers' average CPU utilization. The first output is the VM adjust value and is defined in fuzzy sets in five categories: Much Lack, Lack, Fit, Excess, and Much Excess. The second output is the suitability to trigger a VM consolidation and is defined in fuzzy sets in four categories: Cannot Trigger, Not Suitable, Suitable, and Must Trigger. These categories have more flexible membership requirements that allow for partial membership to a category. The degree to which a value is a member of a category can be any value between 0 and 1. In fuzzy logic, these categories are called fuzzy sets. We cannot specify clear boundaries between classes. The degree of

belongingness of the values of the variables to any selected classes is called the degree of membership [22]. Based on the simulation results of the three scheduling algorithms, we create the input and output variables' ranges for our assessment model. According to the length of the workflows and the average completion time, we create the boundaries for the variable: the average waiting time. According to the whole capacity of our simulation environment and the average usage, we create the boundaries for the variables: the residual capacity and the VM adjust value. Table 5.1 lists the details of the categorizations.

Table 5.1. Input and Output Variables and Their Range

Input variable: the average waiting time, *w (seconds)*		
Value	Notation	Range ()
Low	L	0, 200
Fit	F	100, 400
High	H	300, 600
Input variable: the residual capacity, *r (VM)*		
Value	Notation	Range ()
Low	L	0, 20
Fit	F	10, 50
High	H	40, 80
Input variable: the average CPU utilization, *u (%)*		
Value	Notation	Range ()
Low	L	0, 50
Medium	M	40, 80
High	H	70, 100
Input/output variable: the VM adjust value, *ΔC (VM)*		
Value	Notation	Range ()
Much Lack	MLK	-50, -20
Lack	LAK	-30, -10
Fit	FIT	-10, 10
Excess	EXS	10, 30
Much Excess	MES	20, 50
Output variable: the suitability, *s (%)*		
Value	Notation	Range ()
Cannot Trigger	CNT	0, 20
Not Suitable	NST	20, 50
Suitable	STB	40, 80
Must	MST	60, 100

The second step determines fuzzy sets. Each fuzzy set has a corresponding membership function that returns the degree of membership for a given value within a fuzzy set. Fuzzy sets can have a variety of shapes. In our system, it is appropriate to use a triangle shape, because it can often provide an adequate representation of the expert knowledge, and at the same time significantly simplifies the process of computation [23]. Figures 5.1-5.5 show how we can represent the inputs and outputs by means of membership functions.

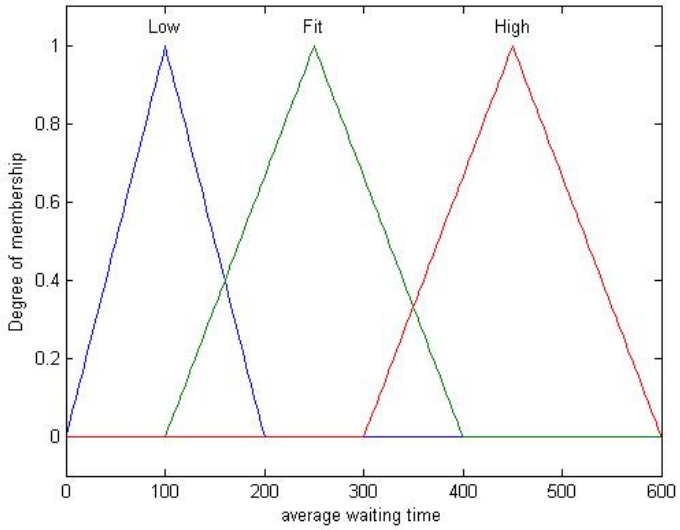


Figure 5.1. Fuzzy Sets of Average Waiting Time

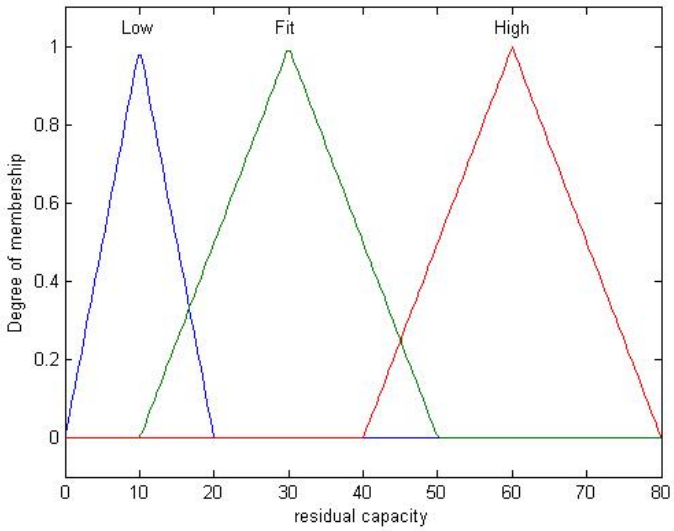


Figure 5.2. Fuzzy Sets of Residual Capacity

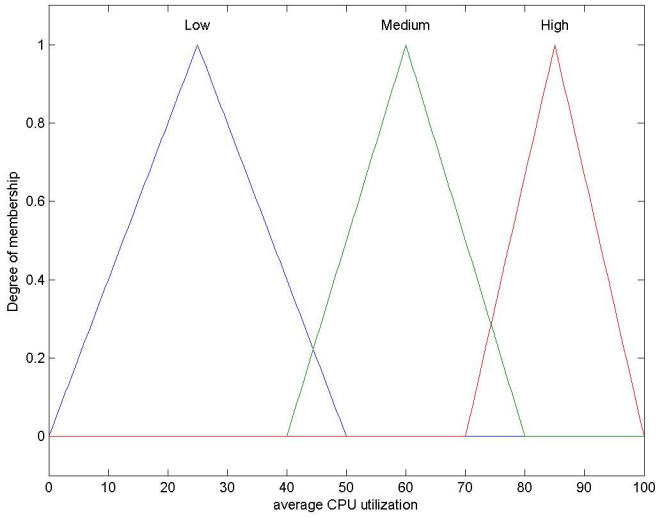


Figure 5.3. Fuzzy Sets of Average CPU Utilization

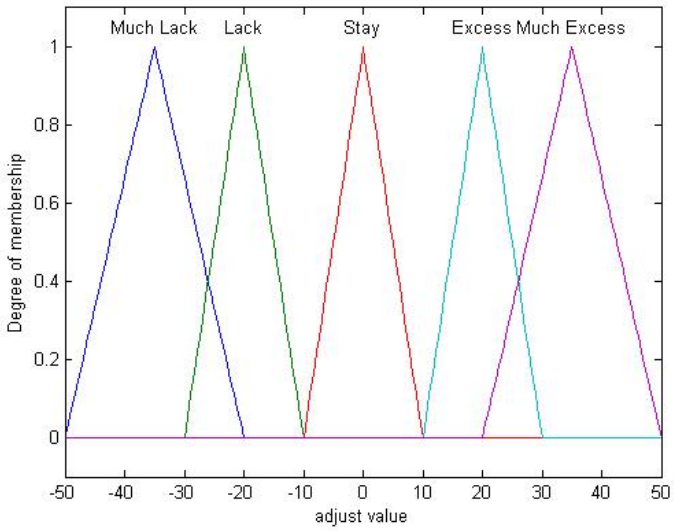


Figure 5.4. Fuzzy Sets of VM Adjust Value

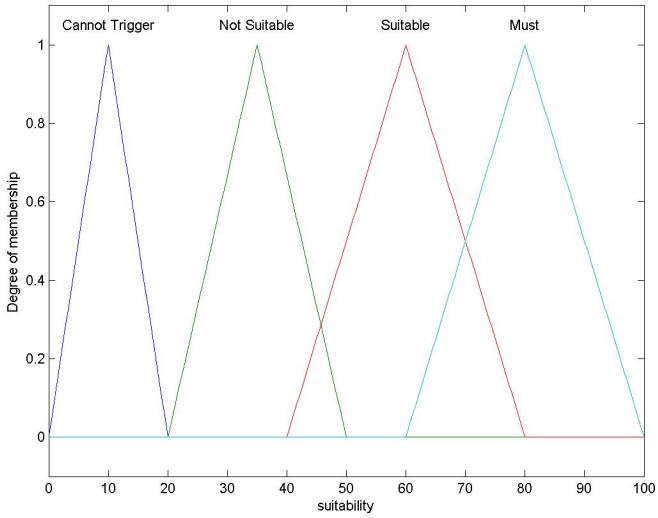


Figure 5.5. Fuzzy Sets of Suitability

Step 3 specifies fuzzy rules. Having specified the two outputs value and their indicators, the next step is to specify how the outputs values vary as a function of the factors. Experts provide fuzzy rules that relate the output values to various levels of indicators based on their knowledge and experience. In our system, there are three inputs and two outputs variables. For each output

variable, there are two input variables associated with it. Based on experts' knowledge, we carried out several experiments to devise a set of rules. Tables 5.2 and 5.3 contain these rules.

Table 5.2. The Rules for the Adjust Value

ΔC		w		
		Low	Fit	High
r	Low	STY	LAK	MLK
	Fit	EXS	STY	LAK
	High	MES	EXS	EXS

Table 5.3. The Rules for the Suitability

s		u		
		Low	Medium	High
ΔC	Much Lack	NST	CNT	CNT
	Lack	STB	NST	CNT
	Stay	MST	STB	NST
	Excess	MST	MST	STB
	Much Excess	MST	MST	STB

Lastly, we encode the fuzzy model and tune the system. Probably this is the most laborious step to evaluate and tune the system to let it meet the requirements specified at the beginning. To build our fuzzy expert system, we use Matlab Fuzzy Logic Toolkit [24]. It provides a systematic framework for computing with fuzzy rules and graphical user interfaces. Based on the rules in Table 5.2 and 5.3, we programmed our assessment system by using the Fuzzy Logic Toolbox. Then it generates a surface to help us analyze the system's performance. Figure 5.6 and 5.7 represent the three-dimensional plots of the system.

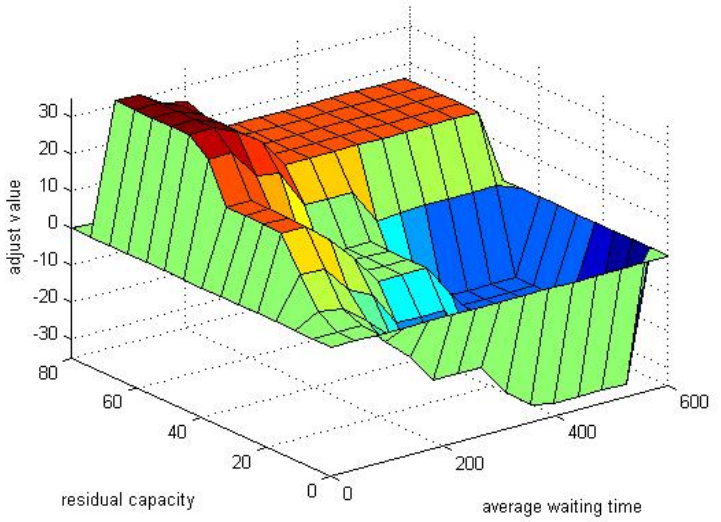


Figure 5.6. Three-dimensional First Output Surface

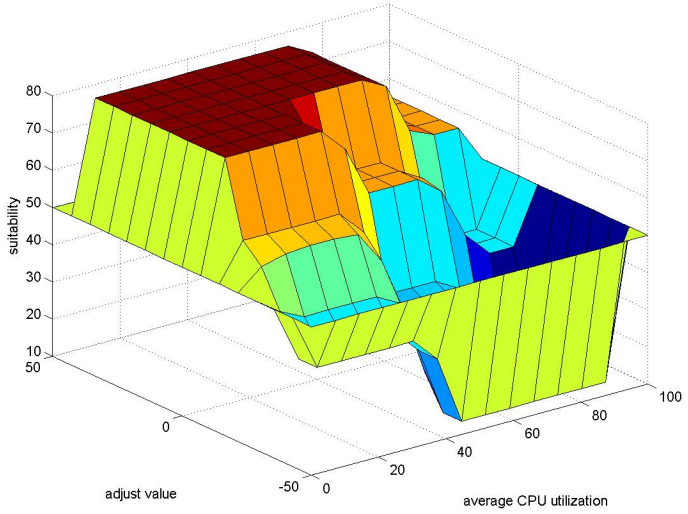


Figure 5.7. Three-dimensional Second Output Surface

Chapter 6. USING THE FUZZY SYSTEM TO DETERMINE CAPACITY PROVISIONING

The use of fuzzy model developed in the Chapter 5 is to determine the VM adjust value and the suitability to trigger a consolidation in Cloud environment of four main steps: Fuzzification, Rule Evaluation, Aggregation, and Defuzzification.

Step 1: Fuzzification: The first step is to take the crisp input, w_1, r_1, u_1 and determine the degree to which these inputs belong to each appropriate fuzzy set. For example, the crisp input w_1 (average waiting time, 200) corresponds to the membership functions W_1 and W_2 (Low, Fit). The crisp input r_1 (residual capacity, 10) maps the membership functions R_1 and R_2 (Low, Fit). The crisp input u_1 (average CPU utilization, 40) corresponds to the membership functions U_1 and U_2 (Low, Medium). In this manner, each input is fuzzified over all the membership functions used by the fuzzy rules.

Step 2: Rule evaluation: The second step is to take the fuzzified input w_1 (Low, Fit), r_1 (Low, Fit), and u_1 (Low, Medium) and apply the following fuzzy rules to the inputs. These rules are derived from those specified in Tables 5.2 and 5.3.

Rule 1: If w_1 is Low and r_1 is Low the offset value is Stay.
Rule 2: If w_1 is Low and r_1 is Fit then offset value is Excess.
Rule 3: If w_1 is Low and r_1 is High then offset value is Much Excess.
Rule 4: If w_1 is Fit and r_1 is Low then offset value is Lack.
Rule 5: If w_1 is Fit and r_1 is Fit then offset value is Stay.
Rule 6: If w_1 is Fit and r_1 is High then offset value is Excess.
Rule 7: If w_1 is High and r_1 is Low then offset value is Much Lack.
Rule 8: If w_1 is High and r_1 is Fit then offset value is Lack.
Rule 9: If w_1 is High and r_1 is High then offset value is Excess.
Rule 10: If u_1 is Low and ΔC is Much Lack then suitability is Not Suitable.
Rule 11: If u_1 is Low and ΔC is Lack then is Suitable.
Rule 12: If u_1 is Low and ΔC is Stay then is Must.
Rule 13: If u_1 is Low and ΔC is Excess then is Must.

Rule 14: If u_1 is Low and ΔC is Much Excess then is Must.
Rule 15: If u_1 is Medium and ΔC is Much Lack then suitability is Cannot Trigger.
Rule 16: If u_1 is Medium and ΔC is Lack then suitability is Not Suitable.
Rule 17: If u_1 is Medium and ΔC is Stay then suitability is Suitable.
Rule 18: If u_1 is Medium and ΔC is Excess then suitability is Must.
Rule 19: If u_1 is Medium and ΔC is Much Excess then suitability is Must.
Rule 20: If u_1 is High and ΔC is Much Lack then suitability is Cannot Trigger.
Rule 21: If u_1 is High and ΔC is Lack then suitability is Cannot Trigger.
Rule 22: If u_1 is High and ΔC is Stay then suitability is Not Suitable.
Rule 23: If u_1 is High and ΔC is Excess then suitability is Suitable.
Rule 24: If u_1 is High and ΔC is Much Excess then suitability is Suitable.

Step 3: Aggregation of the rule outputs: Aggregation is the process of unification of the outputs of all rules. In other words, we take the membership functions of all rule consequents previously clipped or scaled and combine them into a single fuzzy set. Thus, the input of the aggregation process is the list of clipped or scaled consequent membership functions, and the output is one fuzzy set for each output variable.

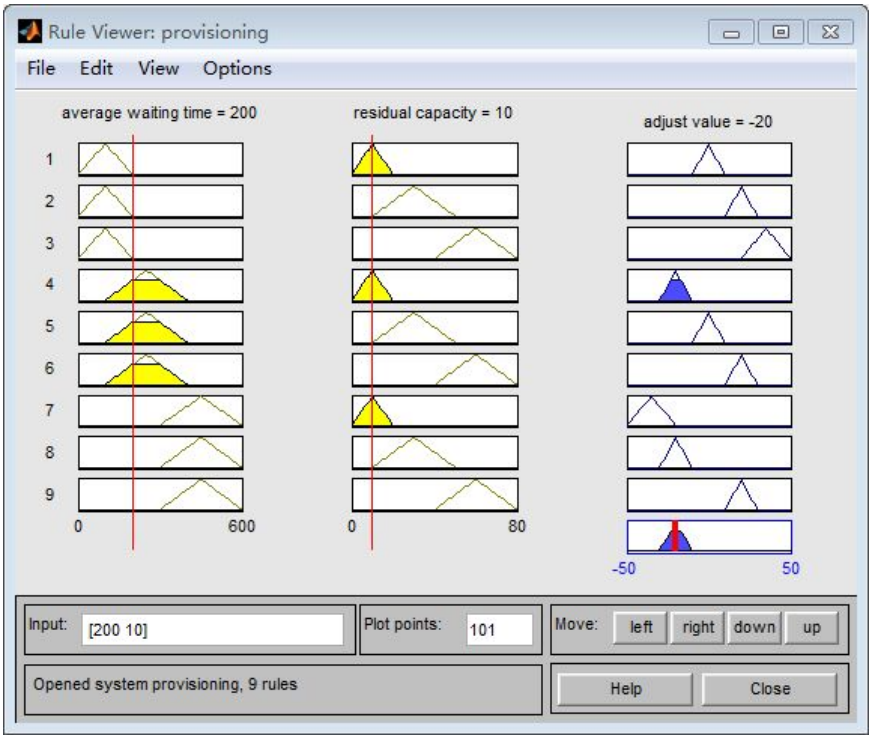


Figure 6.1. Aggregation and Defuzzification Process of the First Output

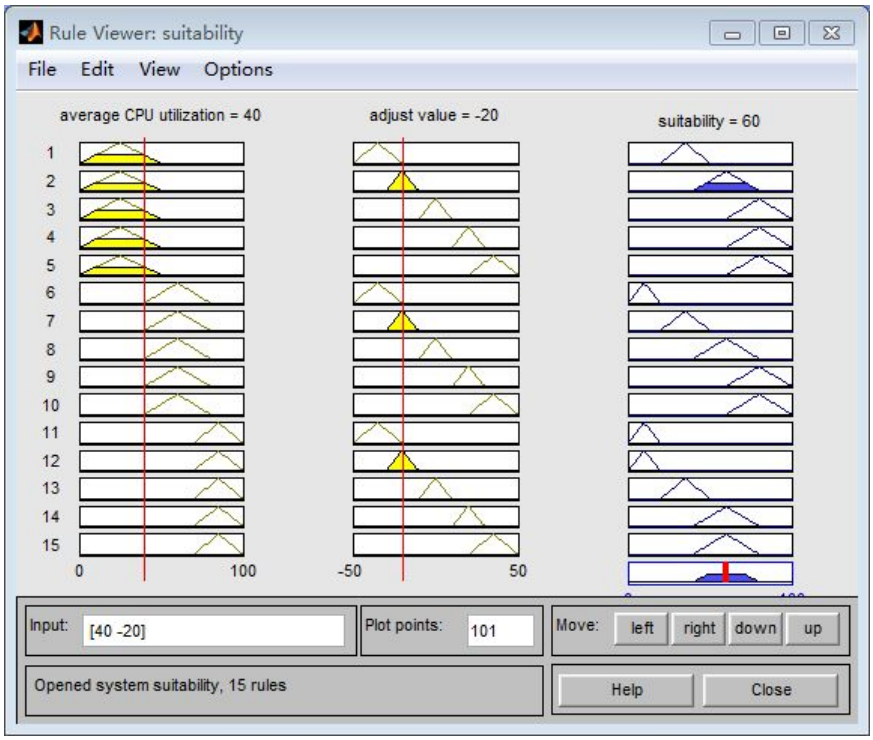


Figure 6.2. Aggregation and Defuzzification Process of the Second Output

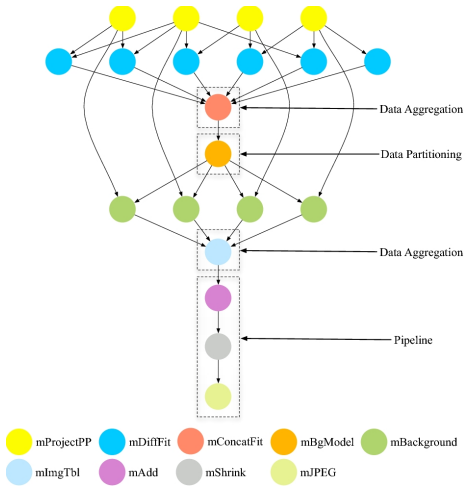
Step 4: Defuzzification: The final output of a fuzzy system has to be a crisp number. The input for the defuzzification process is the aggregate output fuzzy set and the output is a single number. We adopt the most popular method, centroid technique to defuzzifize the output. For example, Figure 6.1 shows the crisp output is -20. It means for instance, that the adjust value involved in our system is -20, which falls into range [-30, -10]. The adjust value determines that the current capacity is in a “Lack” state, which means the Cloud environment should provide more capacity. Figure 6.2 shows the crisp output is 60. It means for instance, that the suitability involved in our system is 60%, which falls into the range [40, 80]. The suitability determines that it is “Suitable” to trigger a VM consolidation.

Chapter 7. PERFORMANCE EVALUATION

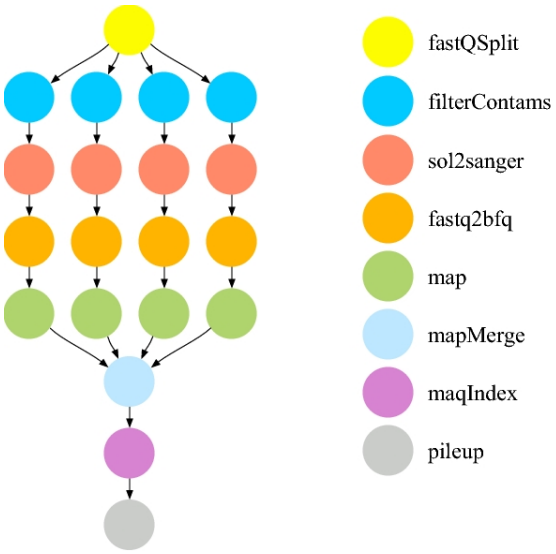
In this section, we will present our simulations of the three algorithms discussed in Chapter 4. Then we will make a comparison after implementing the fuzzy-based assessment system.

7.1 EXPERIMENTAL SETUP

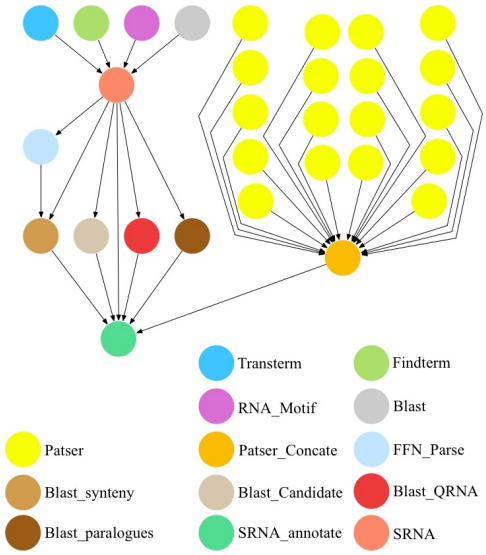
A Java-based simulator, WorkflowSim [25], is used and extended to simulate the Cloud environment for our experiments. The simulator consists of a service provider, which provides services to execute each task type of the five experimental workflows. To evaluate the completion time and energy performance of the algorithms discussed in Chapter 4, we have used a library of realistic workflows that are introduced by Bharathi et al. [12] as our test data. Figure 7.1 shows the approximate structure of a small instance of each workflow. From these five experimental workflows, we have chosen four sizes, Small (about 30 tasks), Medium (about 50 tasks), Large (about 100 tasks), and Extra-large (about 1000 tasks) for our algorithm. Finally, the MinMin and MaxMin algorithms are compared with the HEFT algorithm.



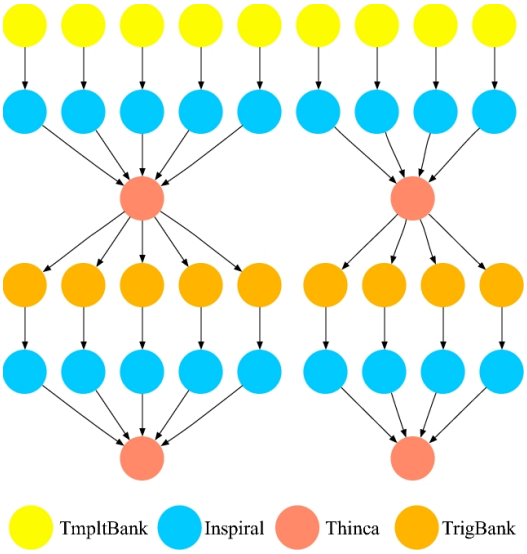
(a) Montage.



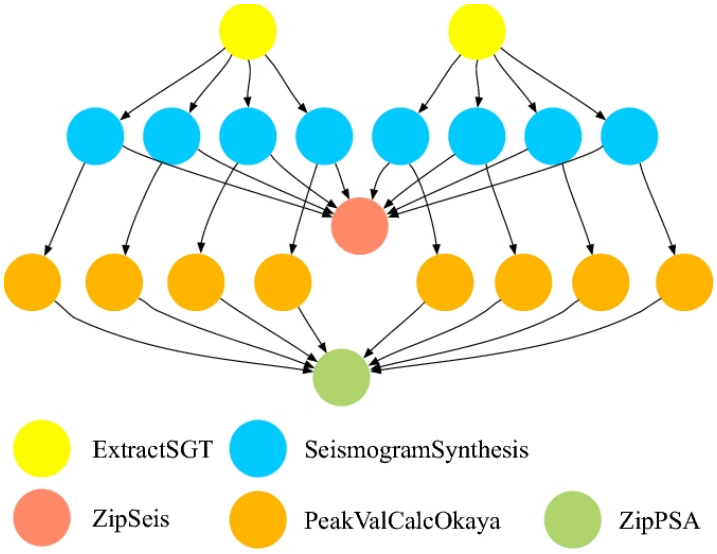
(b) Epigenomics.



(c) SIPHT.



(d) LIGO.



(e) CyberShake.

Figure 7.1. The structure of five realistic scientific workflows

7.2 EXPERIMENTAL RESULTS

Since a large set of workflows with different attributes is tested, it is important to normalize the total energy consumption of each workflow execution. Figure 7.2 shows the completion time and energy saving rate of scheduling large workflows. We use the actual completion time (seconds) as our metric for completion time. As for the metric for energy saving, we choose algorithm HEFT as our baseline. We calculate the energy saving rate of different algorithms based on this baseline. Then, we make comparisons between without and with the fuzzy-based assessment system for time saving rate and energy saving rate.

Montage: This workflow is the only one in which all algorithms have almost similar result for all size workflows. After implementing the assessment system, the completion time stay same when workflow size range from Small to Large. When workflow size come to Extra-large, the time saving rate reached 10.57% due to more VMs have been launched. As for the energy saving rate, after implementing the assessment system, it decreases from 31.56% to -24.65% along with the workflow size range from Small to Extra-large. The reason for this performance is that larger workflow size means higher CPU utilization rate and thus less VM consolidations. When workflow size reaches Extra-large, more VMs have been launched in order to shorten the completion time. Thus the energy saving rate reached -24.65%, which means more energy has been used in order to make a balance between energy saving and completion time.

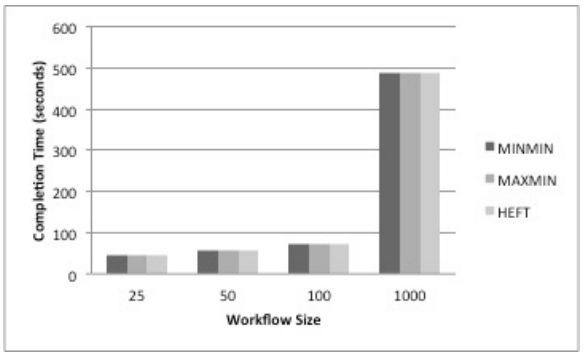
Epigenomics: This workflow has the best performance for the energy saving rate, especially in the Extra-large instance, with 59.14% (Algorithm MinMin) and 59.27% (Algorithm MaxMin) energy saving rate over algorithm HEFT. The performance of the algorithm is improved when the workflow gets larger. Meanwhile, the performance for the completion time is also attractive.

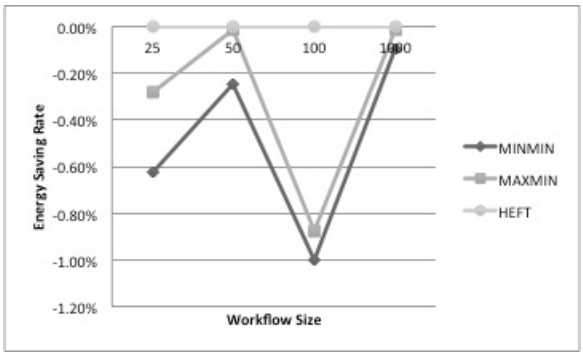
Especially for the Extra-large workflow size, the algorithm HEFT has especially long completion time comparing with MinMin, MaxMin. Because the HEFT algorithm schedules tasks according to tasks' rank value and does not consider the VMs' current status, several VMs could have long waiting list while others idle or waiting. This reason can explain why HEFT algorithm has such long completion time and much energy cost. The performance after implementing the assessment system is similar with Montage.

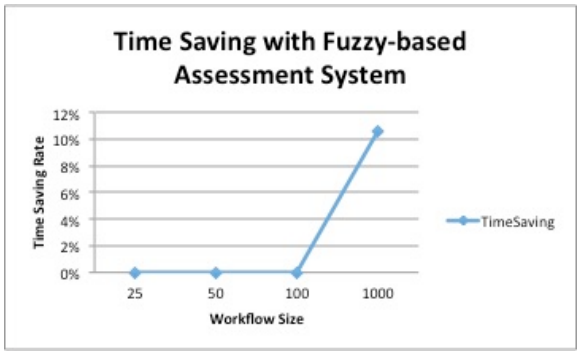
SIPHT: In this workflow, algorithm MinMin has bad performance when the workflow gets larger. When workflow size reaches Large, MinMin has the worst performance, which is -58.36% energy saving rate. After that, the performance gets better and ends with -16.41% for Extra-large size. Algorithm MaxMin has the same performance with baseline for Small, Medium, and Large instances. But the results get slightly worse (-2.99%) when size reaches Extra-large. SIPHT workflow has several bottleneck tasks, which have long executing times and have many successors. Thus, MinMin algorithm suffers long running tasks penalty.

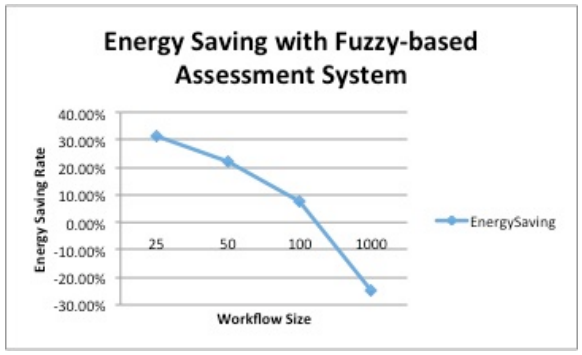
LIGO: MinMin and MaxMin have a little bit worse than baseline for Small, Medium, and Large instances. However, they still have the best performance for Extra-large instance with 39.86% and 39.89% energy saving rate respectively. MinMin and MaxMin can guarantee good load balance, which leads to a good energy saving performance when workflow size reaches Extra-large.

CyberShake: In this workflow, MaxMin is better than MinMin. Similar to the SIPHT workflow, the CyberShake workflow also has one bottleneck (ZipSeis task). MaxMin is able to avoid bottleneck penalty, thus gets better performance than MinMin. But CyberShake only has one bottleneck, so the advantage is not as obvious as SIPHT.

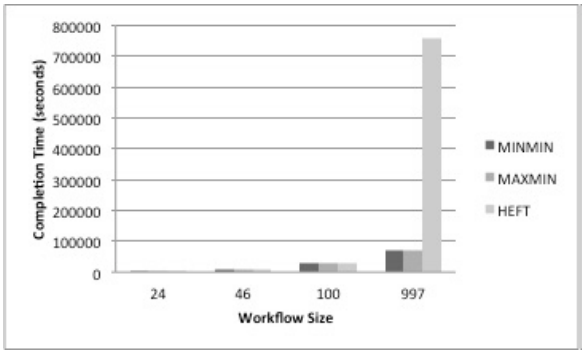


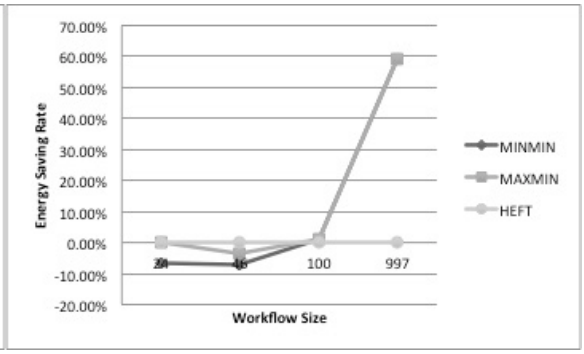


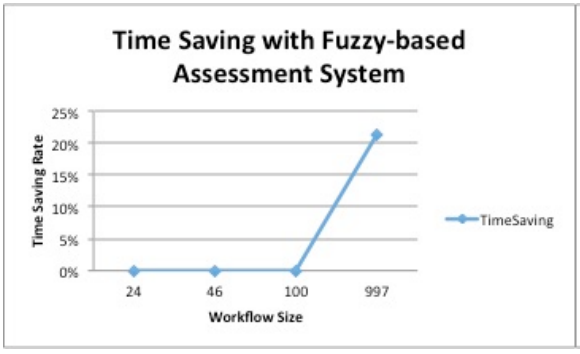


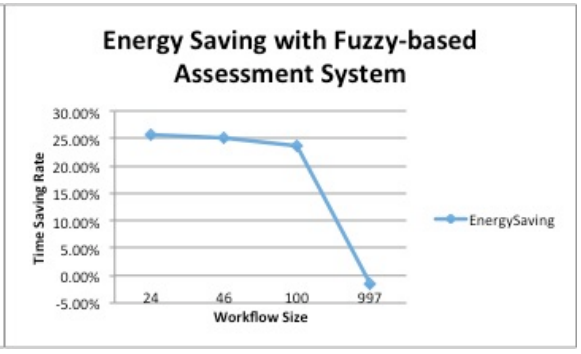


(a) Montage

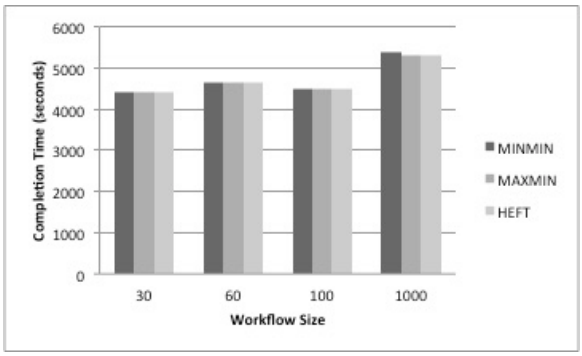


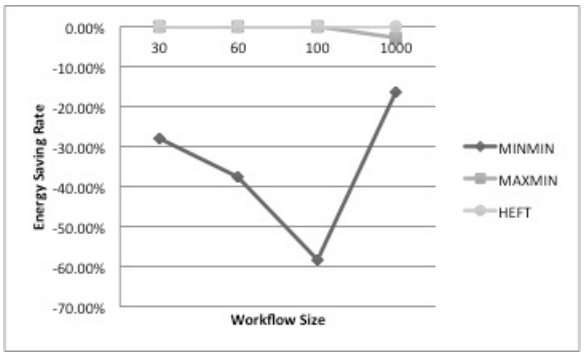


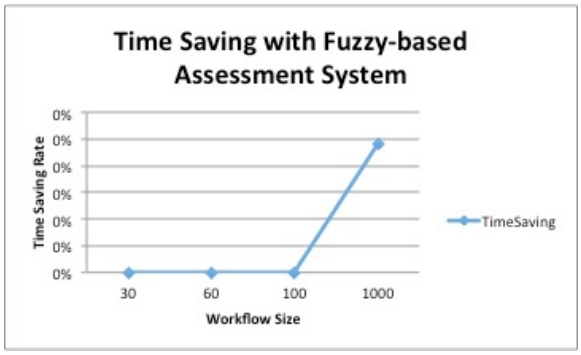


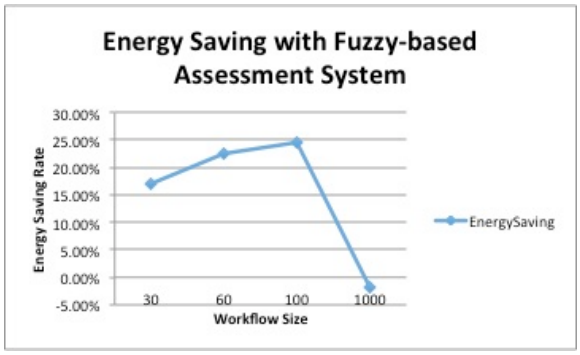


(b) Epigenomics

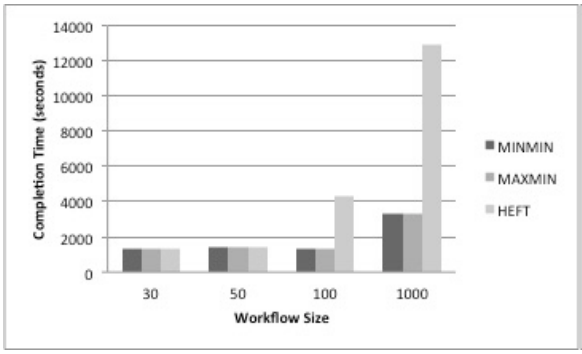


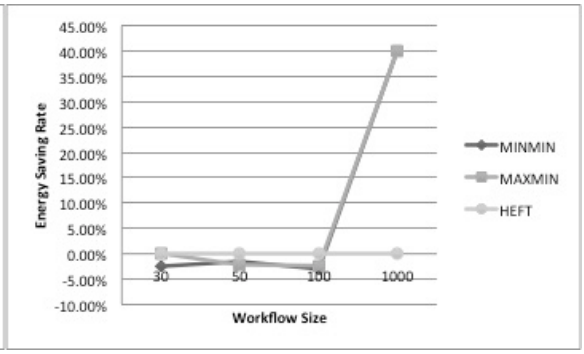


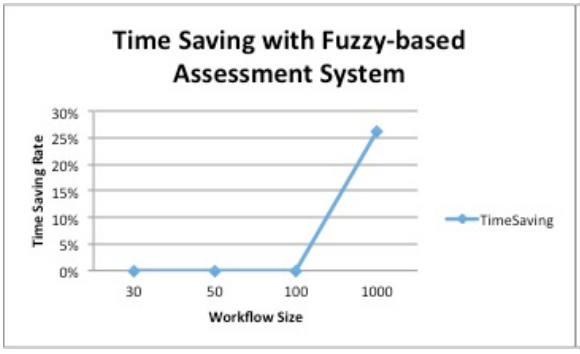


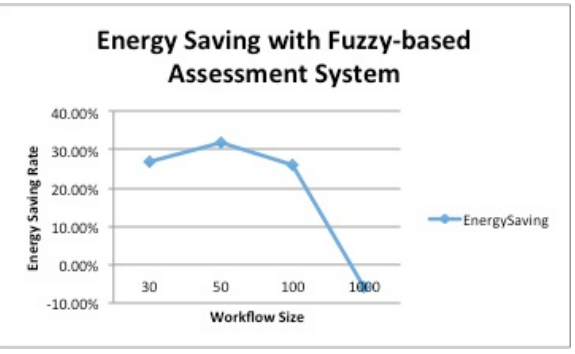


(c) SIPHT

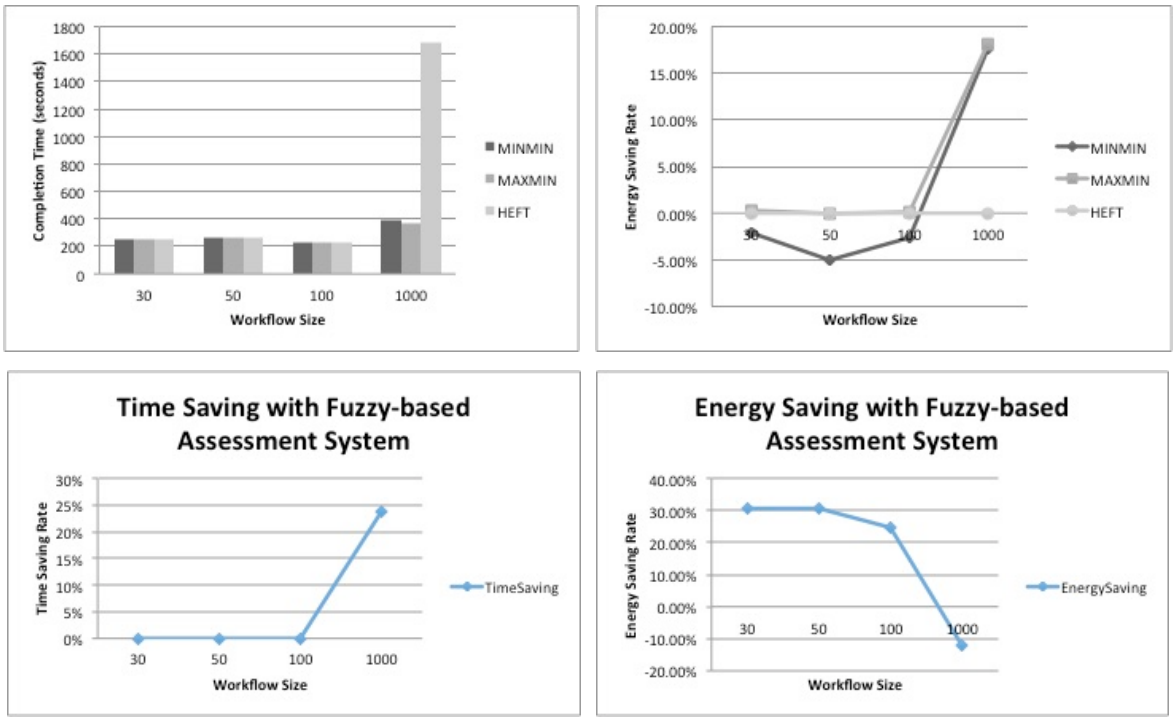








(d) LIGO



(e) CyberShake

Figure 7.2. Completion time and energy saving rate of scheduling workflows

Chapter 8. CONCLUSION AND FUTURE WORK

The Cloud computing environment enables users to obtain their required services without maintaining their own infrastructure, which brings much convenience and money saving. In this thesis, we implemented three algorithms, MinMin, MaxMin and HEFT for academic workflow scheduling in Cloud environment, which minimizes the energy consumption while getting an acceptable completion time. We evaluated the algorithms by simulating them with workflows abstracted from real scientific workflows with different structures and sizes. The results show that the algorithms had good performance for most instances especially for Extra-large instances. Furthermore, the experiments show that the completion time of the MinMin and MaxMin algorithms is also shorter in some instances than the baseline HEFT algorithm. After

implementing the fuzzy-based assessment system, we got good energy saving performance while within an acceptable completion time. However, restricted by the workflow, we set up the simulation environment as a homogeneous Cloud environment. Also, we neglect the overhead of VM adjusting and consolidation. These are the limitations of our current research. In the future, we plan to extend the algorithms to consider other computing resources, e.g. memory; a heterogeneous Cloud environment; and the overhead of VM adjusting and consolidation.

BIBLIOGRAPHY

[1] Anton Beloglazov and Rajkumar Buyya. Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers. Concurrency and Computation: Practice and Experience 24, 2012, 1397.

[2] R. Buyya R, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. Future Generartion Computer Systems 25, no. 6, 2009, 599-616.

[3] A. Beloglazov and R. Buyya. Adaptive Threshold-based Approach for Energy- efficient Consolidation of Virtual Machines in Cloud Data Centers. Proceedings of the 8th International Workshop on Middleware for Grids (MGC), Clouds and e- Science, 29 November – 3 December 2010.

[4] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing. Future Generation Computer Systems, 2011, 755-768.

[5] Nimbus Home Page. http://www.nimbusproject.org/.

[6] Eucalyptus Home Page. http://www.eucalyptus.com/.

[7] Fuzzy logic, URL: en.wikipedia.org/wiki/Fuzzy_logic

[8] Topcuoglu, Haluk; Hariri, Salim Wu, M. "Performance-effective and low-complexity task scheduling for heterogeneous computing". IEEE Transactions on Parallel and Distributed Systems 13 (3), p. 260–274, 2002.

[9] Etminani. K, and Naghibzadeh. M, “A Min-min Max-min Selective Algorithm for Grid Task Scheduling,” The Third IEEE/IFIP International Conference on Internet, Uzbekistan, 2007.

[10] I. Goiri, F. Julia, R. Nou, J. L. Berral, J. Guitart, and J. Torres. Energy-aware Scheduling in Virtualized Datacenters. 2010 IEEE International Conference on Cluster Computing (CLUSTER), 20-24 Sept. 2010, 58-67.

[11] Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B. and Good, J. “ On the use of cloud computing for scientific workflows”, Fourth IEEE Int’l Conference on e-Science (e-Science 2008), Indiana, USA, pp. 640 – 645 (2008).

[12] Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.H. and Vahi, K. “Characterization of scientific workflows”, The 3rd Workshop on Workflows in Support of Large Scale Science, Austin, TX, USA, pp. 1 – 10 (2008).

[13] Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B.P. and Maechling, P. “Scientific workflow applications on Amazon EC2”, 5th IEEE International Conference on e-Science, Oxford, UK (2009).

[14] E. Deelman, et al. Pegasus: a framework for mapping complex scientific workflows onto distributed systems Sci. Program., 13 (2005), pp. 219 – 237.

[15] Saeid Abrishami , Mahmoud Naghibzadeh , Dick H. J. Epema, Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds, Future Generation Computer Systems, v.29 n.1, p.158-169, January, 2013.

[16] Fei Cao, Michelle M. Zhu, Energy-aware Workflow job scheduling for green Clouds, GreenCom-iThings-CPSCom, 2013, pp. 232-239.

[17] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and Performance Management of Virtualized Computing Environments via Lookahead Control. International Conference on Autonomic Computing (ICAC), 2008, 3-12.

[18] L. Albano, C. Anglano, M. Canonico, and M. Guazzone. Fuzzy-Q&E: Achieving Quality of Service Guarantees and Energy Savings for Cloud Applications with Fuzzy Control. IEEE 3rd International Conference on Cloud and Green Computing, 2013, 159-166.

[19] P. Jain and I. Chlamtac. PERFUME: Power and Performance Guarantee with Fuzzy MIMO Control in Virtualized Servers. 2011 IEEE 19th International Workshop on Quality of Service (IWQoS) on: 6-7 June 2011, 1-9.

[20] Amazon simple storage service (amazon s3). http://aws.amazon.com/s3/.

[21] Introduction to Algorithms (Cormen, Leiserson, and Rivest), Chapter 17 "Greedy Algorithms" p. 329, 1990.

[22] W.-H. Au and K.C.C. Chan, Classification with Degree of Membership: A Fuzzy Approach, in: Proc. Of the 1stIEEE Int’l Conf. on Data Mining, San Jose, CA, 2001, p. 35-42.

[23] M. Negnevitsky, “Artificial Intelligence. A Guide to Intelligent Systems”, Addison-Wesley 2002.

[24] Matlab Fuzzy Logic Toolkit, URL: http://www.mathworks.com/products/fuzzy-logic/.

[25] Weiwei Chen , Ewa Deelman, WorkflowSim: A toolkit for simulating scientific workflows in distributed environments, Proceedings of the 2012 IEEE 8th International Conference on E-Science (e-Science), p.1-8, October 08-12, 2012.