Autonomic System for Optimal Resource Management in Cloud Environments

Fahimeh Ramezani

Ph.D. Thesis
This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

University of Technology Sydney
Faculty of Engineering and Information Technology
January 2016

Certificate of Original Authorship

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Student:

Date: 26/07/2016

Dedication

To my love Mohsen for his support, passion and patience and to my beloved parents for their encouragement that let my dreams come true.

And

To my dear son Sina who came to this World after I finished my PhD as the best gift from God to make my life more amazing, enjoyable and meaningful.

ACKNOWLEDGEMENT

I would like to express my appreciation to Professor Jie Lu, who supervised this Ph.D. program, for all of her knowledgeable comments, precious support, right guidance at hard times, and great suggestions along the way. I want to thank Jie for her wonderful scientific and spiritual support during this journey, and her willingness that let my research follow my interests. I am also grateful to Associate Professor Javid Taheri, my co-supervisor, for his knowledgeable suggestions and valuable technical advice through my PhD study. I would also like to express my special thanks to Professor Albert Zomaya for his generous support and letting me grow up beside his amazing research team. I also thank Dr. Farookh Khader Hussain, my co-supervisor, for his supports.

Looking back at my Ph.D., I see my husband, Mohsen, truly shoulder to shoulder with me during this journey. Thank you very much for being so supportive in all circumstances throughout the four years of this Ph.D., for understanding the stress I was subject to, for having sacrificed your time for me while you were doing your Ph.D., and for giving me the freedom to follow my scientific interests unconditionally. I could not have accomplished this without your constant love and support.

I would like to thank my parents, the first ones who taught me, for their encouragement and support despite the geographical distance. Pursuing my PhD was not possible without their love and assistance. I express my gratitude to all my friends and colleagues in the Decision Systems & e-Service Intelligence (DeSI) Laboratory for their help and valuable comments during my study.

I also appreciate the Faculty of Engineering and Information Technology and the Centre for Quantum Computation and Intelligent Systems (QCIS) at the University of Technology Sydney (UTS) for conference registration and travel funds provided during this research. This research was also supported by the International Research Scholarship (IRS) and UTS President's Scholarship (UTSP) funded by UTS.

Last, but absolutely not least, a special thank you goes to Ms. Sue Felix, Ms. Barbara Munday, and Mr. John Hazelton for helping me to identify and correct grammar and syntax problems in my publications.

ABSTRACT

Cloud computing is a large-scale distributed computing paradigm driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet. Considering the lack of resources in cloud environments and fluctuating customer demands, cloud providers require to balance their resource load and utilization, and automatically allocate scarce resources to the services in an optimal way to deliver high performance physical and virtual resources and meet Service Level Agreement (SLA) criteria while minimizing their cost.

This study proposes an Autonomic System for Optimal Resource Management (AS-ORM) that addresses three main topics of resource management in the cloud environment including: (1) resource estimation, (2) resource discovery and selection, and (3) resource allocation. A fuzzy Workload Prediction (WP) sub-system and a Multi-Objective Task Scheduling optimization (MOTS) sub-system are developed to cover the first two aforementioned topics. The WP sub-systems estimates Virtual Machines' (VMs') workload and resource utilization, and predicts Physical Machines' (PMs) hotspots. The MOTS sub-system determines the optimal pattern to schedule tasks over VMs considering task transfer time, task execution cost/time, the length of the task queue of VMs and power consumption.

To optimize the third topic in resource management, resource allocation, VM migration that is the current solution for optimizing physical resources allocation to VMs and load balancing among PMs, is investigated in this study. VM migration has been applied to system load balancing in cloud environments by memory transfer, suspend/resume migration, or live migration for the purpose of minimizing VM downtime and maximizing resource utilization. However, the migration process is both time- and cost-consuming as it requires large size files or memory pages to be transferred, and consumes a huge amount of power and memory for the origin and destination PMs especially for storage VM migration.

This process also leads to VM downtime or slowdown. To deal with these shortcomings, a Fuzzy Predictable Task-based System Load Balancing (FP-TBSLB) sub-system is developed that avoids VM migration and achieves system load balancing by transferring extra workload from a poorly performing VM to other compatible VMs with more capacity. To reduce the time factor even more and optimize load balancing over a cloud cluster, FP-TBSLB sub-system applies WP sub-system to not only predict the performance of VMs, but also determine a set of appropriate VMs that have the potential to execute the extra workload imposed on the poorly performing VMs. In addition, FP-TBSLB sub-system employs the MOTS sub-system to migrate the extra workload of poorly performing VMs to the compatible VMs.

The AS-ORM system is evaluated using a VMware-vSphere based private cloud environment with VMware ESXi hypervisor. The evaluation results show the benefit of the AS-ORM in reducing the time taken for the load balancing process compared to traditional approaches. The application of this system has the added advantage that the VMs will not be slowed down during the migration process. The system also achieves significant reduction in memory usage, execution time, job makespan and power consumption. Therefore, the AS-ORM dramatically increases VM performance and reduces service response time. The AS-ORM can be applied in the hypervisor layer to optimize resource management and load balancing which boosts the Quality of Service (QoS) expected by cloud customers.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms and Procedures

Chapter 1.

INTRODUCTION

1.1 BACKGROUND

Cloud computing provides new business opportunities for both service providers and their clients, by means of an architecture for delivering Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) on-demand over the Internet that promises to introduce significant economic and technical benefits (Celesti et al. 2012). These large scale services can be provided by applying shared virtualized cloud resources. In general, the cloud resources are provided as a collection of several proprietary processes in a virtual environment, called a virtual machine (VM).

Virtualized computational resources are applied in a cloud environment to provision resources on demand. Virtualization also provides the opportunity of using an auto-scaling technique that dynamically allocates computational resources to the services to match their current loads precisely, thereby removing resources that would otherwise remain idle and cost (Dougherty, White & Schmidt 2012). Advances in virtualization techniques and the construction of numerous large commodity data centers around the world have resulted in a new approach to computing referred to as cloud computing becoming an important topic of research and development (Ghanbari et al. 2012). The recent surge in the popularity and

usage of cloud computing services by both enterprise and individual consumers has necessitated the efficient and proactive management of data center resources that host services with a variety of characteristics.

Autonomic computing refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users. Started by IBM in 2001, this initiative ultimately aims to develop computer systems capable of self-management, to overcome the rapidly growing complexity of computing systems management, and to reduce the barrier that complexity poses to further growth. An autonomic system makes decisions on its own, using high-level policies; it will constantly check and optimize its status and automatically adapt itself to changing conditions (Wikipedia 2011). Self-adapting schedules need to meet changes in existing service requirements and self-configuring components to satisfy new service requirements. Hence, more autonomic and intelligent cluster Resource Management Systems (RMSs) are essential to effectively manage the limited supply of resources with dynamically changing service demand (Yeo 2008).

One of the major issues concerning both cloud service providers and consumers is real time autonomic resource management in response to highly unpredictable demands (Bashar 2013). There are two main process related to optimal autonomic resource management in cloud environments: (1) virtual resource discovery and selection to execute cloud services —or task scheduling over virtual resources— as a self-adapting schedule, and (2) physical resource allocation to virtual resources —or load balancing among PMs— to cover part of the self-configuring process.

In a cloud environment, generated tasks by cloud's service users are assigned to the virtual computational resources to be executed by applying an NP-hard problem which is called task scheduling. Task scheduling algorithms mainly seek to achieve optimal resource utilization, maximize throughput, minimize response time, and avoid overload.

Efficiently allocating scarce resources among competing interests in decentralized cloud computing networks is a challenging subject for resource managers. The unpredictable clouds' service/customer demands cause the workload of VMs to fluctuate dynamically, leading to imbalance in both the load and utilization of virtual and physical cloud resources. In light of these uncertainties, therefore, cloud providers need a decision support that will dynamically scale up and scale down resources, or migrate VM over PMs without violating Service Level Agreements (SLA) while simultaneously ensuring adequate revenue.

The adoption of cloud computing requires a detailed comparison of infrastructure alternatives, SLA criteria and objectives, the QoS criteria, and taking a number of aspects into careful consideration for optimal resource management. Optimization techniques allow the provider to determine resource allocations to various clients' demands in order to maximize its revenue while minimizing its costs based on aformentiedn criteria.

However, resource management optimization problems have been extensively studied, the majority of developed optimization systems focused on one aspect including: task scheduling (resource discovery and selection), load balancing (resources allocation), or predicting required resources (resource estimation). To the best of our knowledge, none of them applied their model to propose an autonomic system which optimizes both task scheduling and resource allocation based on their interactions and the predicted resource utilization. In addition, VM migration that is applied for load balancing in a cloud cluster is not an optimal way for large size VMs, because it leads the VM to slow down during the migration process, carries the risk of losing last customer activities, and is cost- and time-consuming.

Considering these facts, this study prepares a combination of the three main topics of resource management as a unique system and proposes an Autonomic System for Optimal Resource Management in Cloud Environments (AS-ORM). Furthermore, new techniques for predicting resource utilization, task scheduling and load balancing have been developed in this study to overcome the drawbacks of current solutions.

The rest of this chapter is organized as follows. The research description including the research problems, objectives and expected contributions are presented in Sections 1.2, 1.3 and 1.4 respectively. A description of the research methodology and plan is given in Section 1.5. Section 6 outlines the thesis structure. Lastly, the publications and awards of the thesis are listed in Section 1.7.

1.2 RESEARCH PROBLEMS

This section explains the main issues which significantly motivate this study and present the research questions:

(1) Resource management problems include, provisioning, requirement mapping, adaptation, brokering, modeling, estimation, discovery and selection, and allocation (Manvi & Krishna Shyam 2014). Although a significant amount of research has been done in the area of resource management such as (Ghanbari et al. 2012; Li et al. 2011; Li et al. 2012; Liao, Jin & Liu 2012; Song, Hassan & Huh 2010; Wei et al. 2010) more improvements are still needed as these approaches tried to cover one aspect of the resource management problem and the interaction between resource discovery and selection (job/task scheduling), and allocation (load balancing) has been neglected. As in the dynamic cloud environment the number of arrival tasks and the amount of required resources change frequently, task scheduling and resource allocation in such an environment should update frequently to provide optimal cloud utilization. In addition, task scheduling model depends on the amount of available physical and virtual resources, and this amount changes on the basis of task scheduling pattern. In conclusion, to achieve optimal solution for cloud utilization, both task scheduling and resource allocation should be taken into account at the same time in a unique optimization system.

(2) It is intuitive that if the dynamic resource scaling system is a reactive one, it might not be able to scale proportionally with the Slashdot effect (Halavais 2001) or sudden traffic surge resulting from special offers or market campaigns (Islam et al. 2012b); thus

turning out to be catastrophic for application performance, leading to an unacceptable delay in response time and in the worst case, application unavailability (Ghanbari et al. 2012). Therefore, proactive estimation of resource utilization is required for optimal resource management in order to cope with the ever fluctuating resource usage pattern of e-commerce services. Predicting resource usage and VM workload considering arrival tasks is the key to several crucial system design and deployment decisions related to resource management such as, workload management, system sizing, capacity planning and dynamic rule generation in the cloud.

(3) Task scheduling problems which are related to the efficiency of the whole cloud computing facilities, are of paramount importance. The task scheduling algorithms in distributed systems usually have the goals of spreading the load on processing nodes and maximizing their utilization while minimizing the total task execution time and makespan of their related jobs. In this study three main drawbacks of exiting task scheduling models are considered including: (1) the number and quality of objective functions, (2) improvement in the performance of applied evolutionary algorithms, (3) considering task priority to determine the optimal suggested solutions.

There are several studies (Guo et al. 2012; Juhnke et al. 2011; Lei et al. 2008; Li et al. 2011; Li et al. 2012; Salman, Ahmad & Al-Madani 2002; Song, Hassan & Huh 2010; Taheri et al. 2014; Tayal 2011) that mainly emphasize the minimization of job makespan, task execution cost, and/or task transfer time in their bi-objective optimization models. However, these studies fail to consider the need to minimize power consumption by the cloud infrastructure, and task queue length in a cluster by considering the workload capacity of VMs. Task queue length is an effective factor for reducing job makespan because it determines the wait and finish time of tasks.

Evolutionary algorithms are widely applied to solve multi-objective task scheduling optimization models that are NP-hard problems. The evolutionary algorithms initialize the first population by random and then start to optimize it. The randomly

determined population is usually not the optimal start point and it takes time for evolutionary algorithm to reach the Pareto optimal solutions from this point. Therefore, more improvement are required to determine optimal first population and start from the near best solution pattern. This will accelerate the performance of evolutionary algorithms to find the best solution faster.

In addition, to schedule a set of dependent task, their propriety should be considered as start time of a task depends on completion of its parent tasks, and requires their output data files. To direct evolutionary algorithms to follow task priorities when they determine their Pareto optimal solutions, new improvement is required to modify the optimal suggested solution patterns to consider task priorities and sequences.

(4) VM migration (live and storage migration) is applied by hypervisors (or Virtual Machine Monitor (VMM)) like VMware EXSi to manage cloud recourse and balance load over PMs (Clark et al. 2005; Jin et al. 2011; Jun & Xiaowei 2011; Liao, Jin & Liu 2012). Using this approach, a VM is migrated from an overloaded PM to another PM with available physical resources. VM migration is also applied to scale up VMs that are delivered as IaaS based on their customer demands, when the original host PM has no idle resources available (Sallam & Li 2014). Cloud providers benefit from VM migration for small size VMs as its entire process will be completed in seconds. However, VM migration for large size VMs results in dirty memory, utilizes a large amount of memory in the primary PM and destination PM, causes the VM to slow down during the migration process, and carries the risk of losing last customer activities. Therefore, in such cases, a need for a replacement solution has been recognized to achieve higher cloud utilization, increase the performance of poorly performing VMs, and as a result allow the cloud providers to deliver higher QoS with lower cost.

Based on the above mentioned issues, the research questions of this study are determined as follows:

- **Research Question 1**: How an autonomic system for optimal resource management in a cloud dynamic environment should be designed and what sub-systems should be included?
- **Research Question 2**: What are the requirements for such optimization system and how they can be achieved?
- **Research Question 3**: How to develop prediction models to estimate VMs' workload fluctuations and PMs' hotspots for facilitating proactive scaling in the cloud, so that hosted applications are able to tolerate the variation in workload with least drop in performance and availability?
- **Research Question 4**: How to design a task scheduling optimization algorithm that correctly converges to the minimum job makespan and optimal resource utilization based on the data arrival rate and computational needs?
- **Research Question 5**: What is a new solution for optimal load balancing over cloud physical resources to overcome the drawbacks of VM migration?
- **Research Question 6**: How to design a system prototype for the proposed model and how to implement it in a real cloud environment?
- **Research Question 7**: How could the performance of the proposed system be evaluated in a dynamic and complex cloud environment?

1.3 RESEARCH OBJECTIVES

This research has seven primary objectives based on the research problems, which are explained as follows as well as presented in Figure 1.1:

Research Objective 1. The first research objective corresponding to research question 1 is to design the Autonomic System for Optimal Resource Management (AS-ORM) that aims to optimize resource utilization in cloud environments. This system should have the ability to do self-adapting and self-configuring in terms of task scheduling and resource allocation respectively to be an applicable and reliable management system for dynamic and complex

environment of clouds. It is also essential for the AS-ORM to predict the resource utilization and customer upcoming demands and activities to accelerate decision making by this system. This predicted information will prevent the PMs becoming overloaded, and will prevent a surge in the number of low performance VMs. To design such a system, the following goals should be satisfied:

- Predicting VM workload fluctuations to determine poorly performing, under-loaded VMs and the host PMs hotspots(i.e. when a PM becomes overloaded and its resources are highly utilized)
- Predicting required physical resources (memory and CPUs)
- Optimizing task scheduling in cloud computing, based on flexible amount of arrival tasks and available resources
- Eliminate the drawbacks of current load balancing and resource allocation approach i.e. VM migration
- Optimize resource allocation based on determined task scheduling pattern (mainly applicable for SaaS and PaaS).
- Developing a software which is implemented based on the designed system.

This system will be applied in the hypervisor layer of cloud architecture and will lead to the development of an optimized intelligent elastic operations layer.

Research Objective 2. The second research objective is to determine the requirements of the proposed AS-ORM. This objective corresponds to research question 2. To identify the requirements of this system the following aspects should be considered and be clarified. This information is applied to maintain and complete each part of the proposed AS-ORM and satisfy its determined goals.

- QoS indicators like response time, cost, availability, reliability (Zhang & Xu 2012).
- System performance indicators in cloud environment including: latency, overhead, gap (job makespan or service response time), the number of processors.

- Pricing strategy in cloud computing that is defined by cloud service providers such as pay-per-use and reserved instance pricing models (Sadeka et al. 2012).
- Type and measurement components of service level agreement.

Research Objective 3. Corresponding to research question 3, the third research objective is to define abnormal situations and propose a practical prediction method to forecast resource utilization.

To do this, a fuzzy Workload Prediction (WP) sub-system is developed to forecast CPU usage and workload fluctuations for every VM in a cloud cluster. The WP determines the poorly performing VMs and their host PMs' hotspot. The WP results are also applied to determine a set of under-loaded VMs. Two WP models are developed for this sub-system including: the ES&NN-Based, and the DBN-Based workload prediction models.

The ES&NN-Based workload prediction model is designed by applying the combination of Neural Network (NN) and a developed fuzzy Expert System (ES). NN predicts CPU usage patterns of VMs using related historical data. ES monitors changes in CPU usage and workload for every VM. The results of the NN and the ES are then combined to determine the abnormal situations. The DBN-Based workload prediction model is purposed by applying Dynamic Bayesian Networks (DBNs). To design these prediction methods, the following steps have been conducted:

- Determine the conditions under which a VM might be overloaded in the near future
- Define appropriate variables to calculate resource utilization under each condition
- Determine related rules to each condition and its variables
- Design the NN, ES and DBN to apply by the prediction models
- Propose the NN&ES-WP and DBN-WP Algorithms

Research Objective 4. The forth research objective corresponding to research question 4 is to develop a comprehensive task scheduling optimization model by applying an

appropriate multi-objective evolutionary algorithm that considers fundamental aspects of optimizing the utilization of physical resources in cloud environments.

The majority of current researches in the area of developing an optimized task scheduling model for cloud and grid environments, considers at most two objective functions for their optimization model mainly task execution time and service cost.

To cover more criteria that are determined in SLA and achieve higher QoS with lower cost, four conflicting objective functions are considered in this study based on the WP results, VMs and host PMs properties (available CPU, memory, etc), tasks specifications and their required resources, and QoS indicators. These objective functions of the developed task scheduling model are: (1) task transfer time, (2) task execution cost/time, (3) length of VM task queue, and (4) power consumption.

In addition, two evolutionary algorithms including Multi-Objective Genetic Algorithm (MOGA) and Multi-Objective Particle Swarm Optimization (MOPSO) are applied and compared to determine the faster and more accurate algorithm to solve the proposed task scheduling optimization model.

Research Objective 5. The fifth research objective is to develop a novel load balancing model over cloud physical resources corresponding to research question 5. A fundamental shortcoming of most existing research to overcome overloaded PMs, poorly performing VMs, and enhance system load balancing is that they relied on complete VM migration. To improve previous approaches and reduce time consumption and cost in such situations, a Fuzzy Predictable Task Based System Load Balancing (FP-TBSLB) approach is proposed which transfer tasks from the task queue of the overloaded VMs instead of VM migration. In addition, to decrease power consumption, this model aims to choose VMs that are located on active PMs as destination VMs. To develop this approach the following steps have been conducted:

- Designing FP-TBSLB sub-system

- Predicting the time when a PM is overloaded or a set of VMs are poorly performing
- Developing a multi-objective optimization model for tasks migration from overloaded VMs to a set of compatible under-loaded VMs.

Research Objective 6. The sixth research objective follows on from research questions 1, 2, 3, 4 and 5 and aims to develop a system prototype based on the proposed models during this study. The prototype is developed based on the determined requirements, and solutions proposed in research objectives 1, 2, 3, 4, and 5. To demonstrate the performance of the AS-ORM sub-systems, three simulation cloud clusters are created based on various types of VMs and PMs with different amounts of allocated resources and properties. Several tasks are also created by different specifications and required resources to be executed. In addition, a private VMware ESXi based cloud cluster is created to implement AS-ORM prototype and evaluate its performance in comparison with current load balancing approaches that are applied for optimal resource management.

Research Objective 7. The seventh research objective aims to evaluate the developed AS-ORM. Evaluation is an important aspect of every methodology because it provides a reasonable amount of confidence in the results of the model. Four evaluation models and empirical environments are relied on in this research to evaluate the performance of four new systems that are developed in this study including:

- WP sub-system (ES&NN-WP and DBN-WP)
- MOTS sub-system
- FP-TBSLB sub-system
- AS-ORM

The performance of the DBN-WP sub-system is evaluated by implementing its prediction models in MATLAB and SMILE(Laboratory 1998). The MOTS-PSO/GA and TBSLB methods are implemented in simulation environments and evaluated by applying CloudSim (Calheiros et al. 2011). Finally, AS-ORM is evaluated by implementing its

subsystems (NN&ES-WP and FP-TBSLB) in a real VMware ESXi based private cloud and evaluated by applying the functionality of Condor(2015), MATLAB and CloudSim.

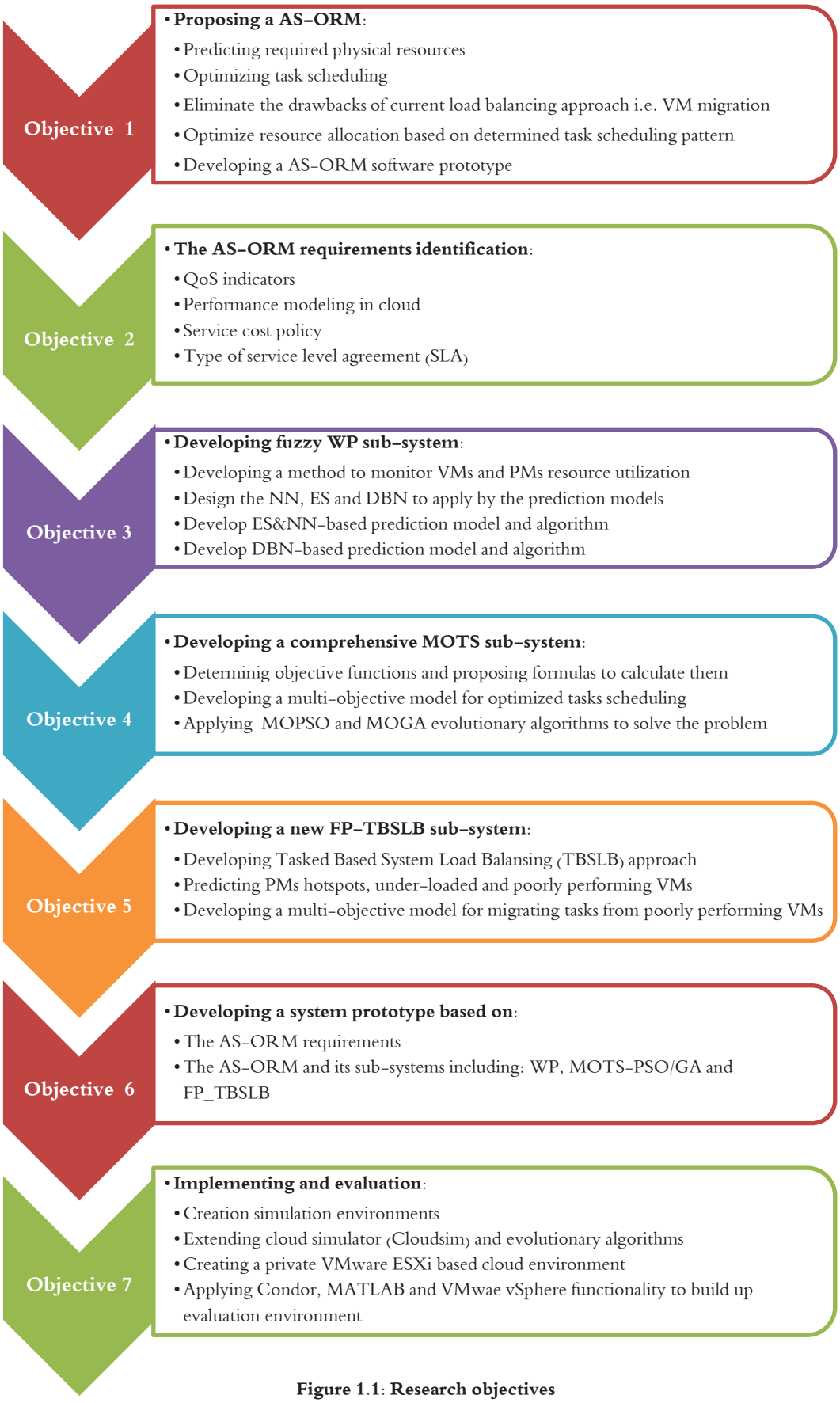


Figure 1.1: Research objectives

1.4 RESEARCH CONTRIBUTIONS

According to the research objectives, several research innovations and contributions of this study are summarized as follows:

1) **Develop an autonomic resource management system**: This research proposes a comprehensive autonomic system that covers three major topics in cloud resource management including: resource prediction, task scheduling, and load balancing based on their interactions. In cloud environments, resource allocation to VMs affect optimal task scheduling pattern. On the other hand, resource allocation should be optimized considering PMs and VMs available capacity and fluctuating workloads that directly determined based on task scheduling schema. Customers' demands and usage behavior also cause workload fluctuation over physical and virtual resources in a cloud cluster and create uncertain environment. Therefore, this research considers both task scheduling and resource allocation optimization based on predicted workload to enhance optimal cloud utilization, maximum QoS for customers and benefits for cloud providers.

   This system is designed to support cloud providers for general or particular resource management. It means that the system has the ability to select all the objects of the multi-objective problem (general) or select a set of objects according to the customer preferences (particular), before running optimization algorithms.

2) **Develop prediction models for VMs' workload and PM hotspots**: The existing researches in the area of VM workload prediction forecasts the future workload of VMs and their CPU usage by applying their previous workload patterns on the basis of related historical data (Yang et al. 2013). However, VM workload is affected by user behaviors and their sudden decisions, so fluctuations in the workload could be independent of the previous workload and CPU usage pattern. Considering these facts, this study proposed two workload prediction models that not only apply historical data, but also control near future changes in CPU usage and workload for

every VM. These models are able to determine the poorly performing VMs and under-loaded VMs, and consequently predict the PM hotspots.

3) **Develop a multi-objective task scheduling optimization model**: Literature review shows, in previous task scheduling optimization models, some but not all of the following objectives are considered:

- Minimizing task execution time
- Minimizing tasks transferring time
- Minimizing cost
- Minimizing energy consumption
- Minimizing number of task in the waiting lists

All of these goals are covered and considered by the proposed Multi-Objective Task Scheduling (MOTS) model to achieve the optimal resource utilization considering customer and providers points of view. In addition, this model is flexible about the number and weigh of the determined objectives according to customer or provider's preferences.

4) **Develop a novel approach for preventing system overload**: This study achieves system load balancing by migrating tasks form overloaded VMs instead of VM migration which is called FP-TBSLB approach. As VM migration utilizes much more time, resource and cost in comparison with tasks migration, the proposed approach significantly reduces power consumption, memory involved, service response time and cost as described below:

- **Reduce power consumption**: The proposed FP-TBSLB approach will reduce power consumption in comparison with traditional load balancing and task scheduling methods. In this approach, schedulers are aimed to send accumulated tasks in the task queue of an overloaded VM to a destination VM that is located on an active PM. Therefore, the need to turn on more PM is reduced. In contrast, for VM migration, more resource capacity on host PM is needed and it is impossible

for every case to avoid choosing an idle PM, even though hypervisor is set to use current active PM as new host for VMs. Therefore, using FP-TBSLB the number of active PMs and power consumption will be reduced compared with VM migration approach for load balancing, as the less number of active PM and CPUs means the less power consumption (Liao, Jin & Liu 2012).

- **Reduce the overall memory involved (idle memory)**: The other efficient feature of FP-TBSLB approach is minimizing the amount of idle memory which is prepared during the load balancing process.

During VM migration process, the amount of memory allocated to original VM on primary PM and the amount of memory considered for this VM on destination host PM is idle. In addition, in VM migration, an amount of memory is consumed for recording last user activities. In FP-TBSLB approach, the overall memory consumption is equal to memory allocated to VM on original host PM, plus small portion of memory involved on destination host PM that is used for the transferred tasks from original VM.

Therefore, using FP-TBSLB approach that eliminates the process of VM migration, less amount of idle memory will be created.

- **Reduce total time taken for load balancing**: In VM migration, the total time taken for load balancing is equal to the time utilized for migration of an entire state of a VM. This time in FP-TBSLB approach is reduced to the time consumed for transferring some tasks.
- **Reduce penalty and total cost for cloud provider**: Penalty is the amount of cost which should be paid if the workload is under-provisioned with longer response time that occurs when VM is slowed down during migration process. Proposed approach avoids VM migration for large size VMs and as a result VMs will not be slowed down during load balancing process. Considering the fact that

FP-TBSLB approach also reduces power and memory consumption, it consequently decreases the cost as well.

5) **Prepare AS_ORM prototype**: The AS-ORM software will be designed appropriately to be applicable in hypervisor layer of cloud environments. To do this, the developed approaches are coded in Java. In addition, the Jswarm open source package is modified and extended in this study and called MO-Jswarm package. The MO-Jswarm package is able to support multi-objective optimization models with up to four objectives.

1.5 RESEARCH METHODOLOGY

Research methodology is the "collections of problem solving methods governed by a set of principles and a common philosophy for solving targeted problems" (Gallupe 2007). Several research methodologies such as case study, field study, design research, field experiment, laboratory experiment, survey, and action research have been proposed and applied in the domain of information systems. The methodology of this research is planned according to the practice of design research (Niu, Lu & Zhang 2009), which has been proposed and applied in information systems.

1.5.1 GENERAL METHODOLOGY

The design research methodology as presented in Figure 1.2 includes five basic stages (Niu, Lu & Zhang 2009):

I. **Awareness of problem**: This is the first step where limitations of existing applications are analyzed and significant research problems are acknowledged. The research problems reflect a gap between existing applications and the expected status. Research problems can be identified from different sources: industry experience, observations on practical applications and literature review. A clear definition of the research problem provides a focus for the research throughout the development process. The output of this phase is a research proposal for new research effort.

II. **Suggestion**: This phase follows immediately behind the identification of research problems where a tentative design is suggested. The tentative design describes what the prospective artefacts will be and how they can be developed. Suggestion is a creative process during which new concepts, models and functions of artefacts are demonstrated. The resulting tentative design of this step is usually one part of the research proposal.

III. **Development**: This phase considers the implementation of the suggested tentative design artefacts. The techniques for implementation will be based on the artefact to be constructed. The implementation itself can be simple and does not need to involve novelty; the novelty is primarily in the design not the construction of the artefact. The development process is often an iterative process in which an initial prototype is first built and then evolves as the researcher gains a deeper comprehension of research problems. Thus, the output of the suggestion step is also feedback of the first step, whereby the research proposal can be revised. This step includes the following sub-steps to create the prototype (Niu, Lu & Zhang 2009): a) planning, b) analysis, c) design, d) development, e) testing, f) implementation, and g) maintenance.

IV. **Evaluation**: This phase consider the evaluation of the implemented artefacts. The artefacts performance can be evaluated according to criteria defined in the research proposal and the suggested design. The evaluation results, which might not meet the expectations, are fed back to the first two steps. Accordingly, the proposal and design might be revised and the artefacts might be improved.

V. **Conclusion**: This is the final phase of a design research effort. Typically, it is the result of satisfaction with the evaluation results of the developed artefacts. However, there are still deviations in the behavior between the suggested proposal and the artefacts that are actually developed. A design research effort concludes as long as the developed artefacts are considered as 'good enough' wherein the anomalous behavior may well serve as the subject of further research.

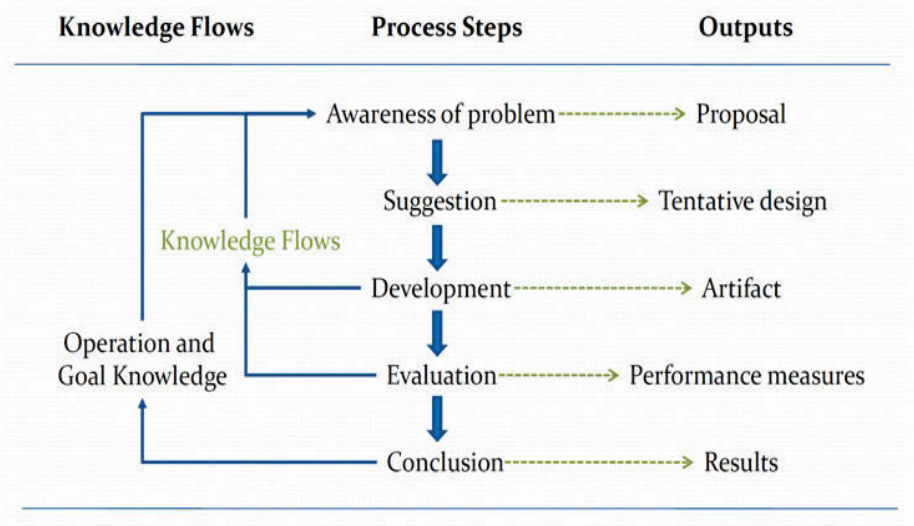


Figure 1.2: The general methodology of research

1.5.2 RESEARCH PLAN

Considering designed general methodology of research, the research plan of this study consisted of the following steps and is summarized in Figure 1.3:

Step 1. **Select a topic**: The choice of a research topic can arise from personal interest, from observation, or from the literatures describing previous theory and research in the area, from social concern or as the outcome of some currently popular issues. The topic of this research was chosen from the previous literature and research and also the author's observation and experience in the process industry.

Step 2. **Review the literature**: Irrespective of the reason for choosing a particular topic, a literature review of previous research in the topic area is an essential component of the research process. Existing literature in the related areas was retrieved and critically reviewed.

Step 3. **Finalize research problems**: The results of the literature review helped to define the specific research questions for this research. The research questions were directly addressed in this research study. As research questions grew clearer and more definite, more literature closely related to the research questions was

reviewed. Existing work is compared to the desirable expectations; gaps and limitations are identified.

Step 4. **Design AS-ORM framework**: After identifying research problems, the need for designing a comprehensive autonomic system for optimal resource management (AS-ORM) is recognized. To develop AS-ORM framework and satisfy Objective 2 of the research the following specifications of cloud environments are considered:

- There are several heterogeneous clouds which are federated.
- There are two types of service request in terms of their priority: (1) advance reservation and (2) benefit-effort.
- Cloud environments are dynamic.
- There are different types of task in terms of their complexity: (1) data intensive and (2) computationally intensive.

Based on the outputs of Step 3 and the environment specifications, the AS-ORM functions and goals are defined to design a practical framework for optimal resource management that covers both cloud providers' and customers' expectations for delivering and receiving high quality cloud services with lowest possible cost. The defined goals and expected AS-ORM functions are summarized as follow:

- Minimize service response time and job makespan
- Minimize energy consumption and cost
- Optimize load balance over cloud virtual and physical resources
- Prevent network overload
- Balance task migration: developing a task assignment algorithms to transfer or migrate tasks from heavily to lightly loaded process nodes so that no processors are idle while there are other tasks waiting to be processed (Zomaya & Yee-Hwei 2001)

- Optimize task scheduling considering available resources (CPU and memory) and network bandwidth (Song, Hassan & Huh 2010).
- Optimize cloud resource utilization

Considering these facts, a novel framework for AS-ORM is proposed that has three sub-systems including: (1) a sub-system to forecast resource utilization over a cloud cluster by VMs' Workload Prediction (WP sub-system), (2) a Multi-Objective Task Scheduling (MOTS) sub-system, and (3) a Fuzzy Predictable Task Based Load Balancing (FP-TBSLB) sub-system that is an optimal alternative for current load balancing approach (i.e. VM migration).

Step 5. **Determine AS-ORM requirements**: To identify the optimization aspects for both task scheduling and resources allocation the following criteria that are related to cloud services(SaaS, PaaS and IaaS) are considered and analyzed:

- The QoS indicators
- Performance modeling in cloud
- Pricing strategy for cloud services
- Type of service level agreement
- The objectives of the optimization models

In this analysis, the major goals and sub-goals of AS-ORM are initially identified, after which important decisions that need to be made and the related solutions are provided.

Step 6. **Develop fuzzy workload prediction models**: To forecast available resource capacity of VMs in a cloud cluster, two VM workload prediction models are investigated in this study including: (1) DBN-based and (2) ES&NN-based workload prediction models. The first prediction model is applied to estimate VMs and PMs' resource utilization based on historical CPU usage data and the probability results of DBN. The ES&NN-based prediction model is also

developed to predict the performance of VMs and determine poorly performing and under-loaded VMs based on the historical data, and current changes in VMs resource usage. In addition, the results of these models consequently determine the PMs' hotspots. These models are applicable for VMs that are applied to deliver SaaS, PaaS or IaaS.

Step 7. **Develop a multi-objective task scheduling model**: In this step a new multi-objective model is developed to offer the best pattern for distributing tasks over virtual resources of a cloud cluster. To achieve this goal, first a multi-objective task scheduling problem is designed based on cloud customers' and providers' expectation of service quality and cost. Then, two multi-objective evolutionary algorithms (MOPSO and MOGA) are extended and applied to solve the proposed task scheduling problem considering task transfer time, task execution cost/time, the length of the task queue of VMs, and power consumption.

To determine the tasks execution time, the execution time of each task on chosen processor is calculated. To calculate tasks transferring time, two types of complex applications/tasks are taken into account. The one is computationally intensive, the other is data intensive. For transferring data intensive applications, the scheduling strategy decreases the data movement which means it decreases the transferring time but for transferring computationally intensive tasks, the scheduling strategy schedules the tasks to the high performance machines (Guo et al. 2012). In this research, bandwidth capacity is also considered as an effective variable on minimizing the task transfer time mainly for data intensive tasks. In addition, available amount of memory and numbers of CPU on destination VM, and CPUs speed are considered to enhance performance for computationally intensive tasks. Considering this fact that the minimum active PMs in clouds will consume minimum energy (Liao, Jin & Liu 2012), for minimizing energy consumption, the proposed approach decreases energy consumption by avoiding

employing idle PMs or processors because the less number of active PMs and CPUs will decrease energy consumption and costs (Liao, Jin & Liu 2012). Finally, for minimizing execution cost the total cost per hour will be calculated.

Step 8. **Develop a Fuzzy Predictable Task Based System Load Balancing (FP–TBSLB) approach**: Live VM migration methods have been applied previously for achieving system load balancing in cloud computing. Live VM migration is a technique for transferring an active VM from one physical host to another without disrupting the VM. This technique has been proposed to reduce the downtime for migrated VMs. As VMs migration takes much time and cost in comparison with tasks migration, this study develops a novel approach to confront the problem of poorly performing VMs and overloaded PMs and to achieve system load balancing, by assigning the arrival task from poorly performing VMs to other compatible VMs in a cloud environment.

Step 9. **Develop a software prototype based on designed AS–ORM**: In this stage of the research an AS-ORM software prototype is developed by applying Java programming language according to the proposed AS-ORM system, and its sub-systems for predicting resources utilization, task scheduling optimization and optimal load balancing. This software has three modules that are described as follows:

- MOTS module that is developed by extending MO-Jswarm and NSGA-II open source packages.
- WP module that is implemented in MATLAB and SMILE
- TBSLB module that is developed by applying HTCondor implemented in a VMware-vSphere based private cloud environment with VMware ESXi hypervisor.

This software can be applied in the hypervisor layer of a cloud to offer optimal resource management in a cloud environment. To develop AS-ORM software, the following steps are conducted:

- **Planning**: Define the system to be developed. Set the scope and define high-level system requirements. Determine the project plan and establish milestones including tasks and resources of the project, and identify critical success factors based on end users' and experts' points of view to develop the system.
- **Analysis**: Design the technical architecture required to support the system models and algorithms that are developed in previous steps
- **Design**: Build a technical blueprint of how the system will function. Technical architecture defines the hardware and software equipment required to run the system. The design phase uses models and graphical representation of the system.
- **Development**: During this phase the system is developed by producing the technical architecture, database and programs.
- **Testing**: This phase involves writing the test conditions, developing detailed steps the system must perform along with the expected results of each sub-system. The sub-systems are tested to verify that they actually work and meet all of the requirements defined. Testing is done under conditions as close to operational as possible.
- **Implementation**: Implementation includes making the system operational, writing detailed user documentation, and providing training for the systems end users.
- **Maintenance**: Monitor sub-systems to ensure they continue to function. Establish a help report to support system end users and provide an environment to support system changes and upgrades.

Step 10. **Implement and evaluate the proposed AS-ORM**: The proposed system is validated in different steps of its development according to experimental results in simulation environments; finally the system prototype is validated against abovementioned abilities in a real VMware-vSphere based cloud environment.

Step 11. **Writing up the thesis**: Writing up the PhD thesis comes at the end of the research.

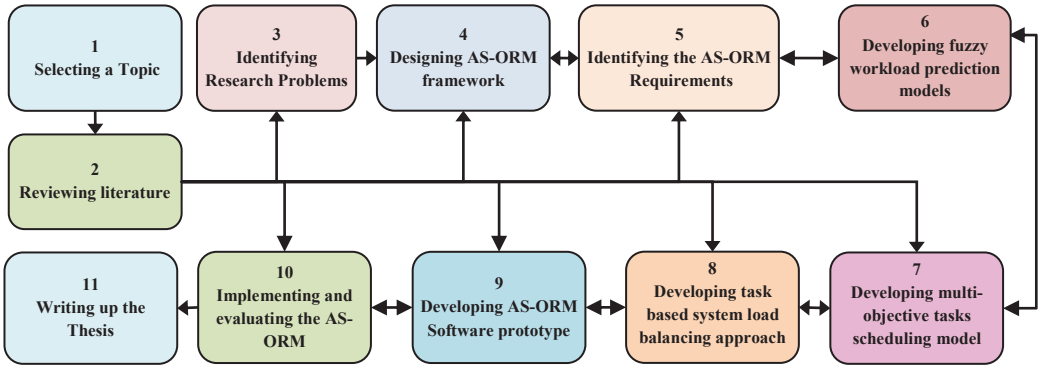


Figure 1.3: **Research process plan**

1.6 THESIS STRUCTURE

This thesis contains eight chapters as shown in Figure 1.4. Chapter 1 presents the research background, challenges, objectives, contributions, methodology, and is presenting the thesis structure. Chapter 2 introduces the preliminary information about cloud computing (e.g. cloud architecture and services, physical topology, resources, middleware and components) and reviews the literature in regard to autonomic resource management, VM workload prediction, task scheduling, and load balancing in cloud environments. Chapter 3 presents an AS-ORM in cloud environments. Chapter 4 introduces a virtual machine workload prediction sub-system. Chapter 5 describes the multi-objective task scheduling sub-system in detail. Chapter 6 represents a fuzzy predictable task based system load balancing sub-system. Chapter 7 shows the implementation and evaluation of the proposed AS-ORM performance. Chapter 8 presents the conclusion and future research directions of this study.

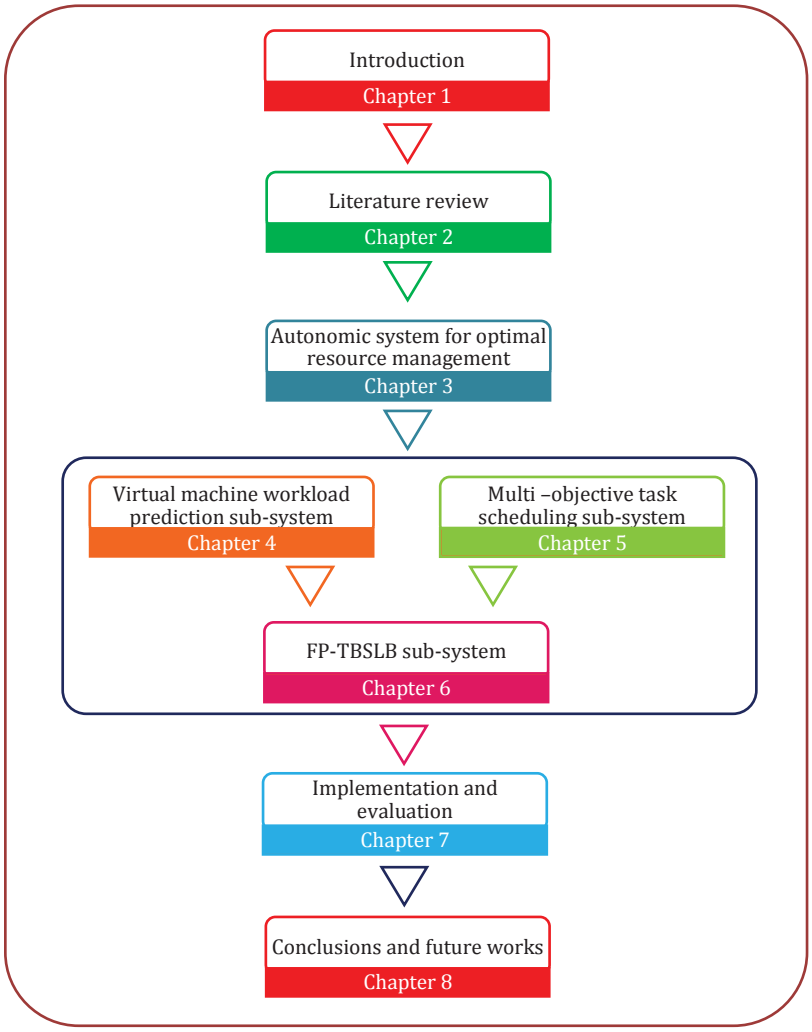


Figure 1.4: Thesis structure

1.7 Publications and Awards Related to This Research

Some chapters of the thesis are based on articles that were published in the peer-reviewed scientific literature during my Ph.D. education. In addition, the research study has gained several awards. The details are as follows:

Peer Reviewed International Journal Papers

1) Ramezani, F., Lu, J., Taheri, J. & Zomaya, A. (2016) "A Multi-Objective Load Balancing System for Cloud Environments", The Computer Journal (Submitted, ERA Tire A* Journal).

2) Ramezani, F., Lu, J., Taheri, J. & Hussain, F. (2014) "Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments", The World Wide Web Journal, Volume 18, Issue 6, Pages 1737-1757 (ERA Tire A Journal).

3) Ramezani, F., Lu, J. & Hussain, F. (2013) "A Tasks-Based System Load Balancing in Cloud Computing Applying Particle Swarm Optimization", International Journal of Parallel Programming, Vol. 42, Issue 5, Pages 739-754 (ERA Tire A Journal).

4) Ramezani, F. & Lu, J. (2012) "A Fuzzy Intelligent Decision Support System for Projects Assessment", Journal of Enterprise Information Management, Vol. 27, Issue3, Pages 278 – 291 (ERA Tire C Journal).

PEER REVIEWED CONFERENCE PAPERS

5) Ramezani, F., Naderpour M., Lu, J. (2015) "Handling Uncertainty in Cloud Resource Management Using Fuzzy Bayesian Networks", the 24th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Istanbul, Turkey (Tier A conference).

6) Ramezani, F., Lu, J. & Hussain, F. (2013) "Task Scheduling Optimization in Cloud Computing Applying Multi-Objective Particle Swarm Optimization", 11th International Conference on Service Oriented Computing (ICSOC 2013), Berlin, Germany.

7) Ramezani, F., Lu, J. & Hussain, F. (2013) "A Fuzzy Predictable Load Balancing Approach in Cloud Computing", The 2013 International Conference on Grid & Cloud Computing and Applications, Nevada, USA.

8) Ramezani, F., Lu, J. & Hussain, F. (2013) "A Fuzzy Decision Support System for Optimizing Resource Management in Cloud Computing", 15th IFSA, Edmonton-Canada.

9) Ramezani, F., Lu, J. & Hussain, F. (2012) "Task Based System Load Balancing Approach in Cloud Environments", Proceedings of International Conference on Intelligent Systems and Knowledge Engineering (ISKE), Beijing-China.

10) Ramezani, F. & Lu, J. (2012) "A Cognitive Group Decision Support System for Projects Evaluation", Proceedings of the 10th International FLINS Conference on Uncertainty Modeling in Knowledge Engineering and Decision Making, Istanbul-Turkey, pp. 231-236.

11) Ramezani, F. & Lu, J. (2012) "A Fuzzy Group Decision Support System for Projects Evaluation", Proceedings of the 14th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU), Catania-Italy, pp. 160-169.

12) Ramezani, F., Lu, J. & Memariani, A. (2011) "A Dynamic Fuzzy Multi-Criteria Group Decision Support System for Manager Selection", Proceedings of International Conference on Intelligent Systems and Knowledge Engineering (ISKE), Shanghai-China, pp. 265-274.

13) Ramezani, F. & Lu, J. (2011) "A New Approach for Choosing the Most Appropriate Fuzzy Ranking Algorithm for Solving MADM Problem", Proceedings of International Workshop on Autonomous Systems, Mallorca- Spain, pp. 13-24.

AWARDS

- 2015 Higher Degree Research Student High Quality Publication Award, the Faculty of Engineering and Information Technology, the University of Technology Sydney, Australia.
- 2014 Higher Degree Research Student High Quality Publication Award, the Faculty of Engineering and Information Technology, the University of Technology Sydney, Australia.

Chapter 2.

LITERATURE REVIEW

2.1 INTRODUCTION

To get a better understanding of this thesis, this chapter explains important background and preliminary information regarding cloud computing, clouds layers architecture and services, service-level agreement, pricing strategy in cloud computing, physical topology of cloud, cloud resources, cloud middleware and components, and cloud resource management system in sections 2.2 to 2.9. Section 2.10 describes the importance of developing autonomic resource management for a cloud environment and explains related concepts. Sections 2.11 to 2.13 provide the reviewed literature with respect to VM workload prediction, task scheduling, and load balancing in cloud environments.

2.2 CLOUD COMPUTING

According to NIST (National Institute of Standards and Technology), Cloud Computing is defined as "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 or service provider interaction"(Vvirtual's Blog 2011; Wang et al. 2012). Wang et al. (2012) has introduced a cloud as a network of data centers all over the

world, where each center is composed of thousands of computers working together that can perform the functions of software on a personal or business computer by providing users access to powerful applications, platforms, and services delivered over the Internet. This metaphor represents the intangible, yet universal nature of the Internet that is capable of providing scalable, customized and inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way by a wide range of geographically dispersed users (Pillai & Ozansoy 2012). Thus, cloud computing provides the users with large pools of resources in a transparent way along with a mechanism for managing the resources so that a user can access it ubiquitously and without incurring unnecessary performance overhead. The cloud provides and assures application based Quality-of-Service (QoS) guarantees to its users. Wang et al. (2012) summarized the key features of cloud computing as follows:

- Agility – helps in rapid and inexpensive re-provisioning of resources.
- Location Independence – resources can be accessed from anywhere and everywhere.
- Multi-Tenancy – resources are shared amongst a large pool of users.
- Reliability – dependable accessibility of resources and computation.
- Scalability – dynamic provisioning of data helps in avoiding various bottleneck scenarios.
- Maintenance – users (companies/organizations) have less work in terms of resource upgrades and management, which in the new paradigm will be handled by service providers of Cloud Computing.

In a cloud environment any user with an Internet connection can access the cloud and its provided services. Since these services are often connected, users can share information between multiple systems and with other users. Examples of cloud computing include online backup services, social networking services, and personal data services such as Apple's Mobile Me. Cloud computing also includes online applications, such as those offered through Microsoft online services. Hardware services, such as redundant

servers, mirrored websites, and Internet-based clusters are also examples of cloud computing (Devi, Gupta & Choudhary 2014).

There are three main types of cloud environments including private, public and hybrid clouds. A private cloud represents a set of virtualized data centers under the ownership of a single administrative domain (i.e., the cloud service provider). Unlike in a public cloud (such as Amazon Web Services, Google Cloud, or Joyent Compute) where the various layers may be offered by multiple providers,in a private cloud the entire stack is controlled by a single provider and so it has access and control over the various applications, middlewares and infrastructure simultaneously (Ghanbari et al. 2012). Hybrid cloud is a cloud computing environment which uses a mix of on-premises, private cloud and public cloud services with orchestration between the two platforms. Hybrid cloud gives businesses greater flexibility and more data deployment options by allowing workloads to move between private and public clouds as computing needs and costs change (Varia 2008). The public and private clouds in a hybrid cloud arrangement are distinct and independent elements. This allows organizations to store protected or privileged data on a private cloud, while retaining the ability to leverage computational resources from the public cloud to run applications that rely on this data. This keeps data exposuremore secure because sensitive data are not stored long-term on the public cloud component (Moltó, Caballer & de Alfonso 2016).

2.3 Clouds Layers Architecture and Services

Cloud can be viewed as a layered architecture where services of a higher layer can be composed from services of the underlying layer. The reference model of Buyya et al. (2009) explains the role of each layer in an integrated architecture. A core middleware manages physical resources and the VMs deployed on top of them. In addition, it provides the required features (e.g., accounting, billing, SLA management, QoS negotiation and execution management) to offer multi-tenant pay-as-you-go services. Cloud development environments are built on top of infrastructure services to offer application development

and deployment capabilities; in this level, various programming models, libraries, APIs, and mashup editors enable the creation of a range of business, Web, and scientific applications. Once deployed in the cloud, these applications can be consumed by end users (Buyya, Broberg & Goscinski 2011).

Based on this architecture, cloud computing services are divided into three classes, according to the abstraction level of the capability provided and the service model of providers, namely: (1) Infrastructure as a Service, (2) Platform as a Service, and (3) Software as a Service (Buyya, Broberg & Goscinski 2011).

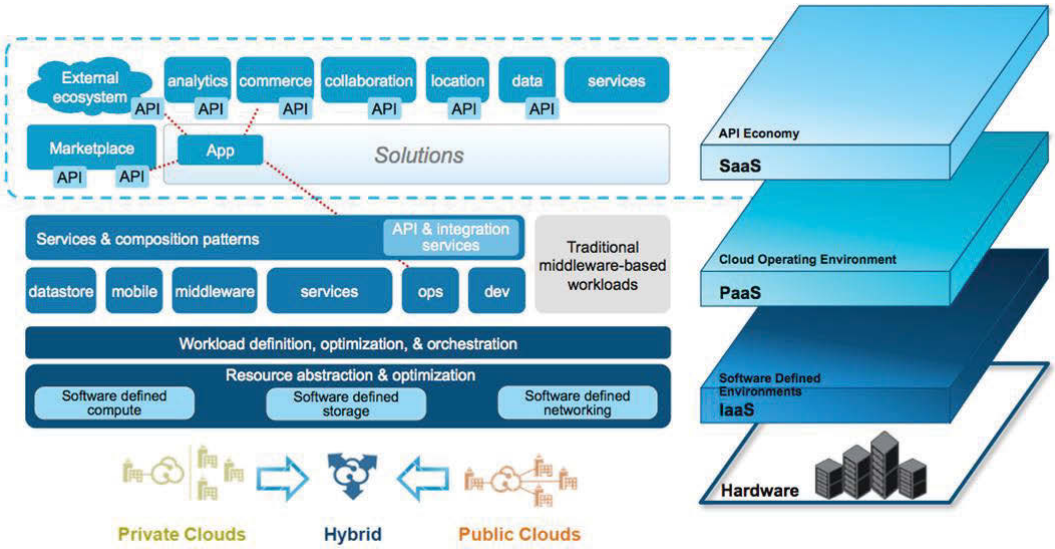


Figure 2.1: Cloud computing services (Angel Diaz & Chris Ferris 2013)

2.3.1 Infrastructure as a Service

Offering virtualized resources (processing, storage, network and other fundamental computing resources) on demand is known as Infrastructure as a Service (IaaS). A cloud infrastructure enables on-demand provisioning of servers running several choices of operating systems and a customized software stack. Infrastructure services are considered to be the bottom layer of cloud computing systems (Buyya, Broberg & Goscinski 2011). The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of

select networking components (e.g., host firewalls). IaaS focus on operations (Example: Amazon EC2) (Platform as a Service Magazine 2015).

2.3.2 PLATFORM AS A SERVICE

In addition to infrastructure-oriented clouds that provide raw computing and storage services, another approach is to offer a higher level of abstraction to make a cloud easily programmable, known as Platform as a Service (PaaS). A cloud platform offers an environment on which developers create and deploy applications using programming languages, libraries, services, and tools supported by the provider. It is not necessary for PaaS customer to know how many processors or how much memory that applications will be using. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment (Platform as a Service Magazine 2015). In addition, multiple programming models and specialized services (e.g., data access, authentication, and payments) are offered as building blocks to new applications (Buyya, Broberg & Goscinski 2011). PaaS focuses on developers (Example: CloudBees, dotCloud or AppFog) (Platform as a Service Magazine 2015).

2.3.3 SOFTWARE AS A SERVICE

On-premises software (often abbreviated as on-prem software) is installed and run on computers on the premises (in the building) of the person or organization using the software, rather than at a remote facility, such as at a server farm or cloud somewhere on the internet. On-premises software is sometimes referred to as "shrink wrap" software, and off-premises software is commonly called "Software as a Service (SaaS)" or "computing in the cloud" (Mangaiyarkarasi, Sureshkumar & Elango 2013).

In SaaS, the applications that reside on the top of the cloud stack are delivered. Services provided by this layer can be accessed by end users through Web portals. Therefore, consumers are increasingly shifting from locally installed computer programs to on-line

software services that offer the same functionally. Traditional desktop applications such as word processing and spreadsheet can now be accessed as a service in the Web. This model of delivering applications alleviates the burden of software maintenance for customers and simplifies development and testing for providers (Buyya, Broberg & Goscinski 2011). The SaaS consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user specific application configuration settings. SaaS focus on end users (Examples: Gmail, Microsoft Office 365 or Salesforce) (Platform as a Service Magazine 2015).

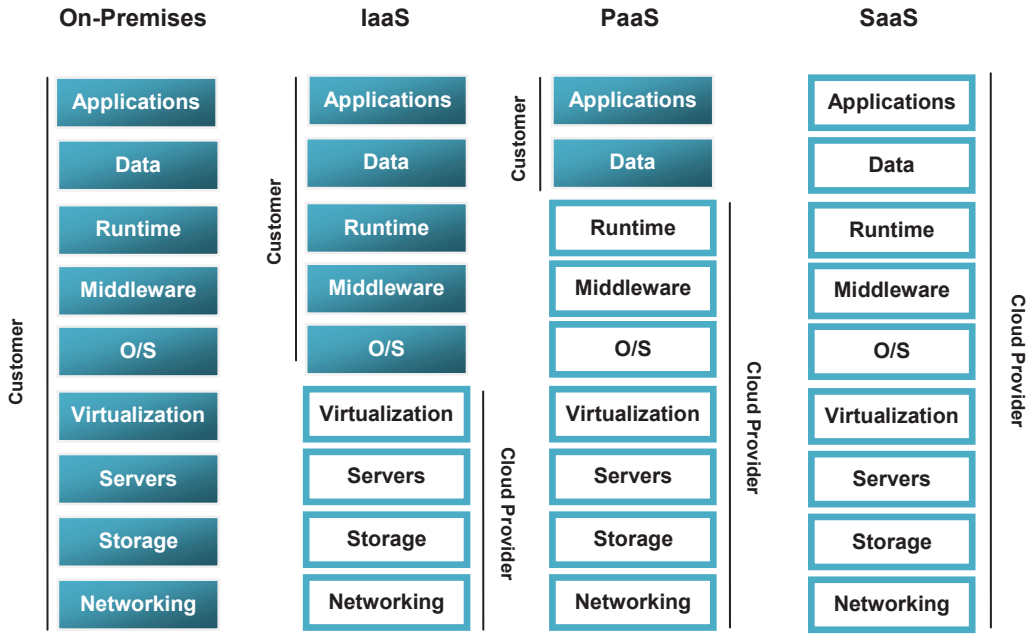


Figure 2.2: Customer and cloud provider responsibilities in different service type (Platform as a Service Magazine 2015)

2.4 Service-Level Agreement

A Service-Level Agreement (SLA) is a part of a service contract where the level of service is formally defined. SLAs are offered by cloud service providers to express their commitment to delivery of a certain QoS and determine the level(s) of service being sold in plain language terms. To customers it serves as a warranty. An SLA usually includes

availability and performance guarantees. Additionally, metrics must be agreed upon by all parties as well as penalties for violating these expectations. The important topics of SLA are: service-level parameter, metric, function, measurement directive, service-level objective, and penalty (Buyya, Broberg & Goscinski 2011).

2.5 Pricing Strategy in Cloud Computing.

In general, cloud providers can offer cloud consumers two provisioning plans for computing resources, namely reservation and on-demand plans (Chaisiri, Lee & Niyato 2011). Based on these plans the cloud hosting providers offer near-infinite cloud resources using different pricing models, e.g. pay-per-use model for on-demand workloads with unforeseeable characteristics, reserved instance pricing model with long-term commitment of availability and spot instance model for workloads with flexible completion time. Therefore, application providers are able to choose an appropriate pricing model based on the anticipated workload characteristics and provision the resources accordingly in the cloud (Sadeka et al. 2012). The cost of utilizing computing resources provisioned by reservation plan is cheaper than that provisioned by on-demand plan (Chaisiri, Lee & Niyato 2011).

2.6 Physical Topology of Cloud

Cloud Architectures address key difficulties surrounding large-scale data processing. In traditional data processing it is difficult to get as many machines as an application needs, and it is not easy to provide the machines when one needs them. It is also a hard process to distribute and co-ordinate a large-scale job on different machines, run processes on them, and provision another machine to recover if one machine fails. In addition, it is impossible to auto-scale up and down running machines based on dynamic workloads. Furthermore, it is difficult to get rid of all those machines when the job is done. Cloud Architectures solve such difficulties (Varia 2008). Figure 2.3 (Culler et al. 1993) illustrates physical topology of a cloud cluster that consists of physical and logical resources such as: physical host (Physical Machine (PM)), Data Storage (DS), network elements and connections, Operating System

(OS), Virtual Machine (VM), energy, bandwidth, delay. A cloud environment also has other related middleware and components including: hypervisor, schedulers, users, jobs, and data files.

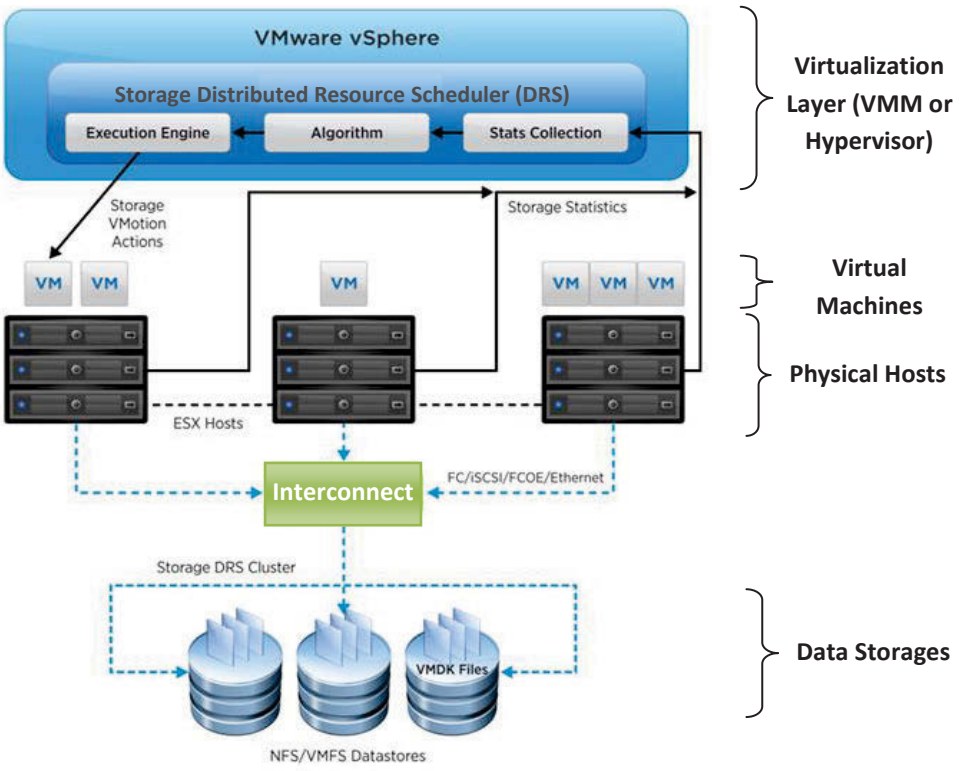


Figure 2.3: Physical topology of a cloud cluster

Cloud Computing provides a novel infrastructure that focuses on commercial resource provision by incorporating virtualization (Grzonka et al. 2015). Virtualization techniques can make hardware resources of physical host (or PM) —such as CPUs, memory— and other physical computing resources —i.e. storage and networking— in a cloud cluster to be consolidated into pools of resources that can be dynamically and flexibly made available and shared with multiple operating systems (vmware 2015). It respectively provides a virtual environment for each operating system allocated on a PM. Therefore, an operating system in the virtual environment can be viewed as an independent VM (Huang et al. 2016). The virtualization layer schedules, allocates the physical resource, and makes each VM think that it totally owns the whole underlying hardware's physical resource (Buyya, Broberg & Goscinski 2011). This creates an integral characteristic for a cloud environment called

elasticity or the ability to rapidly provision and release resources. Elasticity enables the increase and decrease of the number of physical resources allocated to a single VM (Moltó, Caballer & de Alfonso 2016).

In the following physical and logical cloud resources, and its middleware and components are explained in detail.

2.7 CLOUD RESOURCES

Resources are any physical or virtual components of limited availability within a computer system that are categorized in two types: physical and logical resources (Manvi & Krishna Shyam 2014). This section presents the definitions of the main cloud resources that are related to this study.

2.7.1. PHYSICAL RESOURCES

The lowest level of the cloud stack is characterized by the physical infrastructure that constitutes the foundations of the cloud and can be of different nature: clusters, data centers, and desktop computers. On top of these, the IT infrastructure and virtualized environment of the cloud is deployed and managed. A cloud deployment is constituted by data centers hosting hundreds or thousands of machines. This level provides the "horse power" of the cloud (Buyya, Pandey & Vecchiola 2009).

Physical resources of this infrastructure typically include processors, memory, and peripheral devices. Physical resources vary fairly dramatically from computer to computer. A typical mainframe system has several parallel processors, hundreds of disks, tens of millions of bytes of memory, hundreds of terminals, tapes, and other special purpose peripherals, and is connected to a global network with thousands of other similar computers (Manvi & Krishna Shyam 2014).

- **Physical Host (Physical Machine (PM))**: a PM in a cloud cluster is a collection of Computational Nodes (CNs). Each CN consists of several homogeneous processors with identical characteristics, and is characterized by its processing speed, and number of processors. The processing speed for each CN is a relative number to reflect the

processing speed of a CN as compared with other CNs in the system. The number of processors for each CN determines its capability to execute jobs with certain degrees of parallelism in a non-preemptive fashion; i.e., jobs cannot interrupt execution of each other during their run-times (Taheri, Zomaya, et al. 2013).

- **CPU (Central Processing Unit)**: It performs most of the processing inside a computer. The issue in cloud computing is the CPU utilization. CPU utilization refers to a computer's usage of processing resources, or the amount of work handled by a CPU. Actual CPU utilization in cloud varies depending on the amount and type of managed computing jobs/tasks. Certain jobs/tasks require heavy CPU time, while others require less because of non-CPU resource requirements. Proper CPU usage makes it easy to consume massive amounts of compute power for batch processing, data analysis, and high performance computing needs (Manvi & Krishna Shyam 2014).
- **Memory**: The cloud computer architecture asks for a clustered structure of the memory resources in the form of virtual entities. Gone are the days when memory management was done using the static methods. As cloud environment is dynamic and volatile, there is a strong need to inculcate the dynamic memory allocation trends in the cloud based systems. The increased number of cores in cloud servers combined with the rapid adoption of virtualization technologies also creates huge demand for memory (Manvi & Krishna Shyam 2014).
- **Data Storage**: It refers to saving data to a remote storage system maintained by a third party. The Internet provides the connection between the computer and the database. Cloud storage systems generally rely on hundreds of data servers. Because computers occasionally require maintenance or repair, it is important to store the same information on multiple machines. This is called redundancy. Without redundancy, a cloud storage system could not ensure clients that they could access their information at any given time. Most systems store the same data on servers that use different power

supplies. That way, clients can access their data even if one power supply fails (Manvi & Krishna Shyam 2014).

- **Network Elements**: Managing millions of network components (hubs, bridges, switches etc.) leads to unsustainable administrator costs, requiring automated methods for typical system management tasks. The automated methods need to deal with increased monitoring network sizes of several orders of magnitudes higher than current systems. Through this, clouds provide communication as a service (Manvi & Krishna Shyam 2014).

2.7.2. LOGICAL RESOURCES

Logical resources are system abstractions which have temporary control over physical resources. They can support in development of applications and efficient communication protocols. The significance of logical resources in cloud computing is as follows (Manvi & Krishna Shyam 2014).

- **Operating System**: It provides users with a "logical" well-behaved environment to manage physical (hardware) resources as well as offers mechanisms and policies for the control of object/ resources. The operating system performs file management, device management, performance management, security and fault tolerance management, thereby facilitating efficient utilization of the available resources (Manvi & Krishna Shyam 2014).
- **Virtual Machine**: A VM is a software computer that, like a physical computer, runs an operating system and applications. Each VM contains its own virtual, or software-based, hardware, including a virtual CPU, memory, hard disk, and network interface card and is decoupled from specific underlying physical hardware
- **Energy**: The main technique applied to minimize energy consumption is concentrating the workload to the minimum of physical nodes and switching idle nodes off. This approach requires dealing with the power/performance trade-off, as

performance of applications can be degraded due to the work load consolidation (Manvi & Krishna Shyam 2014).

- **Network Throughput/Bandwidth**: In cloud computing, the maximum data throughput in bits per second of a communications link or network access is measured. A typical method of performing a measurement is to transfer a 'large' file from one system to another system and measure the time required to complete the transfer or copy of the file. Higher throughput is desired to make a network work efficiently. Bandwidth management protocols are used to prevent congestion, essentially by accepting or refusing a new-arrival cell. The bandwidth allocation/usage problem is concerned with successful integration of link capacities through the different types of services (Manvi & Krishna Shyam 2014) to prevent network overload in a cloud environment.
- **Delays**: A second or even a millisecond can make a significant difference in the quality of delay-sensitive traffic, the end user experience with cloud-based services, or the ability to trade fairly. A cloud service provider should be able to make accurate decisions for scaling up or down its data-centers while taking into account several utility criteria, e.g., the delay of virtual resources setup, the migration of existing processes, the resource utilization (Manvi & Krishna Shyam 2014).

2.7.3 RESOURCE UTILIZATION

Considering the growth of cloud computing and that resource utilization impacts directly on costs, resource management techniques enable cloud providers to consolidate workloads in order to achieve optimal resource utilization while maintaining SLAs with minimum cost (Weingärtner, Bräscher & Westphall 2015). If X be the total hardware capacity of server, and Y be the current hardware capacity used, then:

- Workload is under-utilization if $\frac{X}{Y} < 1$
- Workload is over-utilization if $\frac{X}{Y} > 1$
- Cloud resource utilization will be optimized when $X \approx Y$

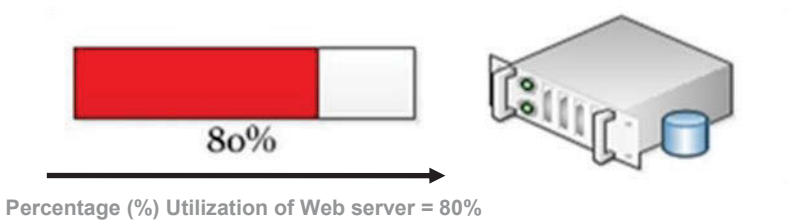


Figure 2.4: Resource utilization

2.8 CLOUD MIDDLEWARE AND COMPONENTS

This section represents the main middleware and components of a cloud cluster.

2.8.1 HYPERVISOR

In computing, a hypervisor —also called Virtual Machine Monitor (VMM)— is one of many hardware virtualization techniques allowing multiple operating systems, termed guests, to run concurrently on a host computer. It is so named because it is conceptually one level higher than a supervisory program (Sahu & Pateriya 2013). The hypervisor is installed on the physical hardware in a virtualized data center, and serves as a platform for running VMs, manages the execution of the guest operating systems/VMs and allows for the consolidation of computing resources. Multiple instances of a variety of VMs may share the virtualized hardware resources. The hypervisor provides physical hardware resources dynamically to VMs as needed to support the operation of the VMs. The hypervisor allows VMs to operate with a degree of independence from the underlying physical hardware. Using this, a VM can be moved from one physical host to another, or its virtual disks can be moved from one type of storage to another, without affecting the functioning of the VM (vmware 2015). For example, VMware ESXi is the hypervisor of VMware vSphere based cloud environment that is elaborated on in Chapter 7.

2.8.2 SCHEDULERS

Schedulers are independent entities in the distributed cloud system, that receive jobs and data files from users and schedule/assign/replicate them to destination processing/storage nodes (i.e. CNs/VMs/SNs). A system can execute multiple parallel jobs simultaneously by

available nodes, while other jobs are queued and wait for processing nodes to become available. The job scheduler manages the queues of waiting jobs and coordinates node allocation (Riesen & Maccabe 2011). Schedulers are in fact the decision makers of the whole system that decide where each job and data file should be executed or stored/replicated, respectively. The main goals of a scheduler are to optimize throughput of a system (number of jobs completed per time unit), provide response time guarantees (finish a job by a deadline), and keep utilization of computation resources high (Riesen & Maccabe 2011). Each individual scheduler can be connected to all CNs/VMs/SNs or only to a subset of them. Schedulers can be either sub-entities of CNs/SNs or individual job/data file brokers that accept jobs and data files from users (Taheri, Zomaya, et al. 2013).

In this study, the schedulers are assumed as individual job brokers to schedule jobs to a set of appropriate VMs to be executed.

2.8.3 USERS

SaaS and PaaS users generate jobs with specific characteristics. Each user is only connected to one scheduler to submit jobs. Although the majority of users only use pre-existing data files in a system, they also can generate their own required data files (Taheri, Zomaya, et al. 2013). In this study IaaS users are also considered who rent a set of VMs from cloud providers.

2.8.4 JOBS AND TASKS

Jobs are generated by users and are submitted to schedulers to be executed by processing nodes. Each job consists of several dependent tasks —described by a Directed Acyclic Graph (DAG)— with specific characteristics, e.g. (1) execution time, (2) number of required processors. Execution time determines the number of seconds a particular task needs to be executed/finalized in the slowest CPU in the system —the actual execution time of a task can be significantly reduced if it is assigned to a faster CPU instead. The number of processors determines a task's degree of parallelism (Taheri, Zomaya, et al. 2013).

Jobs are generated with different shapes to reflect different classes of operations and have the following characteristics: (1) width, (2) height, (3) number of processors, (4) time to execute, (5) shape, and (6) a list of required data files. Width is the maximum number of tasks that can run concurrently inside a job; height is the number of levels/stages a job has; number of processors is the maximum number of processors any of the tasks in the job needs to be run; time to execute specifies the minimum time a job can be run on the slowest CPU in a system and list of required data files determines a list of data files a CPU must download before executing this job. Data to execute each task is provided to it through (1) previously existed data files listed by the list of required data files and/or (2) output of the task's immediate predecessors in the DAG (either as local/temporary data file or inter-processing messages). Jobs' shapes are: (1) series parallel, (2) homogeneous-parallel, (3) heterogeneous-parallel, and (4) single-task. Figure 2.5 illustrates different jobs shapes (Taheri, Zomaya, et al. 2013).

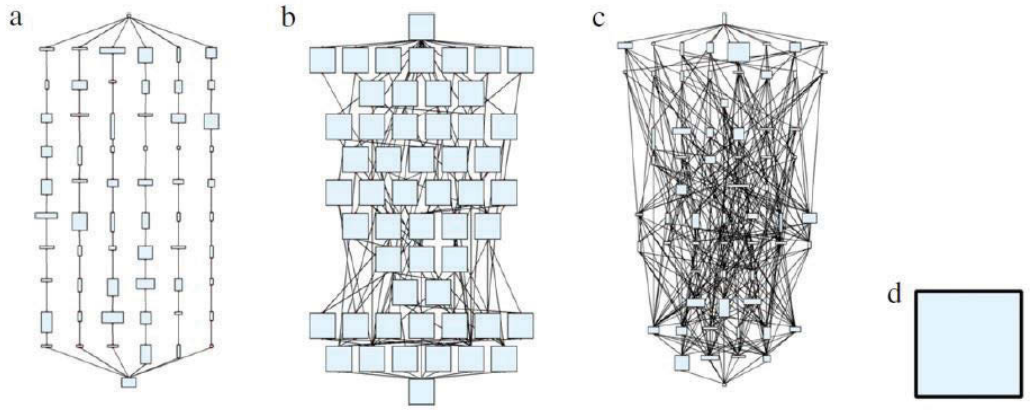


Figure 2.5: Jobs' shapes: (a) series–parallel, (b) homogeneous–parallel, (c) heterogeneous–parallel, and (d) single–task (Taheri, Zomaya, et al. 2013)

2.8.5 DATA FILES

Data files are assumed to be owned by SNs and are allowed to have up to a predefined number of replicas in a system. Schedulers can only delete or move replicas; i.e., the original copies are always kept untouched (Taheri, Zomaya, et al. 2013).

2.8.6 VIRTUAL CLUSTER

A virtual cluster is a collection of VMs that have been configured to act like a traditional High-Performance Computing (HPC) cluster. This typically involves installing and configuring job management software, such as a batch scheduler, and a shared storage system (e.g., network/distributed file system)(Wang, Lu & Kent 2015)

2.9 RESOURCE MANAGEMENT SYSTEM

The Resource Management System (RMS) mechanism helps coordinate IT resources in response to management actions performed by both cloud consumers and cloud providers. Core of this system is the Virtual Infrastructure Manager (VIM) —like VMware vSphere— that coordinates the server hardware so that virtual server instances can be created from the most expedient underlying physical server. A VIM is a commercial product that can be used to manage a range of virtual IT resources across multiple physical servers. For example, a VIM can create and manage multiple instances of a hypervisor/VMM across different physical servers or allocate a virtual server on one physical server to another (or to a resource pool) (CloudPatterns 2015). Tasks that are typically automated and implemented through the resource management system include:

- Managing virtual IT resource templates that are used to create pre-built instances, such as virtual server images
- Allocating and releasing virtual IT resources into the available physical infrastructure in response to the starting, pausing, resuming, and terminating of virtual IT resource instances
- Coordinating IT resources in relation to the involvement of other mechanisms, such as resource replication, load balancer, and failover system
- Enforcing usage and security policies throughout the lifecycle of cloud service instances
- Monitoring operational conditions of IT resources

Resource management system functions can be accessed by cloud resource administrators employed by the cloud provider or cloud consumer. Those working on

behalf of a cloud provider will often be able to directly access the resource management system's native console. Resource management systems typically expose Applications Programming Interfaces(APIs) that allow cloud providers to build remote administration system portals that can be customized to selectively offer resource management controls to external cloud resource administrators acting on behalf of cloud consumer organizations via usage and administration portals (CloudPatterns 2015).

Figure 2.6 illustrates the relations between the resource management system and other components of a cloud environment. The cloud consumer's cloud resource administrator (1) accesses a usage and administration portal externally to administer a leased IT resource. The cloud provider's cloud resource administrator (2) uses the native user-interface provided by the VIM to perform internal resource management tasks (CloudPatterns 2015).

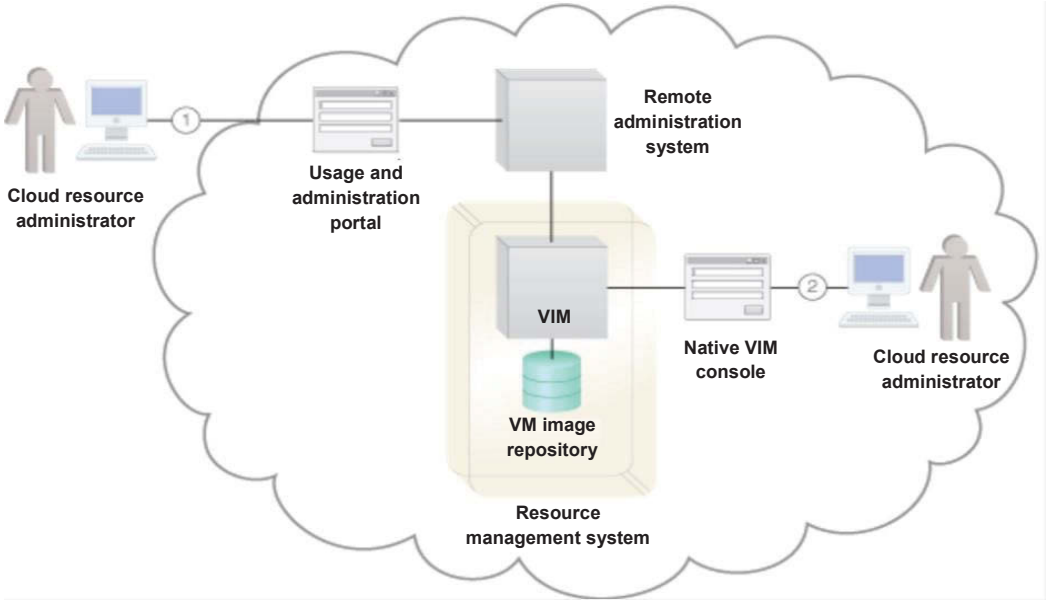


Figure 2.6: Resource management system in a cloud environment (CloudPatterns 2015)

2.10 AUTONOMIC RESOURCE MANAGEMENT

Cloud computing has recently become more and more popular in large-scale computing and data store, due to it enabling the sharing of computing resources that are distributed all over the World. Cloud computing provides resources in a reliable, secure and cost efficient manner (Buyya et al., 2012). Thus, clouds are heterogeneous

environments which are growing in complexity and size, and resource management is a vital aspect of it. Moreover, autonomic cloud is one of the solutions for the management issues that arise with cloud growth(Mendes et al. 2014). It is humanly impossible to manage a cloud that is growing in complexity exponentially; there are too many variables to be considered when managing application and resources in a cloud, especially a public one which has multiple interests and clients (Mendes et al. 2014). Mendes et al. (2014) and Buyya et al. (2012) believe that cloud resource management tools need to be automated and improved in order to cope with the dynamic and elastic nature of clouds. Autonomic systems have characteristics such as self-optimizing, self-monitoring and self-healing which could benefit the cloud environment. However, without a proper profiling and forecasting model to feed the autonomic management model, it may not achieve its true potential (Weingärtner, Bräscher & Westphall 2015). In addition, to feed an autonomic model, an accurate model is needed to predict services computing needs.

Resource management in cloud computing offers several benefits including scalability, quality of service, optimal utility, reduced overheads, improved throughput, reduced latency, specialized environment, cost effectiveness and simplified interface. Resource management problems are classified in the following topics: provisioning, allocation, adaptation, mapping, modeling, estimation, discovery and selection, brokering, and scheduling (Manvi & Krishna Shyam 2014). The general definitions of aforementioned topics are highlighted in Table 2.1 based on Manvi et al. (2014). However, there are other detailed definitions for each topic based on their different aspects and related problems followed by researchers. Resource allocation is also considered as the idea of allocating resources to individual tasks (Vakilinia, Ali & Qiu 2015) and has been optimized by developing job/task scheduling solutions in several works. This definition of resource allocation overlaps with the definition of resource discovery and selection that is introduced by Manvi et al. (2014). In some other work, resource allocation has been identified as the practice of dynamically remapping VMs to PMs aiming to obtain a timely match of supply

and demand (Buyya, Broberg & Goscinski 2011). Based on this, a number of VIMs include a dynamic resource allocation feature that continuously monitors utilization across resource pools and reallocates available resources among VMs according to application needs (Buyya, Broberg & Goscinski 2011).

Significant research has been carried out in parallel in each of the topics of resource management, and several solutions with different points of view are proposed. Optimizing resource management models are proposed in two different approaches: (1) statistic and (2) dynamic resource allocation mechanisms. Some optimization methods such as game-theoretic and stochastic integer programming are widely applied in previous works to develop static model for optimizing resource management. In addition, new threshold-based, feedback-based and online dynamic resource management optimization algorithms are proposed for dynamic resource allocation in clouds. Furthermore, to decrease the delay in response time in resources providing procedure, some prediction based resource management algorithms are proposed applying prediction techniques such as: moving average, auto regression, artificial neural network (Islam et al. 2012b). Resource management optimization problems subject to minimizing the execution time and cost, and maximizing the QoS, have been extensively studied by considering performance metrics such as response time, bandwidth, and link utilization (Xiong & Perros 2006).

Table 2.1: **Problems in resource management**.

Problem	Definition
Resource provisioning	It is the allocation of a service provider's resources to a customer
Resource adaptation	It is the ability or capacity of that system to adjust the resources dynamically to fulfill the requirements of the user
Resource mapping	It is a correspondence between resources required by the users and resources available with the provider
Resource modeling	It is based on detailed information of transmission network elements, resources and entities participating in the network. It is a framework that illustrates the most important attributes of resource management: states, transitions, inputs and outputs within a given environment. Resource modeling helps to predict the resource requirements in subsequent time

	intervals
Resource brokering	It is the negotiation of the resources through an agent to ensure that the necessary resources are available at the right time to complete the objectives
Resource scheduling	A resource schedule is a timetable of events and resources. Shared resources are available at certain times and events are planned during these times. In other words, it is determining when an activity should start or end, depending on its (1) duration, (2) predecessor activities, (3) predecessor relationships, and (4) resources allocated
Resource estimation	It is a close guess of the actual resources required for an application, usually with some thought or calculation involved
Resource discovery and selection	It is the identification of a list of authenticated resources that are available for job submission and to choose the best among them
Resource allocation	It is the distribution of resources economically among competing groups of people (IaaS) or application (SaaS or PaaS).

This study investigates three main topics of resource management including: resource estimation, resource discovery and selection, and resource allocation based on the following perspectives:

- **Resource Estimation**: is considered as estimation of required resources for individual VMs to work with their higher possible performance. Workload prediction models are studied to estimate VMs' workload and CPU utilization and forecast their required resources to deliver high quality cloud services based on fluctuating customers' demands
- **Resource Discovery and Selection**: is targeted as the process of determining a set of virtual resources (VMs) and allocating them to execute jobs/tasks. To do this, job/task scheduling techniques are reviewed to assign arrival tasks to available virtual resources in an optimal way.
- **Resource Allocation**: is investigated as the process of allocating physical resources to VMs. Load balancing approach is considered as a solution to optimally allocate scarce physical resources to virtual resources of cloud environments. This helps to increase the performance of VMs —that are applied to deliver SaaS, PaaS or IaaS— and meet cloud customer demands while optimal resource utilization has been achieved.

Related works in these areas are elaborated in the following sections.

2.11 VIRTUAL MACHINE WORKLOAD PREDICTION

SaaS and PaaS providers are charged by IaaS providers on an hourly basis and therefore need to determine the optimal number of VMs required for their services in each IaaS cluster based on the predicted workload of the VMs, at the same time guaranteeing SLA constraints (Ardagna, Casolari, et al. 2012). IaaS providers also need an estimation of the required capacity of VMs in their cloud clusters for optimal on-demand resource provisioning (Meng et al. 2010). This section presents the literature on VM CPU usage/workload prediction methods that have been used to determine the resources required for given applications, and are mainly applicable to IaaS delivered to SaaS and PaaS providers.

Nagothu et al. (2010b) proposed a new method for load prediction. They separated load prediction into linear and non-linear prediction algorithms. They believed that a linear prediction algorithm could either involve 1-Dim observation sequences or d-Dim observation space signals. They also proposed an alternative method for load prediction which makes use of Burg's algorithm (Nagothu et al. 2010a). Saripalli et al. (2011) demonstrated the use of load prediction algorithms for cloud platforms, using a two-step approach of load trend tracking followed by load prediction, using cubic spline interpolation, and a hotspot detection algorithm for sudden spikes. Meng et al. (2010) proposed an approach called joint-VM provisioning that estimates the aggregate size of multiplexed VMs required, then they consider resource allocation (CPU and memory) to a determined set of compatible VMs instead of allocating resources to an individual VM. This helps to scale up a high utilization VM using the spare capacity (resources) of a co-located VM with low utilization. They developed a workload prediction model to estimate the capacity required for a determined set of VMs. They decoupled the VM workload into regular and irregular fluctuating components. The regular workload refers to deterministic patterns such as trends, cycles and seasonality. The irregular fluctuating workload is the

residue after the regular workload has been removed. They forecasted these two types of workload separately. To forecast the regular workload, they simply assumed that the regular patterns would be preserved in the future, i.e. that a steadily increasing trend would continue to increase at the same rate, and that a daily seasonality would continue to hold. On the other hand, to forecast an irregular workload, they performed a time series forecasting technique based on historic workload patterns. Yang et al. (2013) developed a pattern fusion model for predicting multi-step-ahead CPU loads. They also categorized historical CPU load time series patterns into two sets: patterns that rarely occur and have the lowest possibility of occurrence, and patterns with almost similar trends that have the highest degree of likely occurrence in the future. They discarded the first set of patterns and merged the second set as one pattern to use for prediction. They applied Hamming distance and Euclidean distance to determine similar patterns and then used weighting strategies and Adaboost algorithm to combined prediction results as a predicted CPU load pattern. Islam et al. (2012b) proposed a prediction-based resource measurement method that predicts the CPU utilization pattern for a given application by applying two learning Algorithms: error correction neural network (Theodoridis & Koutroumbas 2008) and linear regression (Weisberg 2005). In addition, they used sliding window (Dietterich 2002) and cross validation (Arlot & Celisse 2010; Efron & Gong 1983) techniques for the training and prediction stages. Ardagna et al. (2012) also developed a prediction-based resource measurement approach by applying the exponential smoothing prediction method. They believed this method is appropriate for predicting run-time and non-stationary behavior. Their model predicts the average number of workload arrivals of a class of web-based services in time scale T_1 on an hourly basis, to determine the required number of VMs (resources) over a federated cloud environment consisting of multiple distributed IaaS data centers (sites). They also performed dynamic load redirection (Ardagna, Panicucci, et al. 2012; Zhu et al. 2009) periodically every time scale T_2 ($T_2 < T_1$) as a short term prediction of changes in the number of workload arrivals in time scale T_1. They used this prediction

model to develop a non-linear resource management optimization technique that minimizes the cost of using IaaS considering capacity allocation and load redirection, while guaranteeing SLA.

There is also a rich literature in this regard that has applied prediction methods such as neural networks, pattern recognition and linear regression to estimate the VM workload in a cloud environment, which can be found in (Weingärtner, Bräscher & Westphall 2015). Current workload and resource prediction methods (see the summary of reviewed literature in Table 2.2) forecast the workload or CPU utilization pattern of the given web-based applications based on their historical data. This gives cloud providers an estimation of the required number of resources (VMs or CPUs) for these applications to optimize resource allocation for SaaS or PaaS (Islam et al. 2012a; Meng et al. 2010; Mian, Martin & Vazquez-Poletti 2013) and reduce their service cost (Ardagna, Casolari, et al. 2012). This study also considers a load balancing optimization model for IaaS. In this case, application behavior cannot be the basic source for VM workload prediction, and everything depends on the sudden decisions of the customer. In addition, there might be an absence of information about running applications on each VM. To solve this problem, a fuzzy Workload Prediction (WP) sub-system is developed that monitors both historical and current VM CPU utilization and workload to predict probable poorly performing VMs. This model is also applied to predict PM resource utilization, and virtual resource discovery.

Table 2.2: The summary of reviewed literature related to VM workload prediction

References	Key Development
Nagothu et al. (2010b)	Used linear and non-linear prediction algorithms
Nagothu et al. 2010a	Used linear and non-linear prediction algorithms
Saripalli et al. (2011)	Used linear and non-linear prediction algorithms
Meng et al. (2010)	Developed joint-VM provisioning approach
Yang et al. (2013)	Developed pattern fusion model

Islam et al. (2012b)	Developed prediction-based resource measurement method by using error correction neural network and linear regression
Ardagna et al. (2012)	Developed a prediction-based resource measurement by using the exponential smoothing prediction method
Ardagna, Panicucci, et al. 2012; Zhu et al. 2009	Developed dynamic load redirection
Weingärtner, Bräscher & Westphall 2015	Estimated the VM workload by using neural networks, pattern recognition and linear regression
Islam et al. 2012a; Meng et al. 2010; Mian, Martin & Vazquez-Poletti 2013	Forecasted the workload or CPU utilization pattern of the given web-based applications to estimation of the required number of resources
Ardagna, Casolari, et al. 2012	Forecasted the workload or CPU utilization pattern of the given web-based applications to reduce their service cost

2.12 TASK SCHEDULING IN CLOUD ENVIRONMENTS

In distributed computing such as grid and cloud computing, hundreds of online users may individually or collectively submit thousands of jobs anytime and anywhere to dynamic resources. Given the large number of arrival tasks resulting from splitting these bulk submitted jobs and considering the amount of data being used by these tasks, their optimal scheduling becomes a serious problem for grid/cloud environments —where stochastic jobs compete for scarce compute and storage resources (Taheri, Choon Lee, et al. 2013). How to adapt to the consequent uncertainties, as well as scheduling overhead and response time, are the main concern in dynamic scheduling (Tong et al. 2014).

There are two types of jobs in the cloud environment: (1) computationally intensive and (2) data-intensive jobs. To schedule computationally intensive jobs, the goal is to schedule jobs among CNsor VMs to minimize the overall makespan of executing all jobs in the whole system; here, it also is assumed that the overall transfer time of all data files (input and output files of each task of the job) is relatively negligible compared to executing jobs. The speed and number of available computer resources in different CNs/VMs and the network capacity between CNs/VMs and SNs are typical considerations taken into account in such

systems. For data-intensive jobs, it is assumed that transfer time of all data files is much more time consuming than executing their dependent jobs. As a result, jobs will need less time to download the associated data files to execute and therefore, the execution time (i.e., makespan of executing jobs plus transfer time of data files) of the system is reduced. The capacity of interconnected network links are typical considerations in such allocations (Taheri, Zomaya, et al. 2013).

There are three main phases of scheduling in such complex systems: (1) resource discovery, (2) matchmaking, and (3) job execution. In the first phase, resource discovery, schedulers conduct a global search to generate a list of all available resources as well as their limitations and history profiles in a system. In the second phase, matchmaking, schedulers try to determine best choices for executing jobs and replicating data files. Capacities of CNs/VMs/SNs as well as quality of the network connecting them are among the basic characteristics that need to be considered by schedulers to perform this phase. In the last phase, job execution, schedulers produce commands for CNs and SNs to execute jobs and replicate data files, respectively (Taheri, Zomaya, et al. 2013).

The resource discovery phase of scheduling is covered in this study by developing a prediction model in Chapter 4. This section focuses on the matchmaking process of schedulers in terms of job scheduling in virtualized environment of clouds, and tasks are scheduled to VMs. The related works in this area are presented as follows.

The task scheduling problem in distributed computing systems is an NP-hard optimization problem that also affects QoS in the cloud environment by optimizing service cost and service response time. Therefore, the use of a heuristic algorithm ensures an acceptable runtime of the scheduling algorithm itself since it significantly reduces the complexity of the search space. In this regard, Song et al. (2010) proposed a general job selection and allocation framework that utilizes an adaptive filter to select jobs and a modified heuristic algorithm (Min-Min) to allocate them. Two objectives —maximizing the remaining CPU capacity, and the utilization of resources— and four criteria —the

resource requirements of CPU, memory, hard-disk and network bandwidth— were considered for the optimization problem. Li et al. (2011) applied an Ant Colony optimization approach to create a better scheduling result with shorter total-task-finish time and mean-task finish time. Li et al. (2012) took resource allocation patterns into account and proposed a task and resource optimization mechanism. Tayal (2011) proposed a fuzzy-GA based optimization approach to enhance the accuracy of GA results for the job scheduling process, which makes a scheduling decision by evaluating the entire group of tasks in the job queue. Juhnke et al. (2011) proposed a multi-objective scheduling algorithm for cloud-based workflow applications to minimize cost and execution time by applying the Pareto Archived Evolution Strategy, which is a type of GA. Lei et al. (2008) and Salman et al. (2002) developed an optimization model to optimize task execution time, and showed that the Particle Swarm Optimization (PSO) algorithm is able to obtain a better schedule than GA in grid computing and distributed systems. Guo et al.(2012) also proposed a multi-objective task scheduling model to minimize task execution time and cost using the PSO algorithm. Taheri et al. (2014) considered the data-files required for jobs from public or private clouds and proposed a bi-objective job scheduling optimization model to minimize job execution and data file transfer time using PSO. A heuristic approach, named JDS-HNN (Job and Data Scheduling using Hopfield Neural Network), is also proposed by Taheri et al. (2013) that simultaneously schedules jobs and replicates data files to different entities of a grid system to minimize the overall makespan of executing all jobs and the overall delivery time of all data files to their dependent jobs. Their approach schedules jobs and replicates data files with respect to (1) characteristics of CNs/SNs in a system, (2) inter-dependences between jobs and data files, and (3) network bandwidth among CNs/SNs to host these jobs and data files (Taheri, Zomaya, et al. 2013). JDS-HNN model is inspired by an algorithm for distributing a variety of stones into different jars. Hopfield Neural Network (HNN) optimizer has been also applied in this model to find the best jar to place a specific stone to concurrently minimize both objectives of the stated data

aware job scheduling problem. These studies mainly emphasized the minimization of the job makespan and task execution cost in their multi-objective optimization models by applying evolutionary algorithms, and the reduction of power consumption is neglected in such studies.

Meanwhile, there are many works which focus on reducing power consumption in their proposed bi-objective task scheduling models by minimizing the value of their developed predefined power consumption objective functions for multi-core processors and cloud environment (Shieh & Pong 2013; Wang, Wang & Cui 2014). Different objective functions have been suggested in these models for estimating energy consumption (Priya, Pilli & Joshi 2013; Tchernykh et al. 2014; Zhang & Guo 2013). Shieh and Pong (2013) designed an energy- and transition-aware algorithm to schedule periodic tasks in multi-core systems considering voltage transition overheads. They suggested an integer linear programming model to find the optimal power-aware scheduling for specific task set and core number. Wang et al. (2014) developed an energy-aware multi-objective bi-level programming model based on MapReduce. They considered energy consumption in the data placement process, combined with a local multi-job scheduling scheme. Their approach has three steps. First, they considered changes in energy consumption along with the performance of servers. Then, their model dynamically adjusts data locality based on the current network state. In the last step, they developed an integer bi-level programming model considering the fact that task-scheduling schemas depend on data placement patterns.

Several investigations have also been conducted into developing energy-aware task scheduling algorithms to optimize energy consumption by applying dynamic voltage scaling technique (Mahabadi, Zahedi & Khonsari 2013; Rizvandi, Taheri & Zomaya 2011; Su et al. 2013; Wang et al. 2013). In these works, the authors determined task-slack time by considering critical and non-critical paths in job DAGs. They then extended non-critical task execution times by scaling down the voltage frequency of corresponding processors. In

other works, energy is reduced by sending non-critical tasks to the most cost efficient VMs (Su et al. 2013).

Although a significant number of studies have been carried out in the area of minimizing task execution/transfer time and/or power consumption in a cloud/grid environment (see the summary of reviewed literature in Table 2.3), the reduction of task queue length in a cluster by considering the workload capacity of VMs has been neglected in earlier multi-objective task scheduling models. Task queue length is an effective factor for reducing job makespan because it determines the waiting and finish time of tasks. In addition, four optimization objectives including: (1) task transfer time, (2) task execution cost, (3) power consumption, and (4) the task queue length of VMs, have been considered in this study to develop a comprehensive task scheduling model.

Table 2.3: The summary of reviewed literature related to task scheduling in cloud environments

References	Key Development
Song et al. (2010)	Developed general job selection and allocation framework by using modified heuristic algorithm (Min-Min) to optimize two Objectives including: maximize the remaining CPU capacity, and resources utilization
Li et al. (2011)	Used Ant Colony optimization algorithm to optimize two Objectives including: minimize total-task-finish time and mean-task finish time
Tayal (2011)	Developed a fuzzy-GA based optimization algorithm
Juhnke et al. (2011)	Used Pareto Archived Evolution Strategy, which is a type of GA to optimize two Objectives including: minimize cost and execution time
Lei et al. (2008) and Salman et al. (2002)	Used and compared PSO and GA algorithms to optimize execution time
Guo et al.(2012)	Used the PSO algorithm to optimize two Objectives: including: minimize task execution time and cost
Taheri et al. (2014)	Developed their model considering the data-files required for jobs by using PSO to optimize two Objectives including: minimize job execution and data file transfer time
Taheri et al. (2013)	Developed heuristic approach, named JDS-HNN by

	using Hopfield Neural Network (HNN) to optimize two Objectives including: minimize jobs makespan and data files transfer time
Shieh & Pong 2013; Wang, Wang & Cui 2014, Priya, Pilli & Joshi 2013; Tchernykh et al. 2014; Zhang & Guo 2013	Focused on reducing power consumption and optimized two Objectives including: optimize power consumption and other objective
Shieh and Pong (2013)	Considered voltage transition overheads by using linear programming model to optimize two Objectives including: minimize energy and transition
Wang et al. (2014)	Developed energy-aware multi-objective bi-level programming model based on MapReduce
Mahabadi, Zahedi & Khonsari 2013; Rizvandi, Taheri & Zomaya 2011; Su et al. 2013; Wang et al. 2013	Considered critical and non-critical paths in job DAGs and used dynamic voltage scaling technique to optimize energy consumption

2.13 Load Balancing in Cloud Environments

Resource allocation strategy is all about integrating cloud provider activities for utilizing and allocating scarce resources within the limit of cloud environment so as to meet the needs of the cloud application (Anuradha & Sumathi 2014). In large scale, efficiently allocating scarce resources among competing interests in decentralized cloud computing networks is a challenging subject for resource managers (Wei et al. 2010). Resources allocation in a cloud cluster should effectively handle workload fluctuations, while providing QoS guarantees to the end users and achieve optimal resource utilization.

Load balancing is a computer networking method to distribute workload across multiple computers or a computer cluster, network links, central processing units, disk drives, or other resources, to achieve optimal resource utilization, maximize throughput, minimize response time, and avoid overload (Wikipedia 2012).

Virtualization technique has improved utilization and system load balancing by enabling VM migration, and has provided significant benefits for cloud computing (Jain et al. 2012). Several methods have been proposed to migrate a VM from a physical host to another one

with more available resources for optimizing cloud utilization. These methods are categorized in two main classes: (1) suspend /resume, and (2) live migration.

The suspend/resume VM migration approach has three steps: pause the original VM, copy the VM's related data (memory pages and processor state) into a new host PM, and then resume the VM on the destination host (Kozuch & Satyanarayanan 2002; Sapuntzakis et al. 2002; Whitaker et al. 2004). Using this method, applications running on the VM need to be stopped and are not made available until the migration process has been completed and all the data have been transferred to the new destination. Moreover, this method results in long VM downtime. To reduce downtime, the ZAP system (Osman et al. 2002) only transfers a process group, but it still uses a stop-and-copy strategy.

In contrast, live migration, in which a running instance of VM is migrated between hosts (PMs) in a local area network disconnecting the client or application. VM live migration is an important feature of virtualization that eliminates the stop-and-copy process and minimizes VM downtime (See Figure 2.7). Live migration also brings the benefits of workload balancing, elastic scaling, fault tolerance, hardware maintenance, sharing infrastructures and transparency to users (Huang et al. 2016).

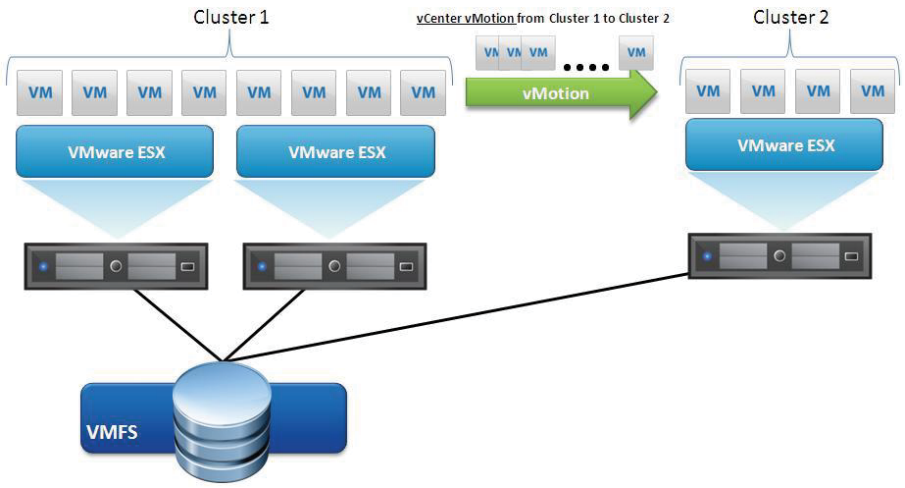


Figure 2.7: VM live migration (*Vvirtual's Blog* 2011)

Jun and Xiaowei (2011) developed a VM live migration policy for the IPv6 network environment. In this migration method, the VM does not provide new services but continues its work and then stops after completing its old services. A pre-copy migration

method is applied by vMotion (Nelson, Lim & Hutchins 2005) and Xen (Clark et al. 2005) for live migration. Using this method, VM's run-time memory state files are pre-copied (migrated) from the source host to the destination host while the VM is still working. This method generates a very large amount of dirty memory and takes a long time, because it is necessary to transfer a large amount of data. In addition, when the dirty memory generation rate in some cases is faster than the pre-copy speed, live migration will be prolonged. To overcome these drawbacks, Jin et al. (2011) suggested using the pre-copy based model from VM live migration in combination with an optimized algorithm that reduces the speed of changing memory by controlling the CPU scheduler of the VM monitor. To ease the VM migration process, Nicolae et al. (2013) developed a repository check pointing strategy called BlobCR that frequently stores live snapshots of the whole VM instances disk. There are also several VM live migration techniques that consider power consumption reduction as well as downtime and migration time. Liao et al. (2012) developed a live VM mapping framework to map VMs onto a set of PMs without significant system performance degradation while reducing power consumption. Sallam and Li (2014) also suggested a multi-objective VM migration technique that considers power and memory consumption, thus making live VM migration more beneficial for cloud providers.

Lin et al. (2011) believed that the load balancing strategies that focus on VM migration for optimizing on-demand resource provisioning needed to be improved. They proposed a threshold-based dynamic resource allocation approach for load balancing in the cloud environment that dynamically allocates the VMs among the cloud's applications based on their load changes. Atif and Strazdins (2014) also developed a similar cloud utilization optimization framework for Application as a Service (AaaS). They used VM monitor facilities (which have traditionally been used for live migration) to create sets of homogenous clusters of computing frame (VMs). They used these clusters to schedule or migrate application tasks over a set of homogenous VMs based on estimated task execution time to optimize resource utilization and enhance application performance. However, this

method cannot be used when a determined homogenous cluster has high utilization and is in an overloaded state.

The reviewed literature in this area is summarized in Table 2.4. The fundamental drawback of these load balancing approaches is that the majority attempt to migrate the VM (Clark et al. 2005; Jin et al. 2011; Jun & Xiaowei 2011; Liao, Jin & Liu 2012). However, the dynamic nature of VMs, especially live migration, makes them difficult to maintain the consistency of security. For example, VMs' live migration among physical servers could spread security vulnerabilities and human negligence in an ignorant and rapid way. This will be a disaster against a pool of virtualized servers for production use, because there are generally no physical firewalls separating the VMs in a virtual environment. Additionally, VM live migration adds difficulties to anomaly detection because the inconsistent performance of VM before, in the course of and after the live migration cover anomalies and attacks (Huang et al. 2016). Moreover, a large amount of memory in both primary and destination PMs are consumed during live VM migration to migrate large size VMs. This problem becomes worse when storage migration is also required and VM disk files should be transferred to other storage. These make the VM slow down during the migration process and consequently increase the service response time.

In this study, by proposing FP-TBSLB approach, the VM migration is not required and consequently the pre-copy process is eliminated. Therefore, FP-TBSLB approachreduces the amount of dirty memory produced, as well as the load balancing time- and power-consumption, compared to VM migration. In addition, the proposed system is not restricted to distributing the extra workload over a set of VMs in predefined clusters. In fact, the new destination VMs are determined dynamically over the entire local cloud environment. A workload prediction model is also developed and applied in this approach to determine poorly performing VMs to reduce the risk of confrontation with a high utilization cluster.

Table 2.4: The summary of reviewed literature related to load balancing in cloud environments

References	Key Development
Jun and Xiaowei (2011)	Developed a VM live migration policy for the IPv6 network environment
(Nelson, Lim & Hutchins 2005) (vMotion) and (Clark et al. 2005)	Developed a pre-copy migration method for live migration
Jin et al. (2011) (Xen)	Combined the pre-copy based model and an optimized algorithm that reduces the speed of changing memory
Nicolae et al. (2013)	Developed a repository check pointing strategy called BlobCR for live migration
Liao et al. (2012)	Developed a live VM mapping framework with minimum system performance degradation while reducing power consumption
Sallam and Li (2014)	Developed bi-objective VM migration technique that considers power and memory consumption
Lin et al. (2011)	Developed threshold-based dynamic resource allocation approach for load balancing
Atif and Strazdins (2014)	Developed cloud utilization optimization framework for AaaS to optimize load balancing
Clark et al. 2005; Jin et al. 2011; Jun & Xiaowei 2011; Liao, Jin & Liu 2012	Developed different load balancing approaches based on VM migration

2.14 SUMMARY

This chapter reviews the background and related literature of this research. First, the preliminary introduction of cloud computing, clouds layers architecture and services, service-level agreement, pricing strategy in cloud computing, physical topology of cloud, cloud resources, cloud middleware and components, cloud resource management system, and autonomic resource management are described. Second, the chapter presents the related works in the area of VM workload prediction, task scheduling, and load balancing in cloud environments to provide a strong foundation for the subsequent analysis, methods, and systems presented in later chapters.

Chapter 3.

AUTONOMIC SYSTEM FOR OPTIMAL RESOURCE MANAGEMENT IN CLOUD ENVIRONMENTS

3.1 INTRODUCTION

Cloud computing is a large-scale distributed computing paradigm driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet. Considering the lack of resources in cloud environments and growing customer demands, cloud providers need to optimize their resource allocation policies to meet Service Level Agreement (SLA) criteria while minimizing their cost. Therefore, it is vital for them to control and balance their resource load and utilization with the purpose of delivering high performance physical and virtual resources.

Virtualization technology is used to increase resource utilization and reduce operating costs of cloud services. Two key mechanisms for flexible resource utilization and load balancing among PM, offered by virtualization are: a) allocating resources dynamically to VMs (i.e., changing CPU share or memory allocation), and b) migrating VMs to other

physical nodes. However, with this flexibility come uncertainty, security issue, cost, resource and time consumption in managing resources.

Considering these facts, resource management is a vital aspect of the complex and enormous environment of clouds. Optimal resource management system should consider the interaction between different events and process in this dynamic environment, and automatically organize physical and virtual resources of clouds in relation to their fluctuating on-demand services.

Consequently,in this chapter an Autonomic System for Optimal Resource Management (AS-ORM) addresses three main topics of resource management in the cloud environment including: (1) resource estimation, (2) resource discovery and selection, and (3) resource allocation. The AS-ORM contains two fuzzy prediction methods for resource estimation. These prediction models estimate VMs'workloadand resource utilization to predict PM load status and hotspots, and handle some of the uncertainty in resource utilization.A multi-objective task scheduling model is also embedded as part of AS-ORMthat suggest optimal pattern to schedule tasks to selected virtual resources. In addition, a fuzzy predictable task based system load balancing approach is proposed as the third part of AS-ORM to promote the current load balancing approaches and optimize resource allocation.

This system supports cloud providers to automatically allocate scarce resources to the on-demand cloud services in an optimal way to reduce service response time, memory usage, power consumption, and cost while boosting the Quality of Service (QoS) expected by cloud customers.

3.2 The AS-ORM Framework

Corporate data centers are in the process of adopting a cloud computing architecture where computing resources are provisioned on a per-demand basis instead of being statically allocated. Such cloud infrastructures should improve the average utilization rates of IT resources (Van, Tran & Menaud 2009), handle peak loads and optimize system performance by optimal management of its scarce resources. Another important issue in

resource management for cloud computing, is how resources should be allocated to cloud services such that the SLAs of all services are met (Hu et al. 2009).

The AS-ORMis developed in this study to satisfy these requirements and achieve optimal resource utilization and the highest QoS in cloud environments. AS-ORM is designed for the complex anddynamic environment of cloud that considers optimizing both virtual resource discovery and selection to execute arrival task, and physical resource allocation to VMs to create high performance clusters based on performance metrics such as response time, bandwidth usage and cost. It is also taken into account that optimization models should be flexible to be able to offer the best solution based on customer preferences (e.g. service cost, time, etc.) that are referred in SLA, and predicted resources utilization.

To achieve these goals and elaborate on the objective functions of the system the following indicators are considered:

- QoS indicators: the appropriate services can be selected by using some indicators such as response time, cost, availability, reliability and reputation, which means that all the cost indexes are smaller than the original service as well as all the efficiency indicators are bigger than the original service (Zhang & Xu 2012).
- Performance modeling in cloud environment: based on (Atif 2010) the most popular performance model is LogP proposed by Culler et al. (1993). LogP was developed as a realistic model for parallel algorithm design, in which critical performance issues could be addressed without reliance on the machine details. In this model, the performance of a system is characterized in terms of four parameters including: latency, overhead, gap and processor. The first three parameters describe the time to perform an individual point-to-point message event, and the last describes the computing capability of a system. The description of the aforementioned parameters are as follows (Atif 2010):
  - Latency: An upper bound on the time to transmit a message from its source to destination.

- Overhead: The time period during which the processor is engaged in sending or receiving a message.
- Gap: The minimum time interval between consecutive message transmissions or consecutive message receptions at a processor. The reciprocal of the gap gives the effective bandwidth in messages per unit time.
- Processor: The number of processors.

- Pricing strategy: the strategy is defined by cloud service providers and are mainly divided into policies: pay-per-use and reserved instance (Sadeka et al. 2012).
- Task types: two different types of tasks in cloud environment in terms of their complexity: (1) Data intensive and (2) Computationally intensive, are investigated.
- Type and measurement components of SLA: SLA provides a framework within which both seller and buyer of a service can pursue a profitable service business relationship. It outlines the broad understanding between the service provider and the service consumer for conducting business and forms the basis for maintaining a mutually beneficial relationship. From a legal perspective, the necessary terms and conditions that bind the service provider to provide services continually to the service consumer are formally defined in SLA. Service-level parameter, metric, function, measurement directive, service-level objective, and penalty are some of the important components of SLA and are described in Table 3.1 (Buyya, Broberg & Goscinski 2011).

Table 3.1: Key components of a service-level agreement

Service-Level Parameter	Describes an observable property of a service whose value is measurable
Metrics	These are definitions of values of service properties that are metrics and constants. Metrics are the key instrument to describe exactly what SLA parameters mean by specifying how to measure or compute the parameter values.
Function	A function specifies how to compute a metric's value from the values of other metrics and constants. Functions are central to describing exactly how SLA parameters are computed from resource metrics.
Measurement directives	These specify how to measure a metric.

Based on these indicators and parameters the following optimization objectives are determined for AS-ORM to achieve:

- Minimizing service response time
- Decreasing VMs' downtime
- Increasing VMs' performance
- Preventing network overload
- Minimizing energy consumption and cost
- Optimizing cloud utilization and achieving better load balance

The framework of AS-ORM is designed based on the aforementioned objective functions to support important topics in clouds resource management including: resource estimation, resource discovery and selection, and resource allocation. In this regard, the AS-ORM has three sub-systems as illustrated in Figure 3.1.These sub-systems are: (1) VM workload prediction sub-system, (2) multi-objective task scheduling sub-system, and (3) fuzzy predictable task based system load balancing sub-system.

The AS-ORM starts with resources allocation to VMs by hypervisor, and scheduling arrival SaaS and PasS task over organized resource pool considering by applying MOTS sub-system. Then it employs the DBN-WP algorithm of WP sub-system to predict PMs' and VMs' workload status, and determine required resources in the target cluster. After that, the AS-ORM applies DBN-WP sub-system to discover whether a host PM is going to be overloaded or not. If the PM is under-loaded for the next time slot, then the workload and resource usage of its allocated VM are examined to be scaled up if it is required. In some cases when the PM is overloaded, the FP-TBSLB sub-system is used to handle the situation as described in Figure 3.1. The overall descriptions of the AS-ORM subsystems are also presented in the following sections.

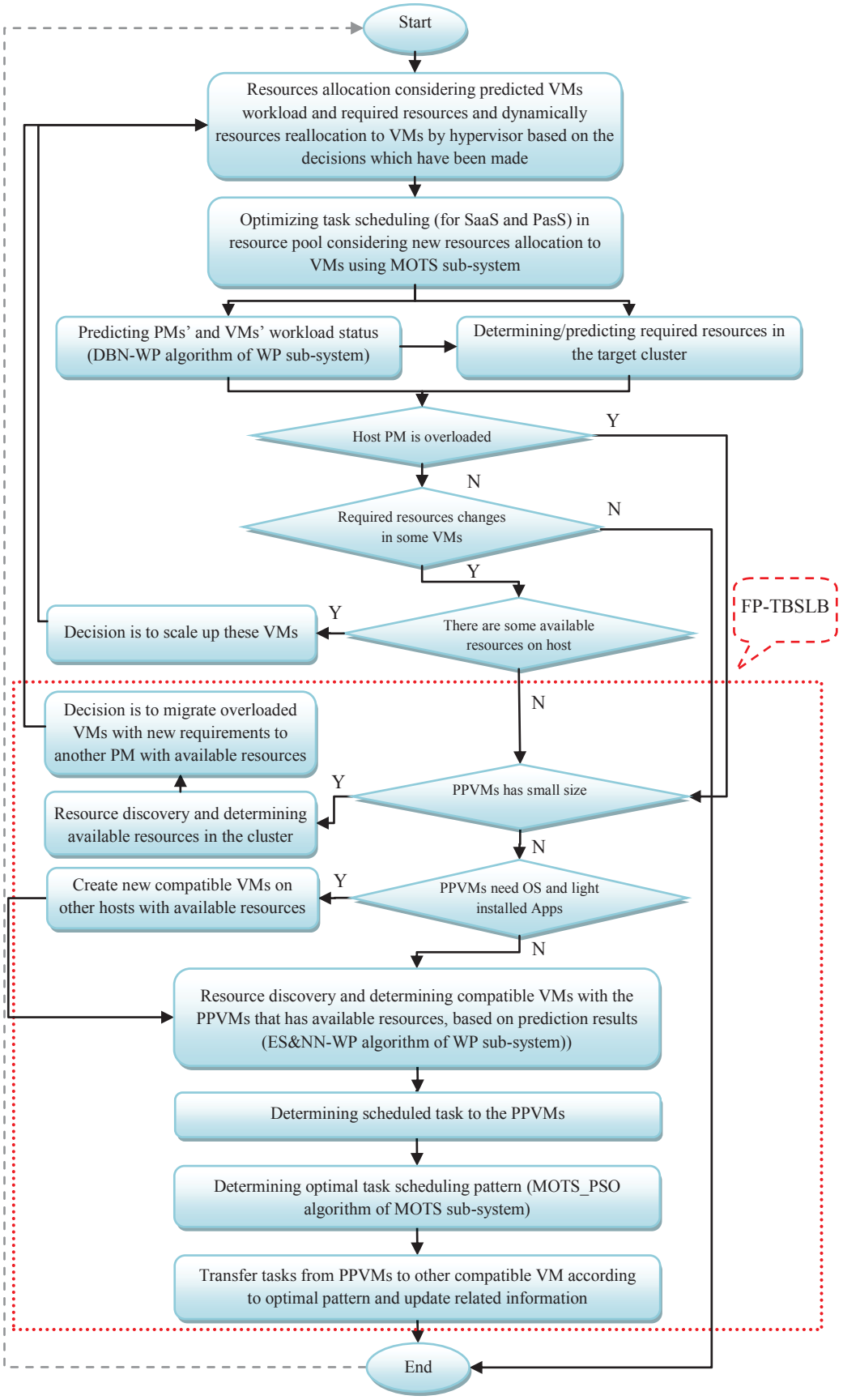


Figure 3.1: The AS–ORM framework

3.2.1 VM WORKLOAD PREDICTION SUB-SYSTEM

To accelerate resource management, the VMs' workload pattern, PM host spots and the required resources should be predicted. Therefore, a WP sub-system is developed to assist the VMMs to discover which VMs would be poorly performed, and which PMs are likely to be overloaded. The sub-system includes two prediction models named: DBN-WP and ES&NN-WP which are developed based on Dynamic Bayesian Networks (DBN), Neural Network (NN) and Expert System (ES) respectively. The ES&NN-WP prediction model is also applied as a vital part of the FP-TBSLB sub-system to suggest a set of under-loaded VMs to execute extra workload of Poorly Performed VMs (PPVMs), or determine under-loaded PMs as new host for PPVMs. These models are presented in Chapter 4.

3.2.2 MULTI-OBJECTIVE TASK SCHEDULING SUB-SYSTEM

System virtualization in cloud computing provides low-cost, flexible and powerful executing environment for virtualized data centers, which plays an important role in the infrastructure of Cloud computing. However, the virtualization also brings some challenges, particularly to the resource management and task scheduling (Kong et al. 2011). Cloud online users submit their jobs (Directed Acyclic Graph (DAG) of tasks) anytime and anywhere to dynamic resources. In distributed computing environment of cloud, task arrival and execution processes are stochastic (Tong et al. 2014). The arrival tasks accumulate in several queues to be sent to the task schedulers. Then, the task schedulers allocate these tasks to computing nodes or VMs for execution. The main concern in dynamic scheduling is to adapt to the consequent uncertainties, and scheduling tasks with shortest response time and optimal resource utilization (Tong et al. 2014).

To address this problem, an efficient dynamic Multi-Objective Task Scheduling optimization model by applying Particle Swarm Optimization and Genetic Algorithms (MOTS-PSO/GA) for virtualized data centers is proposed in Chapter 5. This model is designed to schedule different types of tasks related to SaaS and PaaS over heterogeneous

VMs with different amounts of allocated virtual sources in a cloud cluster. This optimization model aims to reduce service response time and cost while increasing QoS. The MOTS sub-system is also applied as part of the FP-TBSLB sub-system to find an optimal solution to the scheduling of tasks from poorly performing VMs to a set of under-loaded VMs.

3.2.3 Fuzzy Predictable Task Based System Load Balancing Sub-system

A key enabling technology of cloud systems is server virtualization which allows decoupling applications and services from the physical server infrastructure. Server virtualization makes it possible to execute concurrently several VMs on top of a single PM/server, each VM hosting a complete software stack (operating system, middleware, applications) and being given a partition of the underlying resource capacity (i.e. CPU power and RAM size) (see Figure 3.2). On top of that, the live migration capability of hypervisors allows the migration of a VM from one physical host to another with no or little interruption of service (Van, Tran & Menaud 2009).

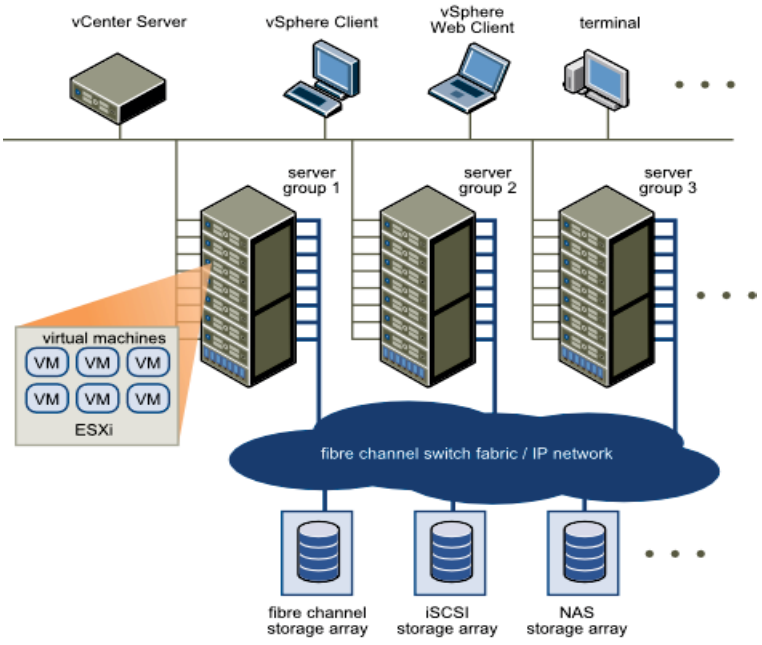


Figure 3.2: Physical topology of vSphere datacenter (vmware 2015)

In such environment of a cloud cluster, imbalanced load occur in two circumstances: (1) some VMs located on a PM are poorly performing while the others are under-loaded, (2) all VMs that are located on a PM are poorly performing (overloaded) and fully utilizing the resources of their host PM that leads this PM to become overloaded as well. In the first situation, additional resources (CPU/memory) will be allocated to PPVMs to scale them up and enhance their performance, if the original host PM has idle resources available. Otherwise, these VMs will be migrated to another PM using suspend/resume or live migration strategy for stronger computation power, larger memory, fast communication capability, or energy savings (Jin et al. 2011). To solve the imbalance load in the second situation, a number of PPVMs are migrated from an overloaded PM to an under-loaded PM. If a PPVM is small, it would be reasonable to migrate it to a new physical host (see Figure 3.3). However, VM migration is not an optimal solution when the PPVM is large. This results in dirty memory due to the pre-copy process, utilizes a large amount of memory in the primary and new host PMs, causes the VM to slow down during migration process, carries the risk of losing the most recent customer activities and is cost- and time-consuming. The resume/suspend migration strategy not only has live migration shortcomings, but also causes a long VM downtime. To overcome these drawbacks and improve the current load balancing approaches which rely on VM migration, a Fuzzy Predictable Tasks Based System Load Balancing (FP-TBSLB) approach is proposed that transfers tasks from a PPVM to a new compatible VM instead of migrating the PPVM. The FP-TBSLB sub-system is explained in detail in Chapter 6.

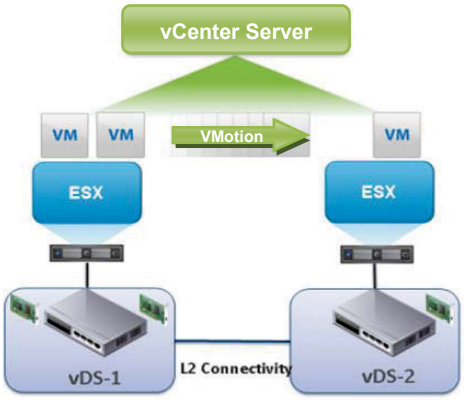


Figure 3.3: VM migration (Schwoebel 2015)

3.3 SUMMARY

In this chapter a framework of Autonomic System for Optimal Resource Management (AS-ORM) in cloud environments is presented. This system can be applied in the hypervisor layer of cloud infrastructure to reduce the time and cost taken to deliver cloud services. This gives cloud providers the opportunity to enhance the QoS with minimum cost and power consumption. The AS-ORM sub-systems that are briefly introduced in this chapter are elaborated on in Chapters 4, 5 and 6.

Chapter 4.

VIRTUAL MACHINE WORKLOAD PREDICTION SUB-SYSTEM

4.1 INTRODUCTION

The success of cloud services depends critically on the effective management of virtualized resources to match them with alterable customer demands. This chapter aims to propose and implement two prediction models to estimate VM workload and CPU usage. These models give the providers the opportunity to handle uncertainties in resource management from the cloud provider perspective that enables underlying complexity, automates resource provisioning and controls client-perceived quality of service. The first Workload Prediction (WP) model is designed by applying the combination of Neural Network (NN) and Expert System (ES) techniques that is named ES&NN-WP. The second prediction model relies upon a Dynamic Bayesian Network (DBN) called DBN-WP. These models are able to determine the current status of a cloud infrastructure, including physical and VMs, and to predict the near future state. This helps the hypervisor to make a proper decision for optimal physical resource allocation to VMs and consequently reduce execution time and meet quality of service requirements.

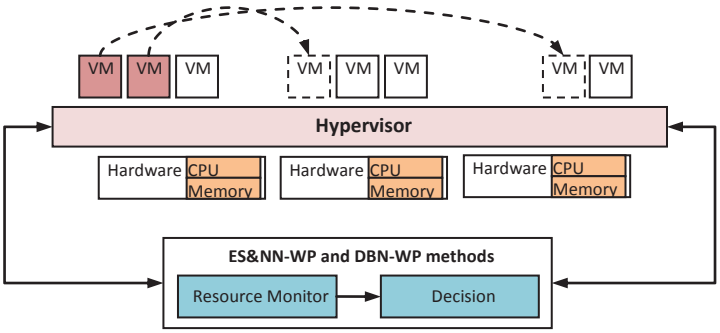


Figure 4.1: The workload prediction methods in relation to other clouds components

4.2 BACKGROUND

This section explains the important background information that is applied to develop ES&NN-WP and DBN-WP models including fuzzy logic, Bayesian networks, neural networks and expert systems.

4.2.1 FUZZY SETS AND EXPERT SYSTEMS

Fuzzy logic is a concept to deal with uncertainty, vagueness, or imprecise problems that uses membership functions with values between 0 and 1. Fuzzy set theory, which is based on fuzzy logic, was first proposed by Zadeh in 1965. In fuzzy set theory unlike conventional set theory based on Boolean logic, a particular object or variable has a degree of membership in a given set that may be anywhere in the range of 0 (completely not in the set) to 1 (completely in the set) (Zadeh 1965).

4.2.1.1 FUZZY SETS AND FUZZY NUMBERS

Definition 1 (Fuzzy set): Fuzzy set A is defined in terms of a universal set X by a membership function that assigns to each element $x \in X$ a value $\mu_A(x)$ in the interval [0,1], i.e. $A: X \rightarrow [0,1]$ (Zadeh 1965).

Definition 2 (Support of a fuzzy set):The support of a fuzzy set A ($supp(A)$) in the universe of discourse X is a crisp set that contains all the elements of X that have nonzero membership values in A, that is:

$$supp(A) = \{x \in X | \mu_A(x) > 0\} \quad (4.1)$$

Definition 3 (α-cut): Let A be a fuzzy set in the universe X, $\alpha \in (0,1]$. The α-cut or α-level set of the fuzzy set A is the crisp set A_α defined by (Shapiro 2009):

$$A_\alpha = \{x \in X | \mu_A(x) \geq \alpha\} \quad (4.2)$$

Figure 4.2 shows an example of an α-cut in which the domain under consideration is limited to a set of elements with a degree of membership of at least alpha. The support of fuzzy set A is all x such that $\mu_A(x) > 0$, and its α-cut is from X_{left}^{α} to X_{right}^{α}. Values outside the interval are considered as insignificant values that should be excluded from consideration i.e. they are cut out.

Definition 4 (Fuzzy number): A fuzzy set A in $\mathbb{R}$ satisfies the following conditions (Naderpour 2015; Shapiro 2009):

- A is normal i.e. there is at least one point $x \in \mathbb{R}$ with $\mu_A(x) = 1$
- A_α is a closed interval for every $\alpha \in (0,1]$
- The support of A is bounded

Figure 4.2 represents the general characteristic of a fuzzy number A where $\mu_A(x)$ denotes the membership function of x in the fuzzy set. This shape of fuzzy number is referred to as a "triangular" fuzzy number, and is denoted by the triple (a_1, a_2, a_3).

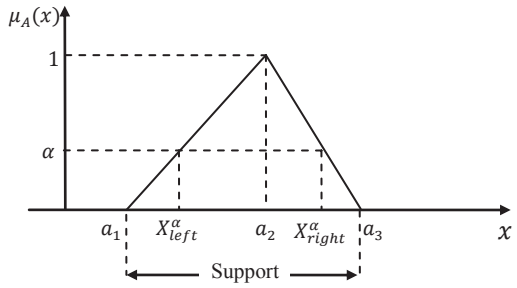


Figure 4.2: A fuzzy number

Definition 5 (Fuzzy random variable): Let $(\Omega, \mathcal{F}, P)$ be a probability space, $F(\mathbb{R})$ the set of fuzzy numbers in $\mathbb{R}$ with compact supports and W is a mapping $\Omega \to F(\mathbb{R})$. Then W is a fuzzy random variable if and only if given $\omega \in \Omega$, $W_\alpha(\omega)$ is a random interval for any $\alpha \in [0,1]$ where $W_\alpha(\omega)$ is a α-level set of the fuzzy set $W(\omega)$ (Kwakernaak 1978).

Note: A subset of Euclidean space $\mathbb{R}^n$ is called compact if it is closed and bounded. For example, in $\mathbb{R}$, the closed unit interval $[0,1]$ is compact.

Definition 6 (Fuzzy state): Let the crisp state set $\mathcal{P}$ consist of the states $p_1, p_2, \ldots, p_n$. Then, each fuzzy state can be written as a vector $q = [q_1, q_2, \ldots, q_n]$, where $q_i \in [0,1]$. This way, each fuzzy state can be considered as a possibility distribution or alternatively as a fuzzy set $q \in \mathcal{F}(\mathcal{P})$, ($\mathcal{F}(\mathcal{P})$ the set of all fuzzy subsets defined for $\mathcal{P}$) determining the degree q_i by which the system participates in each crisp state p_i, provided it is in the current fuzzy state q (Schmidt & Boutalis 2012).

Definition 7 (Linguistic variable): Linguistic variable is a variable whose values are words or sentences in a natural or artificial language (Zadeh 1975). Linguistic variable is characterized by (X, T, U, M) where:

- X is the name of the linguistic variable (Weather temperature)
- T is the set of linguistic values that X can take (T = {Cold, Pleasant, Hot})
- U is the actual physical domain in which the linguistic variable X takes its quantitative (crisp) values (U = [−20, 40] °C).
- M is a semantic rule that relates each linguistic value in T with a fuzzy set in U; (M relates 'Cold', 'Pleasant', and 'Hot' with the membership functions shown in Figure 4.3).

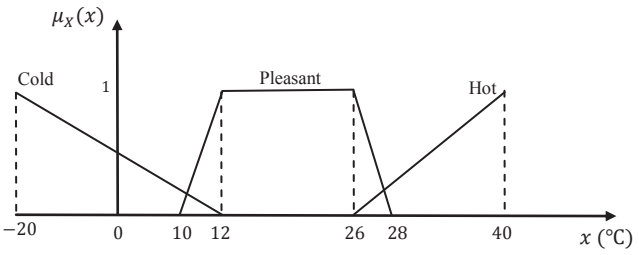


Figure 4.3: Membership function of weather temperature

Let x, y be linguistic variables in the physical domains U, V, and A, B be fuzzy sets in U, V:

- Connective 'And' x is A and y is B use fuzzy intersection:

- $A \cap B \in U * V: \mu_{A \cap B}(x, y) = t[\mu_A(x), \mu_B(y)]$
- $t: [0,1] * [0,1] \rightarrow [0,1]$ is any *t*-norm that for $x, y, z \in [0,1]$satisfies the following four axioms (Wang 1999):
  - $t(x, y) = t(y, x)$(commutativity),
  - $t(x, t(y, z)) = t(t(x, y), z)$ (associativity),
  - $t(x, y) \leq t(x, z)$whenever $y \leq z$ (monotonicity),
  - $t(x, 1) = x$ (boundary condition).

- Connective 'Or' *x* is *A* or *y* is *B* use fuzzy union:
  - $A \cup B \in U * V: \mu_{A \cup B}(x, y) = s[\mu_A(x), \mu_B(y)]$
  - $s: [0,1] * [0,1] \rightarrow [0,1]$ is any *s*-norm that for $x, y, z \in [0,1]$satisfies the following four axioms (Wang 1999):
    - $s(1,1) = 1,\ s(0, x) = s(x, 0)$ (boundary condition),
    - $s(x, y) = s(y, x)$ (commutative),
    - If $x \leq x'$and $y \leq y'$then $s(x, y) \leq s(x', y')$ (nondecreasing),
    - $s(s(x, y), z) = s\big(x, s(y, z)\big)$ (associative).
- Connective 'Not' *x* is not *A* use fuzzy complements

4.2.1.2 FUZZY EXPERT SYSTEMS

The basic idea behind Expert Systems (ES) is simply that expertise, which is the vast body of task-specific knowledge, is transferred from a human to a computer. Then like a human consultant, it gives advice and explains, if necessary, the logic behind the advice. Fuzzy logic is applied to design the ES to mathematically emulate human reasoning and provide an intuitive way of designing function blocks for intelligent systems. Fuzzy logic allows an operator to express his/her knowledge in the form of related imprecise inputs and outputs in terms of linguistic variables, which simplifies knowledge acquisition and representation, and the knowledge obtained is easy to understand and modify (Naderpour, Lu & Zhang 2014b).

A fuzzy ES as shown in Figure 4.4 includes three parts: fuzzification, fuzzy inference engine and de-fuzzification. In the fuzzification process, the fuzzy sets are formed for all input variables. The fuzzy inference engine takes into account the input variables and the logic relations/rules between them, and uses fuzzy logic operations to generate the output. In the defuzzification process, the output fuzzy set is converted into a crisp value (Markowski et al. 2011).

A rule contains information obtained from a human expert, and represents that information in the form of IF–THEN. The rule can then be used to perform operations on data to inference in order to reach appropriate conclusion. These inferences are essentially a computer program that provides a methodology for reasoning about information in the rule base or knowledge base, and for formulating conclusions (Naderpour & Lu 2012a; Naderpour & Lu 2012b; Shu-Hsien 2005).

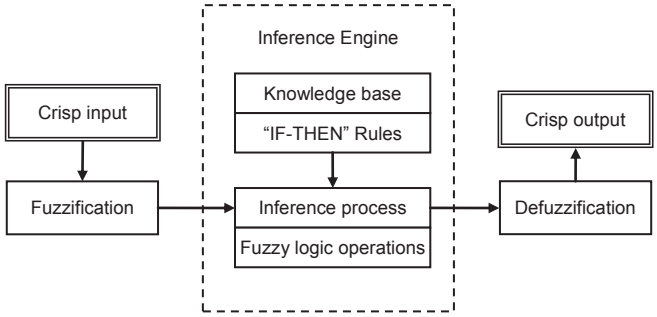


Figure 4.4: A fuzzy expert system

There are several inference methods, however, Mamdani (1977) and Takagi and Sugeno (1985) methods are most commonly used in industrial and fuzzy software tools. The characteristic of Mamdani's model also known as the Max-Min fuzzy rule based inference are presented in Table 4.1.

Table 4.1: Characteristics of the Mamdani model

Operation	Operator	Formula
Union (OR)	MAX	$\mu_C(x) = max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x)$
Intersection (AND)	MIN	$\mu_C(x) = min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x)$
Implication	MIN	$min(\mu_A(x), \mu_B(x))$
Aggregation	MAX	$max(min(\mu_A(x), \mu_B(x)))$
Defuzzification	CENTROID	$COA = Z^* = \frac{\int z\mu_C(z)\, dz}{\int \mu_C(z)\, dz}$

Note:
$\mu_C(x)$ = value of the resultant membership function.
$\mu_A(x)$ = value of the membership function where the input belongs to the fuzzy set A.
z = abscissa value, ($\mu_C(z)$ is the ordinate).

In summary, Figure 4.5 shows a Mamdani fuzzy inference system with two rules. It fuzzifies the two inputs by finding the intersection of the crisp input value with the input membership function. It uses the minimum operator to compute the fuzzy AND for combining the two fuzzified inputs to obtain a rule strength. It clips the output membership function at the rule strength. Finally, it uses the maximum operator to compute the fuzzy OR for combining the outputs of the two rules.

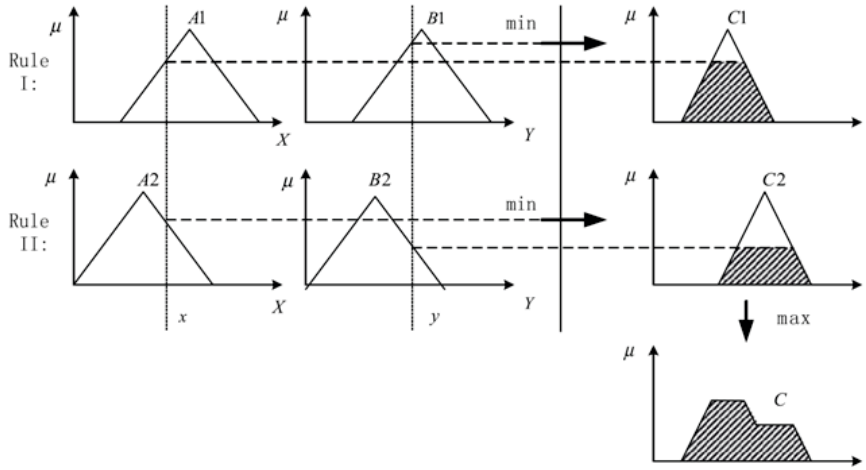


Figure 4.5: Mamdani fuzzy inference system for two inputs and single output

The Sugeno fuzzy inference method is quite similar to the Mamdani's; however, the difference is that the output consequence is not computed by clipping an output membership function at the rule strength. In fact, in the Sugeno's model there is no output membership function at all. Instead the output is a crisp number computed by multiplying each input by a constant and then adding up the results.

4.2.2 BAYESIAN NETWORKS

A BN is defined as a couple: $\mathcal{G} = ((N, A), \mathcal{P})$, where (N,A) represents the graph; N is a set of nodes; A is a set of arcs; $\mathcal{P}$ represents the set of probability distributions that are associated with each node. When a node is not a root node, the distribution is a conditional probability distribution that quantifies the probabilistic dependency between that node and its parents (Weber & Jouffe 2006). A discrete random variable X is represented by a node $n \in N$ with a finite number of mutually exclusive states. States are defined on $S_n: \{s_1^n, s_2^n, \dots, s_M^n\}$. The set $\mathcal{P}$ is represented with Conditional Probability Tables (CPT), and each node has an associated CPT. For instance, if n_i is a parent of n_j, and the nodes n_i and n_j are defined over the sets $S_{n_i}: \{s_1^{n_i}, s_2^{n_i}, \dots, s_M^{n_i}\}$ and $S_{n_j}: \{s_1^{n_j}, s_2^{n_j}, \dots, s_L^{n_j}\}$, the CPT of n_j is then defined as a matrix by the conditional probabilities $P(n_j|n_i)$ over each n_j state knowing its parents states (n_i) (Naderpour & Lu 2014):

$$P\left(n_j \middle| pa(n_j)\right) = \begin{bmatrix} p\left(n_j = s_1^{n_j} \middle| n_i = s_1^{n_i}\right) & \dots & p\left(n_j = s_L^{n_j} \middle| n_i = s_1^{n_i}\right) \\ \vdots & \vdots & \vdots \\ p\left(n_j = s_1^{n_j} \middle| n_i = s_M^{n_i}\right) & \cdots & p\left(n_j = s_L^{n_j} \middle| n_i = s_M^{n_i}\right) \end{bmatrix} \quad (4.3)$$

Various inference algorithms can be used to compute marginal probabilities for each unobserved node, given information on the states of a set of observed nodes; the junction tree algorithm is a classic example of such an algorithm. Inference in BN then allows us to take into account any state variable observation (an event) to update the probabilities of the other variables. When observations are given, this knowledge is integrated into the network and all the probabilities are updated accordingly. Hard evidence of the random variable X indicates that the state of the node $n \in N$ is one of the states $S_n: \{s_1^n, s_2^n, \dots, s_M^n\}$.

Nevertheless, when this knowledge is uncertain, soft evidence can be used. Soft evidence for a node n is defined as evidence that enables the updating of the prior probability values for the states of n (Naderpour 2015).

A DBN is a BN that includes a temporal dimension. This new dimension is managed by time-indexed random variables X_n which are represented at time step k by a node $n_{(i,k)} \in N$ with a finite number of states $S_{n_i}: \{s_1^{n_i}, s_2^{n_i}, \dots, s_M^{n_i}\}$. Several time stages are represented by several sets of nodes and an arc that links two variables belonging to different time slices represents a temporal probabilistic dependence between these variables. DBNs then allow us to model random variables and their impact on the future distribution of other variables. Defining this impact as transition probabilities between the states of the variable at time step $k-1$ and those at time step k leads to the definition of CPTs that are relative to inter-time slices. With this model, the future slice (k) is conditionally independent of the past given the present ($k-1$) (Naderpour, Lu & Zhang 2014a, 2014c).

4.2.3 Neural Networks

A neural network (NN) consists of a number of layers: the input layer has a number of input neurons $X = \{x_1, \dots, x_m\}$; the output layer has one or more output neurons $O = \{o_1, \dots, o_n\}$ and several hidden layers with hidden neurons $H = \{h_1, \dots, h_l\}$ in between. In a fully-connected NN, the neurons at each layer are connected to the neurons of the next layer; these connections are known as synapses. Each synapse is associated with a weight, which is to be determined during training.

During the training phase, the NN is fed with input vectors and random weights are assigned to the synapses. After presentation of each input vector, the network generates a predicted output $\hat{y}$. The generated output is then compared with the actual output y; the difference between the two is known as the error term which is then used as a feedback to correct the synaptic weights of the network. Since the error value guides the network towards convergence to the target output for a set of real-world inputs, therefore it is called a feed forward or error correction network. The rate at which the weight is updated is

called the learning rate ρ, which is generally a value within the range [0, 1]. The training of the NN continues until a specific criterion is met, e.g. the sum of squared errors falls below a certain threshold. A typical three-layer neural network is shown in Figure 4.6 (Islam et al. 2012b).

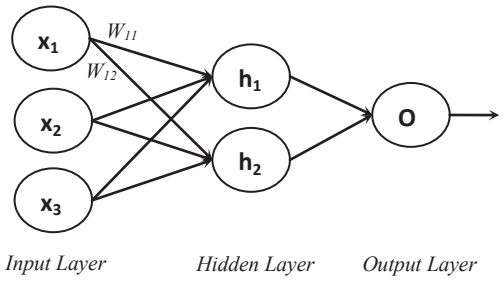


Figure 4.6: Three layer neural network

4.3 A VM WORKLOAD PREDICTION SUB-SYSTEM

Resource provisioning in cloud environments often requires an estimate of the capacity needs of VMs. The estimated VM size and its resource utilization are the basis for allocating resources (i.e. CPU/memory) commensurate with demand (Meng et al. 2010). Most of the existing researches in the area of VM workload prediction have applied prediction methods such as neural networks, pattern recognition and linear regression to forecast the workload of VMs or their CPU usage in the cloud environment. These methods predict the future workload of VMs by applying their previous workload patterns in time slot *t*, determined on the basis of related historical data (Yang et al. 2013). They are mostly suggested for resource prediction or VM remapping, and are applied for SaaS and PaaS, where cloud providers are aware of application and software types and can trace their behavior. However, for IaaS (where IaaS is not delivered to PaaS and SaaS providers), there is no information about upcoming executing applications on each VM. In these cases, application behavior is not applicable in estimating VM workload and CPU usage. In addition, VM workload is affected by user behaviors and users' sudden decisions, so fluctuations in the workload could be independent of the previous workload and CPU

usage pattern, and could change dramatically on the basis of dynamic, unpredictable and fluctuating resource demands by users (see Figure 4.7).

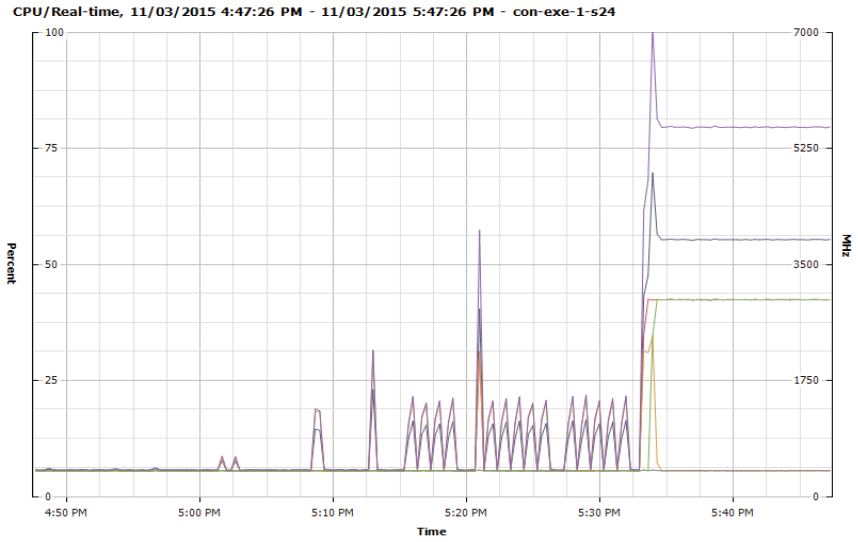


Figure 4.7: CPU usage trend of a VM in the cloud cluster

Considering these facts, two fuzzy workload prediction methods called ES&NN-WP and DBN-WP are proposed. ES&NN-WP not only applies NN to predict VM CPU usage patterns using historical data, but also applies a fuzzy ES to control near future changes in CPU usage and workload for every VM that is applied to deliver SaaS, PaaS or IaaS. DBN-WP is a probabilistic prediction method that is developed based on DBN technique and is also applied in AS-ORM to estimate the overloading probability of virtual and physical resources in a cloud environment. Based on the results of the WP methods, the upcoming poorly performing VMs are determined, and consequently the PM hotspots are predicted. In addition, these prediction models have the ability to forecast a set of compatible VMs (i.e. the VMs with the same OS as the poorly performing VM and have the required applications installed) to execute the extra workload of the poorly performing VM. Using these models, the new destination VMs can be chosen from a set of VMs that are delivered as IaaS, or other VMs in a cluster that are executing SaaS or PaaS.

4.4 THE ES&NN-BASED WORKLOAD PREDICTION MODEL

To design the ES&NN-WP model, first the conditions under which a VM might be overloaded (poorly performing) in the near future are determined. Then, the corresponding input/output variables of the ES&NN-WP algorithm are defined on the basis of the conditions thus determined. These conditions and variables are applied to extract the ES rules. The ES&NN-WP is run every two minutes to control and forecast the VM workload situation based on the historical CPU usage data of the VM, and current changes in usage.

4.4.1 THE POORLY PERFORMING VM: CONDITIONS AND VARIABLES

Considering the fluctuation in CPU usage and the fact that it might increase suddenly and decrease soon after, checking CPU usage frequently at specific times, as applied in (Islam et al. 2012a), is not a reliable means of estimating a VM's upcoming CPU usage and workload. Therefore, the ES&NN-WP method monitors the CPU usage trends and fluctuations over a small period of time (e.g., every two minutes) to forecast the VM's workload level for the next interval. Four definitions are determined based on expert's knowledge and literature to explain a poorly performing (overloaded) VM based on the VM's CPU usage and its workload, and are elaborated below, where *ct* is the current time, *t* is a given period of time (e.g., two minutes), and time slot $Ts = \{ct - t, ct\}$:

Definition 1: If VM_{Ucpu}^{k} is the total amount of CPU utilization of VM_k, this VM will be overloaded if:

$$\lim_{s \to ct} VM_{Ucpu}^{k}(s) \geq 80\% , s \in Ts \tag{4.4}$$

Based on this definition, VM_k has the potential to be overloaded under the following conditions:

Condition 1.1: The CPUs allocated to VM_k stay busy during the last minutes of time slot Ts,

To check this condition, the current value of CPU utilization by VM_k, i.e. $VM^k_{Ucpu}(ct)$, if firstly determined. If $VM^k_{Ucpu}(ct) \geq 80\%$, then the average value of VM^k_{Ucpu} in the last part of time slot Ts (i.e. lt) is calculated. Based on these assumptions, VM_k will probably be overloaded if:

$$VM^k_{Ucpu}(ct) \geq 80\% \ \& \ \underset{s_i \in lt}{\text{Avg}} VM^k_{Ucpu}(s_i) \geq 80\%, \quad lt \subset Ts \tag{4.5}$$

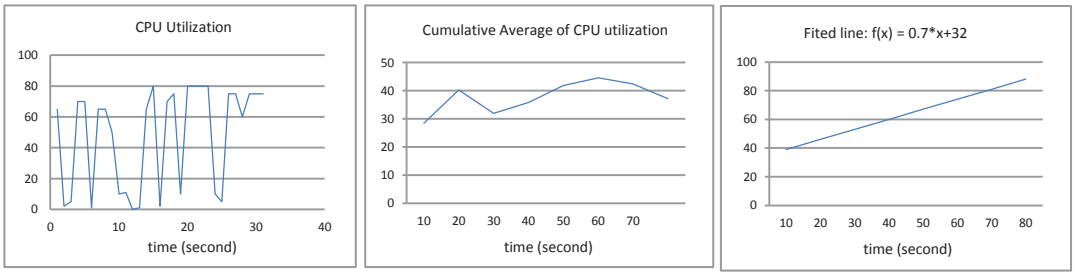
Condition 1.2: The cumulative average of CPU usage of VM_k (calculated every 20 seconds) has an increasing trend.

The CPU usage usually fluctuates dramatically and it is difficult to estimate its overall increasing or decreasing trend (Lopez & Minana 2012) (see Figure 4.8a). Therefore, the cumulative average of CPU usage is used to estimate its trend during time slot Ts as follows:

$$CA^k_{cpu}(x) = \sum_{i=1}^{x*20} \frac{VM^k_{Ucpu}(s_i)}{x * 20}, x \in \left\{1,2,3,..,div(\frac{Ts}{20})\right\} \tag{4.6}$$

where div is the integer division (see Figure 4.8b). The Polynomial fitting tool in MATLAB is then applied to determine the overall trend of $CA^k_{cpu}(x)$ (see Figure 4.8c). $CA^k_{cpu}(x)$ has an increasing trend if the derivative of its fitted line ($fCA^k_{cpu}(x)$) is positive in Ts i.e.:

$$fCA'^k_{cpu}(x) \geq 0 \ , x \in \left\{1,2,3,..,div(\frac{Ts}{20})\right\} \tag{4.7}$$



a **b** **c**

Figure 4.8: Estimating CPU utilization trend in time slot Ts

Fuzzy Variable 1: Variable $VM^k_{Huti}(Ts)$ is defined based on VM_k utilization to control whether or not conditions 1 and 2 are satisfied:

$$VM^k_{Huti}(Ts) =$$
$$\begin{cases} H \ \ [VM^k_{Ucpu}(ct) \geq 80\%] \text{ and } \left[\underset{s_i \in lt}{\text{Avg}}\, VM^k_{Ucpu}(s_i) \geq 80\%\right] \text{ and } \left[fCA'^k_{cpu}(x) \geq 0\right] \\ L \ \ \text{Otherwise} \end{cases} \quad (4.8)$$

Definition 2: If $VM^k_{et}(s)$ are the number of scheduled tasks to VM_k as a time series in time slot *Ts*, then the VM_k will be overloaded if the time series $VM^k_{et}(s)$ has an increasing trend.

Condition 2.1: The cumulative average of time series $VM^k_{et}(s)$ that shows the overall trend of $VM^k_{et}(s)$ has an increasing trend.

The cumulative average of $VM^k_{et}(s)$ is calculated every 20 seconds as follows:

$$CA^k_{et}(x) = \sum_{i=1}^{x*20} \frac{VM^k_{et}(s_i)}{x * 20}, x \in \left\{1,2,3,..,div(\frac{Ts}{20})\right\} \quad (4.9)$$

where div is the integer division. $CA^k_{et}(x)$ has an increasing trend if its Polynomial fitted line has a positive derivative in Ts i.e.:

$$fCA'^k_{et}(x) \geq 0\ , x \in \left\{1,2,3,..,div(\frac{Ts}{20})\right\} \quad (4.10)$$

Fuzzy Variable 2: Variable $VM^k_{RisEt}(Ts)$ is defined as follows based on Condition 3 to show whether or not the status of $VM^k_{et}(s)$ will lead to VM_k being overloaded.

$$VM^k_{RisEt}(Ts) = \begin{cases} H & fCA'^k_{et}(x) \geq 0 \\ L & \text{Otherwise} \end{cases} \quad (4.11)$$

Definition 3: Rao (2011) proposed a metric called Productivity Index (*PI*) to measure the system processing capability. He defined *PI* as:

$$PI^k(s) = \frac{CW_k(s)}{CC_k(s)}, s \subset Ts \quad (4.12)$$

where $CW_k(s_i)$ is the number of completed tasks and $CC_k(s)$ isthe amount of resource consumed (CPU utilization) during the time slot *Ts* by VM_k. According to Rao's definition, VM_k will be overloaded if *PI* begins to drop —although Rao believes that for online

identification, the single *PI* metric is not enough to identify the system state because any change in *PI* can be due to either the system capacity or the input load change. Condition 3.1 is suggested to determine *PI* trend and monitor a VM's workload during *Ts* with a constant amount of resources.

Condition 3.1: The time series $PI^k(s)$ has a decreasing trend.

Fuzzy Variable 3: Variable $PI^k_{Dec}(Ts)$ is defined to indicate the increasing or decreasing trend of $PI^k(s)$ during *Ts* as follows:

$$PI^k_{Dec}(Ts) = \begin{cases} H & fPI'^k(s) \leq 0, \quad s \in Ts \\ L & \text{Otherwise} \end{cases} \tag{4.13}$$

Where $fPI^k(s)$ is the Polynomial fitted line of $PI^k(s)$.

Definition 4: The workload status of VM_k is estimated based on the CPU usage prediction result of a designed NN.

In this method, a three-layer feed-forward NN is designed and then trained based on the historic data of the CPU usage of VM_k to predict its CPU utilization pattern. It has an input layer with three neurons $I = \{VM^k_{Ucpu}, VMm_k, VMc_k\}$, and a hidden layer with two hidden neurons $H = \{h_1, h_2\}$ in between. The output layer of the NN $(O = NN^k_R(Ts))$ has one neuron that represents its prediction results for upcoming CPU usage by VM_k. The activation functions that are applied in the designed NN are Sigmoidals in the hidden layer and Linears in the output.

Condition 4.1: The prediction result of the designed NN indicates that VM_k will be overloaded.

Fuzzy Variable 4: Variable $NN^k_R(Ts)$ is defined to show NN prediction results as:

$$NN^k_R(Ts) = \begin{cases} O & VM_k \text{is overloaded} \\ U & VM_k \text{is under} - \text{loaded} \end{cases} \tag{4.14}$$

However, none of these conditions by itself can indicate risk unless it happens when some of other conditions are satisfied as well. Therefore, the combination of fuzzy variables 1-4 is

applied by ES as inputs to estimate the value of the output variable, which is defined as follows:

Output Fuzzy Variable: The variable VM_{load}^{k} is defined to show the predicted VM_kworkload situation by ES as:

$$VM_{load}^{k}\,(Ts) = \begin{cases} O & VM_k \text{ is likely to be overloaded} \\ N & \text{Neutral condition} \\ U & VM_k \text{is likely to be under} - \text{loaded} \end{cases} \quad (4.15)$$

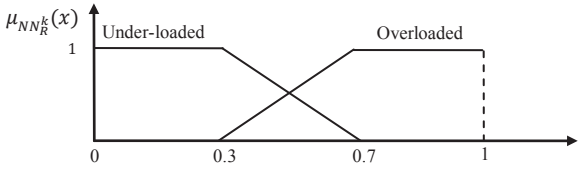
In some situations, the ES answer is neutral and does not suggest an obvious result. In such cases, the workload situation of VM_k will be forecasted through the next prediction round. The defined variables are summarized in Table 4.2, and the membership functions of these fuzzy variables are explained in detail in Table 4.3, and illustrated in Figure 4.9.

Table 4.2: The ES&NN-WP variables

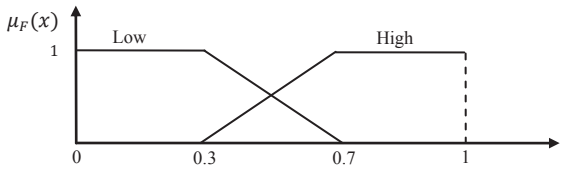
Symbol	Definition
$CW_k(Ts)$	The amount of work completed during the time slot *Ts* by VM_k
$CC_k(Ts)$	The amount of resource (CPU) consumed during the time slot *Ts* by VM_k
VM_{load}^{k}	VM_k workload status
VM_{Ucpu}^{k}	The amount of CPU utilization by VM_k
$CA_{cpu}^{k}\,(x)$	The cumulative average of time series $VM_{Ucpu}^{k}(si)$ in every 20 seconds
$fCA'^{k}_{cpu}\,(x)$	The derivative of fitted line to $CA_{cpu}^{k}\,(x)$
$VM_{et}^{k}(s_i)$	The number of executing tasks in the VM_k at time s_i
$CA_{et}^{k}\,(x)$	The cumulative average of time series $VM_{et}^{k}(s_i)$ in every 20 seconds
$fCA'^{k}_{et}\,(x)$	The derivative of the Polynomial fitted line to $CA_{et}^{k}\,(x)$
$VM_{RisEt}^{k}(Ts)$	Shows VM_k's workload has had an increasing or decreasing trend during *Ts*
$VM_{Huti}^{k}(Ts)$	Shows CPU utilization of VM_k has had an increasing or decreasing trend during *Ts*
$PI^{k}(s)$	The productivity index for VM_k during *Ts*
$PI_{Dec}^{k}(Ts)$	Shows $PI^{k}(s)$ has had an increasing or decreasing trend during *Ts*
$NN_{R}^{k}(Ts)$	The neural network forecasting results for CPU usage by VM_k during *Ts*

Table 4.3: Fuzzification of variables

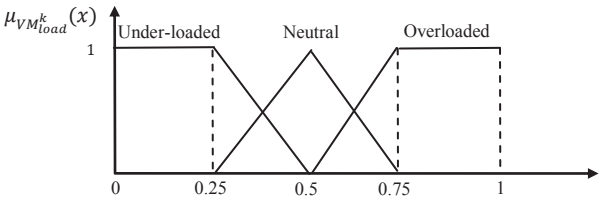
Input Variable: $NN_R^k(Ts)$			
Set	**Linguistic term**	**α level cuts**	
		1–level cut	**0–level cut**
U	Under-loaded	0,0.3	0.7
O	Overloaded	0.7,1	0.3
Universe of discourse: (0,1)			
Input Variables: $VM_{Huti}^k(Ts), VM_{RisEt}^k(Ts)$, and $PI_{Dec}^k(Ts)$			
Set	**Linguistic term**	**α level cuts**	
		1–level cut	**0–level cut**
H	Low	0,0.3	0.7
L	High	0.7,1	0.3
Universe of discourse: (0,1)			
Output Variable: $VM_{load}^k(Ts)$			
Set	**Linguistic term**	**α level cuts**	
		1–level cut	**0–level cut**
U	Under-loaded	0,0.25	0.5
N	Neutral	0.5	0.25, 0.75
O	Overloaded	0.75,1	0.5
Universe of discourse: (0,1)			



a. The Membership Function of $NN_R^k(Ts)$



b. The Membership Function of $VM_{Huti}^k(Ts)$, $VM_{RisEt}^k(Ts)$, and $PI_{Dec}^k(Ts)$



c. The Membership Function of $VM_{load}^k(Ts)$

Figure 4.9: The membership functions of input and output variables

4.4.2 THE POORLY PERFORMING VM: RULES

The ES rules are determined based on the aforementioned conditions and related variables to monitor VM CPU usage and workload changes, and predict which VM is likely to highly utilize its allocated resources and become overloaded (exhibit poor performance). 48 ($2^4 * 3$) rules can be extracted by different combinations of four input variables ($VM^k_{Huti}(Ts), VM^k_{RisEt}(Ts), PI^k_{Dec}(Ts)$ and $NN^k_R(Ts)$) and the output variable $VM^k_{load}(Ts)$. In this study, eleven main rules as summarized in Table 4.4 are selected based on expert's knowledge and applied to control the system.

Table 4.4: ES rules

Based on Historical Data		**Based on Actual Data in the past specified minutes**						**VM workload situation**
		Definition 1		**Definition 2**		**Definition 3**		
$NN^k_R(Ts)$		$VM^k_{Huti}(Ts)$		$VM^k_{RisEt}(Ts)$		$PI^k_{Dec}(Ts)$		VM^k_{load}
O	and	H	and	H	and	H	then	Overloaded
O	and	H	and	L	and	H	then	Overloaded
O	and	H	and	H	and	L	then	Overloaded
O	and	L	and	H	and	H	then	Overloaded
U	and	H	and	H	and	H	then	Overloaded
U	and	H	and	H	and	L	then	Overloaded
U	and	L	and	H	and	L	then	Neutral
U	and	H	and	L	and	L	then	Neutral
U	and	L	and	L	and	H	then	Under-loaded
O	and	L	and	L	and	L	then	Under-loaded
U	and	L	and	L	and	L	then	Under-loaded

4.4.3 THE ES&NN-WP ALGORITHM

The ES&NN-WP algorithm —that is responsible for predicting VMs with over- and under-utilization based on their CPU usage and workload status— is summarized as follows. This algorithm plays a key role in AS-ORM. ES&NN-WP is also applied in the FP-TBSLB sub-system to determine the origin and destination VMs for extra workload in a cloud cluster.

Algorithm 4.1: The ES&NN-WP Algorithm

Input: All variables in Table 4.2.

[Begin ES&NN-WP algorithm]

6.1. Calculate the value of following variables:

- $VM^k_{Ucpu}(ct)$ and $VM^k_{Ucpu}(s_r)$
- $CA^k_{cpu}(x)$ and $fCA'^k_{cpu}(x)$
- $VM^k_{Huti}(Ts)$
- $CA^k_{et}(x)$ and $fCA'^k_{et}(x)$
- $VM^k_{RisEt}(Ts)$
- $PI^k_{Dec}(Ts)$ and $fPI'^k(s)$

6.2. Calculate NN results about VM workload situation $(NN^k_R(Ts))$

6.3. Determine the value of $VM_{load}(Ts)$ for each VM based on ES rules, then forecast:

- Poorly performing VMs
- A set of compatible VMs as new destination for extra workload imposed on the poorly performing VMs

[End ES&NN-WP algorithm]

4.5 The DBN-Based Workload Prediction Model

Probability theory is the method of choice for dealing with uncertainty in many science and engineering disciplines. As a probability theory tool, the Bayesian Network (BN) model of a system is the compact representation of a joint probability distribution of the variables comprising the system (Oztekin 2009). In this sense, BNs can be used for building representative models of cloud resource management systems. This paper develops a BN-based predictive module to provide decision support for the virtualization layer to handle uncertainties in scaling utility computing resources up or down based on SLA parameters. As the module contains continuous variables such as CPU and memory utilizations, therefore, fuzzy BNs are relied on to complement probability theory with fuzzy sets to perform exact inferencing.

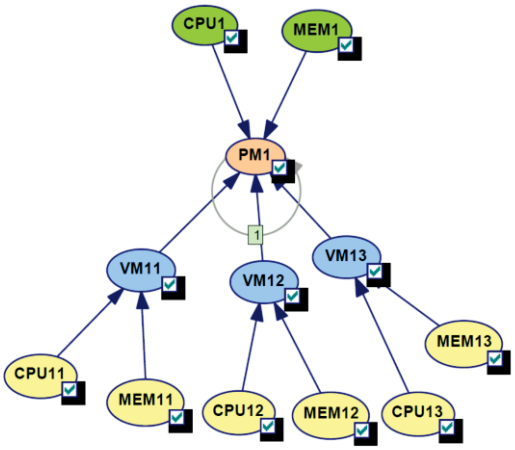


Figure 4.10: A DBN model

The DBN models based on relationships between VMs and PMs are developed. For instance, a DBN model of a PM and its allocated VMs are illustrated in Figure 4.10. The states of the VM and PM nodes are defined as Under-loaded and Overloaded. The states of the resource variables, i.e. CPU and Memory, are determined as Low and High. The CPTs of the VM and PM nodes are set based on the "OR gate" or "AND gate" definitions. The temporal arcs are represented by connecting the PM nodes to themselves. The interpretation is that the PM is Overload when the status persists for a short time.

The evaluation of BN models can therefore be investigated by sensitivity analysis, according to the following three axioms (Naderpour, Lu & Zhang 2014b):

- A slight decrease/increase in the prior probabilities of each parent node should result in a relative decrease/increase of the posterior probabilities of the child node.
- Given the variation in the subjective probability distributions of each parent node, the magnitude of influence of the parent node on the values of the child node should remain consistent.
- The magnitude of the total influence of the combination of probability variations from x attributes (evidence) on the values should be always greater than the probability variations from the set of x-y $(y \in x)$ attributes (sub-evidence).

To validate the proposed DBN models, the parameters need to be closely monitored over a long period, therefore the above axioms are useful for partial validation.

4.5.1 Continuous Variable Discretization

As the online resource variables are continuous, a discretization process is required to use them in BNs. In general, mapping a continuous variable to a discrete variable can be achieved with a crisp set or a fuzzy set. However, fuzzy sets provide a smooth option (Naderpour, Lu & Zhang 2014b). Now suppose $X = \{X_1, X_2, \dots, X_n\}$ is the set of variables in a BN. If the variable X_i is a continuous variable then it can be transformed into a fuzzy random variable W_i. The corresponding set U_i can be utilized to map the variable X_i to fuzzy states:

$$U_i = \{\hat{X}_{i1}, \hat{X}_{i2}, \dots, \hat{X}_{im}\} \tag{4.16}$$

where $\hat{X}_{ij}$ is the j-th fuzzy state and m denotes the number of the fuzzy states, and fuzzy state $\hat{X}_{ij}$ can be defined as follows:

$$\hat{X}_{ij} = \left\{\mu_{\hat{X}_{ij}}(x) \middle| x \in \boldsymbol{X}_i\right\} \tag{4.17}$$

where $\mu_{\hat{X}_{ij}}(x)$ is the membership function of fuzzy state $\hat{X}_{ij}$, $\boldsymbol{X}_i$ is the frame of X_i, and x denotes the value of variable X_i. For a continuous variable X_i, its condition probability in the BN with its parent can be replaced by $P(W_i|Pa(W_i))$:

$$P\left(X_i \middle| Pa(X_i)\right) \to P\left(W_i \middle| Pa(W_i)\right) \tag{4.18}$$

where W_i is the corresponding fuzzy random variable defined by X_i. For a node without parents, soft evidence is equivalent to modifying its prior probability; otherwise, soft evidence on a variable X_i is represented by a conditional probability vector $P(X_i = x|H_i)$ for $i=1,2,\dots,m$, where H_i denotes the hypothesis that the true state is the i-th state. To simplify the inference process for continuous variables, consider the fuzzy random variable W_i with states $\{\hat{X}_{i1}, \hat{X}_{i2}, \dots, \hat{X}_{im}\}$. Define H_j, $j=1,2,\dots,m$ as hypotheses that W_i is in state $\hat{X}_{ij}$. The results of fuzzy function member $\mu_{\hat{X}_{ij}}(x)$, $j=1,2,\dots,m$ form the soft evidence vector:

$$e = \{\mu_{\hat{X}_{i1}}(x), \mu_{\hat{X}_{i2}}(x), \dots, \mu_{\hat{X}_{im}}(x)\} \tag{4.19}$$

The $\mu_{\hat{X}_{ij}}(x)$ is approximately considered to be equivalent to the condition probability $P(\mu_{\hat{X}_{ij}}|X_i = x)$ Then the soft evidence vector can be defined as:

$$e = \{P(W_i = 1|H_1), P(W_i = 1|H_2), \dots, P(W_i = 1|H_m)\} \tag{4.20}$$

where $P(W_i = 1|H_j)$ represents that the observed value of W_i is "1" if the state is $\hat{X}_{ij}$, which is indeed the probability $P(\mu_{\hat{X}_{ij}}|X_i = x)$.

The shapes of the membership functions are defined as a combination of trapezoidal and triangular numbers to simplify the operation and increase sensitivity in a number of bounds. For example, the CPU utilization [%] in terms of operation can be partitioned into fuzzy states Low and High. Its membership function is defined as follows, as well as being shown in Figure 4.11:

$$\mu_{CPU(L)}(x) = \begin{cases} 1 & 0 \le x \le 50 \\ (80 - x)/30 & 50 < x \le 80 \\ 0 & x > 80 \end{cases} \tag{4.21}$$

$$\mu_{CPU(H)}(x) = \begin{cases} 0 & 0 \le x < 50 \\ (x - 50)/30 & 50 \le x < 80 \\ 1 & x \ge 80 \end{cases} \tag{4.22}$$

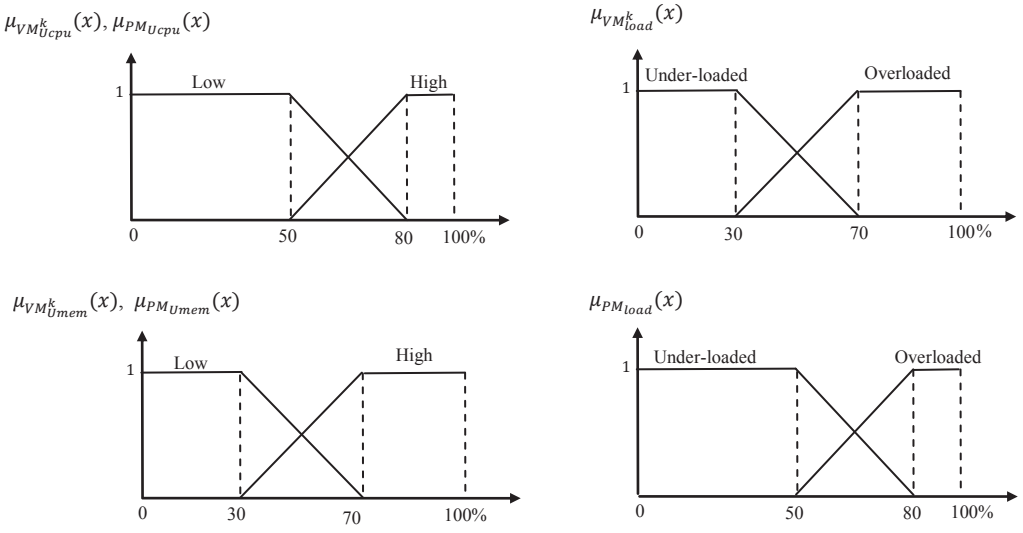


Figure 4.11: The membership functions of variables

The variables that are applied by DBN-WP are defined in the following table:

Table 4.5: The DBN-WP variables

Symbol	Definition
$PM_{Ucpu}(Ts)$	The amount of CPU utilization during the time slot Ts by host PM
$VM^{k}_{Ucpu}(Ts)$	The amount of CPU utilization during the time slot Ts by VM_k
$PM_{Umem}(Ts)$	The amount of activated memory during the time slot Ts by host PM
$VM^{k}_{Umem}(Ts)$	The amount of activated memory during the time slot Ts by VM_k
VM^{k}_{load}	VM_k workload status
PM_{load}	PM workload status

4.5.2. PREDICTIVE ANALYSIS

In the forward method or predictive analysis, the probability of any VM or PM to be overloaded is calculated on the basis of their prior probabilities and online resource states called evidence E, thus yielding the posteriors. This new information usually becomes available during the operational life of the cluster:

$$P(X|E) = \frac{P(X,E)}{P(E)} = \frac{P(X,E)}{\sum_x P(X,E)} \tag{4.23}$$

The static BN models are extended to DBN models by introducing relevant temporal dependencies that capture the dynamic behaviors of the cluster variables between representations of the static network at different times. Two types of dependency are distinguished in the DBNs: contemporaneous and non-contemporaneous.

Contemporaneous dependencies refer to arcs between nodes that represent variables within the same time period, e.g. arcs between one PM and corresponding VMs. Non-contemporaneous dependencies refer to arcs between nodes which represent variables at different times, e.g. arcs between one PM and itself. The DBN models are defined as a pair $(B_1, 2TBN)$ where B1 is a BN which defines the prior distribution $P(X_1)$ and 2TBN is a two-slice temporal BN with

$$P(X_t|X_{t-1}) = \prod_{i=1}^{n} P\left(X_t^i|Pa\left(X_t^i\right)\right) \quad (4.24)$$

where X_t^i is a node at time slice t and $Pa\left(X_t^i\right)$ is the set of parent nodes which can be in time slice t or in time slice t-1. The nodes in the first slice of a 2TBN have no parameters associated with them, but each node in the second slice has an associated conditional probability table for discrete variables, which defines $P\left(X_t^i|Pa\left(X_t^i\right)\right)$ for all $t > 1$. The semantics of the DBN models are defined by "unrolling" the 2TBN until there are T time-slices. The resulting joint distribution is then given by the following equation (Naderpour, Lu & Zhang 2014a):

$$P(X_{1:T}) = \prod_{t=1}^{T}\prod_{i=1}^{n} P\left(X_t^i|Pa\left(X_t^i\right)\right) \quad (4.25)$$

4.5.3 The DBN-WP Algorithm

The DBN-WP algorithm is applied in AS-ORM and is responsible for estimating the probability of exhibiting over- and under-utilization by a set of VMs, based on their CPU and memory usage. DBN-WP algorithm is summarized as follows.

Algorithm 4.2: The DBN-WP Algorithm

Input: CPU and memory usage by VMs and PMs at time slot t (VM_{Ucpu}^k, VM_{Umem}^k, PM_{Ucpu} and PM_{Umem})

[Begin DBN-WP algorithm]

1. Descritisize the continues input variables using fuzzy membership functions
2. Assign the fuzzy values to the DBN models
3. Calculate the posterior probabilities for each VM and PM in the cluster
4. Determine the possible overloading VMs and PMs

[End DBN-WP algorithm]

4.5.4 Implementation

CloudSim (Calheiros et al. 2011) and SMILE (Structural Modeling, Inference, and Learning Engine) (Laboratory 1998) toolkits are applied in this study to implement DBN-WP method. CloudSim is an open source simulation toolkit that enables modeling and

simulation of Cloud computing systems and application provisioning environments. CloudSim allows the extension and definition of policies in all the components of the software stack, and is a customizable research tool that can handle the complexities arising from provisioning, deploying, configuring real resources in a simulated cloud environment. The CloudSim toolkit supports both system and behavior modeling of Cloud system components such as data centers, VMs and resource provisioning policies. SMILE is a library of C++ classes for implementing BNs in intelligent systems.

The experimental setup is composed of three PMs that run seven VMs in total. The properties of PMs and VMs are summarized in Tables 4.6 and 4.7 respectively.

Table 4.6: PM properties

Host ID	MIPS	Memory (MB)	OS	VMM
1	1000	2048	Linux	Xen
2	1000	2048	Linux	Xen
3	1000	2048	Linux	Xen

Table 4.7: VM properties

PM ID	VM ID	MIPS	VM image size	VM memory (Ram)	Bandwidth	Number of CPUs
1	11	250	1000	512	1000	1
1	12	300	1000	256	1000	1
1	13	250	1000	512	1000	1
2	21	250	1000	512	1000	1
2	22	250	1000	512	1000	1
3	31	300	1000	256	1000	1
3	32	250	1000	512	1000	1

4.5.5 Evaluation and Results Analysis

The prior probability of the PMs is set to 1 for the Under-load state and 0 for the Overload state, and it is assumed that none of the PMs is initially Overload. The time interval is set to two minutes, and a scenario which contains sixty time intervals is considered. The use of resources is randomly produced and assigned to the DBN models.

For instance, Figure 4.12 illustrates the monitored CPU utilization of each VM of PM_1 over a 180-minute period. Each VM exhibits a time-varying utilization pattern with interspersed peaks. The figure also depicts a simple capacity bound required by each VM that is based on conservatively satisfying each instantaneous peak, i.e., the capacity required for each VM is set as the maximum demand observed historically.

Figure 4.12: The VM capacity requirements of PM_1

By assigning the fuzzy partitioning values of observable variables to the DBN models, the posterior probabilities of the VMs and PMs are calculated. For example, Figure 4.13 shows the fuzzy states of VM CPUs, and Figure 4.14 presents the VM posterior probabilities of PM_1. As can be seen, there is a sharp increase in the overloading probability of VM_{11} at interval 13. In addition the posterior probability of PM_1 shows that it is not overloaded in this interval. Therefore, based on the posterior probabilities the decision is to expand the VM_{11} as there is idle resource in PM_1. At interval 27, VM_{13} is overloaded and PM_1 is also overloaded, so the decision is to migrate VM_{12} because it is the smallest in this

PM. There is no idle resource for its expansion, therefore migrating it will assist in meeting the terms of the SLA.

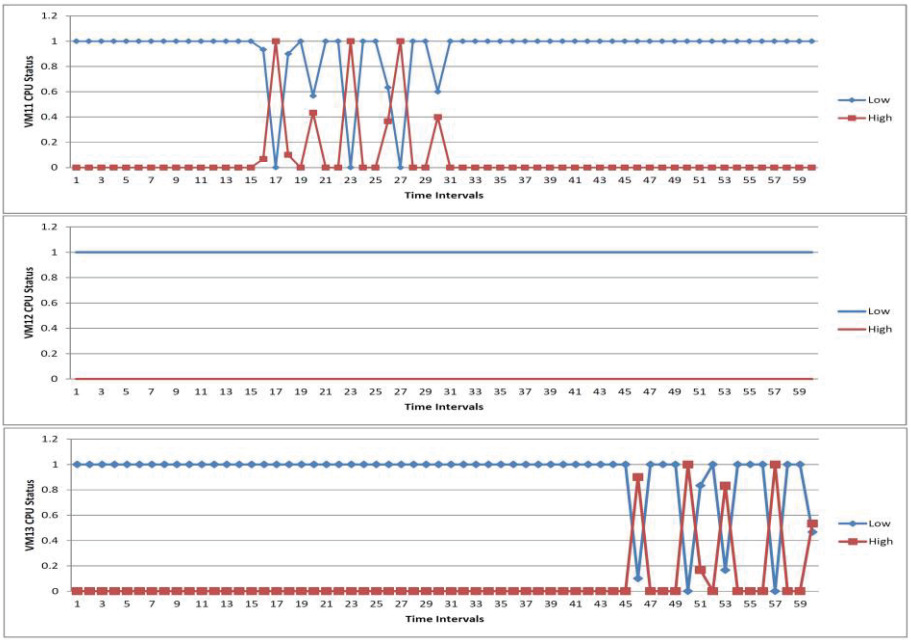


Figure 4.13: The fuzzy states of VM CPUs

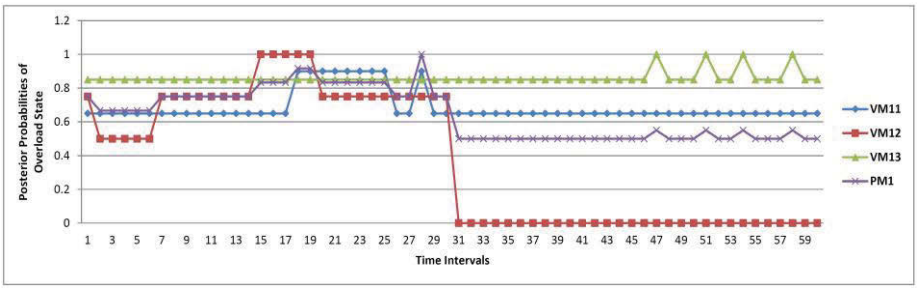


Figure 4.14: The posterior probabilities of VM workloads in PM1

4.6 SUMMARY

Prediction methods such as neural network and linear regression are widely applied in previous works to forecast workload and resource utilization of VMs in cloud environments to estimate required resource and size for each VM. However, these prediction methods rely on related historical data in a time slot *t* to predict future resource utilization.

Considering this fact that future VMs' workload and resource utilization could be independent from their previous workload pattern, the proposed prediction methods in this study not only applies a Neural Network (NN) to predict resource utilization patterns by VMs, but also applies ES and DBN techniques to predict near future changes in resource utilization for every VMs, and determine the time that a set of VMs will be poorly performed or a PM become overloaded.

Chapter 5.

MULTI-OBJECTIVE TASK SCHEDULING SUB-SYSTEM

5.1 INTRODUCTION

Cloud computing is a service-oriented computing paradigm that has significantly revolutionized computing by offering three web-based services – Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) (Taheri et al. 2014). SaaS and PaaS users generate several jobs in a cloud environment. To deliver these services over the World Wide Web, jobs generated by users are submitted to schedulers to be executed by a set of processors/VMs (cloud resources). Each job consists of several dependent tasks described by a Directed Acyclic Graph (DAG) (Taheri, Zomaya, et al. 2013). In the cloud environment, the number of tasks in a workflow, as well as the number of available resources, can grow quickly, especially when virtual resources are allocated. Calculating all possible task-resource mappings in the cloud environment and selecting the optimal mapping is not feasible, since the complexity would grow exponentially with the number of tasks and resources (Juhnke et al. 2011). Therefore, the development of intelligent task scheduling mechanisms that take into account the efficiency of all the cloud computing facilities, has become a critical part of current cloud optimization problems and

plays a key role in improving flexible and reliable systems with optimum resource utilization. The main purpose is to schedule tasks to adaptable resources in accordance with time, which involves establishing a proper sequence whereby tasks can be executed under transaction logic constraints (Tayal 2011).

To create a higher level of resource utilization while minimizing cost and maximizing QoS, a multi-objective model for optimizing task scheduling is developed in this study that considers four aspects of the task scheduling optimization problem: task execution cost, task transfer time, task queue length and power consumption.

The efficiency of the proposed model is evaluated by applying the CloudSim (Calheiros et al. 2011) toolkit through different scenarios. Simulation results show that the proposed model significantly increases QoS by considering more criteria for optimization, and has the ability to satisfy both users and providers. In fact, the proposed model is able to determine trade-off solutions that offer the best possible compromises among the optimization objectives, and not only helps cloud providers to reduce the cost of power consumed, but also assists them to maintain the expected level of QoS. It has also been found that MOPSO is a faster and more accurate evolutionary algorithm than MOGA for solving such problems. The contributions of this chapter are summarized as follows:

1) Develop a multi-objective task scheduling model
2) Develop intelligent methods to estimate: (1) task transfer time, (2) task execution cost, (3) power consumption based on the number of active PMs, and (4) task queue length based on the remaining resource capacity (memory and idle CPUs).
3) Develop a Multi-Objective Task Scheduling algorithm by applying MOPSO and MOGA called MOTS-PSO/GA to solve the proposed multi-objective task scheduling problem, and compare two evolutionary algorithms in different scenarios.
4) Improve the functionality of MOPSO and MOGA algorithms to optimize their first initialization and consider task priority and dependency.

5) Extend the CloudSim (Calheiros et al. 2011)package by applying MOPSO and MOGA as its task scheduling algorithms to implement and evaluate the proposed model.

The rest of this chapter is organized as follows. In Section 5.2, MOPSO and MOGA are introduced. Section 5.3 illustrates a multi-objective model for task scheduling optimization. The objective functions of the model are described in Section 5.4. In Section 5.5 the conducted improvement in the functionality of MOPSO and MOGA algorithms are discussed. Section 5.6 presents the model of the developed multi-objective problem. In Section 5.7 the MOTS-PSO/GA algorithm is developed; this algorithm determines the optimal solution (task scheduling pattern) for the proposed multi-objective model. The model is evaluated in Section 5.8 Lastly, the conclusion and future works are provided in Section 5.9.

5.2 MULTI–OBJECTIVE EVOLUTIONARY ALGORITHMS

In this section the preliminary definitions of two popular evolutionary algorithms including MOPSO and MOGA are presented.

5.2.1 MULTI–OBJECTIVE PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) is a population-based search algorithm based on the simulation of the social behavior of birds which was originally proposed by Kennedy and Eberhart (Kennedy & Eberhart 1995). Although originally adopted for balancing weights in neural networks, PSO soon became a very popular global optimizer, mainly in problems in which the decision variables are real numbers (Engelbrecht 2005, 2007). In PSO, particles are flown through hyper-dimensional search space. Changes to the position of the particles within the search space are based on the social–psychological tendency of individuals to emulate the success of other individuals. The position of each particle is changed according to its own experience and that of its neighbors. Let $\vec{X}_i(t)$ denote the position of particle *i*, at iteration *t*. The position of $\vec{X}_i(t)$ is changed by adding a velocity $\vec{V}_i(t+1)$ to it, i.e.:

$$\vec{X}_i(t+1) = \vec{X}_i(t) + \vec{V}_i(t+1) \quad (5.1)$$

The velocity vector reflects the socially exchanged information and, in general, is defined in the following way:

$$\vec{V}_i(t+1) = W\vec{V}_i(t) + C_1 r_1 \left(\vec{x}_{pbest_i} - \vec{X}_i(t)\right) + C_2 r_2 \left(\vec{x}_{gbest_i} - \vec{X}_i(t)\right) \quad (5.2)$$

where C_1 is the cognitive learning factor and represents the attraction that a particle has towards its own success; C_2 is the social learning factor and represents the attraction that a particle has towards the success of the entire swarm; W is the inertia weight, which is employed to control the impact of the previous history of velocities on the current velocity of a given particle; $\vec{x}_{pbest_i}$ is the personal best position of the particle i; $\vec{x}_{gbest}$ is the position of the best particle of the entire swarm; and $r_1, r_2 \in [0,1]$ are random values (Gao et al. 2011; Lu, Zhang & Ruan 2007; Mahmoodabadi et al. 2012).

5.2.2 Multi-Objective Genetic Algorithm

A multi-objective genetic algorithm (MOGA) is concerned with the minimization of multiple objective functions that are subject to a set of constraints. In this algorithm, an initial population whose scale is N is first randomly generated. The first generation child population is gained through non-dominated sorting and basic operations such as selection, crossover and mutation (Srinivas & Deb 1994).

The crossover operator is the most important operator of GA. In crossover, generally two chromosomes, called parents, are combined together to form new chromosomes, called offspring. The parents are selected among existing chromosomes in the population with preference towards fitness so that offspring is expected to inherit good genes which make the parents fitter. By iteratively applying the crossover operator, genes of good chromosomes are expected to appear more frequently in the population, eventually leading to convergence to an overall good solution. The mutation operator introduces random changes into characteristics of chromosomes. Mutation is generally applied at the gene level. In typical GA implementations, the mutation rate (probability of changing the

properties of a gene) is very small and depends on the length of the chromosome. Therefore, the new chromosome produced by mutation will not be very different from the original one. Mutation plays a critical role in GA. As discussed earlier, crossover leads the population to converge by making the chromosomes in the population alike. Mutation reintroduces genetic diversity back into the population and assists the search escape from local optima (Konak, Coit & Smith 2006).

After generating the first child population, from the second generation on, the parent population and the child population are merged and sorted based on fast non-dominated solutions. The crowding distance between individuals on each non-dominated layer is calculated. According to the non-dominant relationship and crowding distance between individuals, appropriate individuals for forming a new parent population are selected. Lastly, a new child population is generated through basic operations of the genetic algorithm which iterates until the conditions of the process end can be met (Zhang et al. 2011).

5.3 Multi-Objective Task Scheduling Optimization Model

A cloud environment dynamically receives a large number of tasks from its applications' users in every portion of each second. These tasks accumulate in several queues and are then sent to the task schedulers. The task schedulers are responsible for allocating these tasks to VMs for execution. These VMs which are in turn allocated to PMs (see Figure 5.1), have different numbers of virtual CPUs and different memory size. The task schedulers apply optimization procedures to allocate tasks among VMs to achieve optimal resource utilization in a cloud environment. The scheduling process repeats dynamically to schedule every set of arrival tasks among VMs. Considering this fact that independent users arbitrarily send tasks to the cloud environment, the number and type of tasks in each queue may significantly change from one scheduling to another.

There are several studies (Guo et al. 2012; Juhnke et al. 2011; Lei et al. 2008; Li et al. 2011; Li et al. 2012; Salman, Ahmad & Al-Madani 2002; Song, Hassan & Huh 2010;

Taheri et al. 2014; Tayal 2011) in the area of job/task scheduling that mainly emphasize the minimization of job makespan and task execution cost in their multi-objective optimization models by applying evolutionary algorithms. However, these studies fail to consider the need to minimize power consumption by the cloud infrastructure. Meanwhile, there is a considerable amount of research work that focuses on reducing power consumption in their proposed bi-objective task scheduling models by minimizing the value of their developed predefined power consumption objective functions for multi-core processors and cloud environment (Shieh & Pong 2013; Wang, Wang & Cui 2014). In addition, several studies have been undertaken in the area of energy-aware task scheduling by applying the dynamic voltage frequency scaling technique (Mahabadi, Zahedi & Khonsari 2013; Rizvandi, Taheri & Zomaya 2011; Su et al. 2013; Wang et al. 2013).

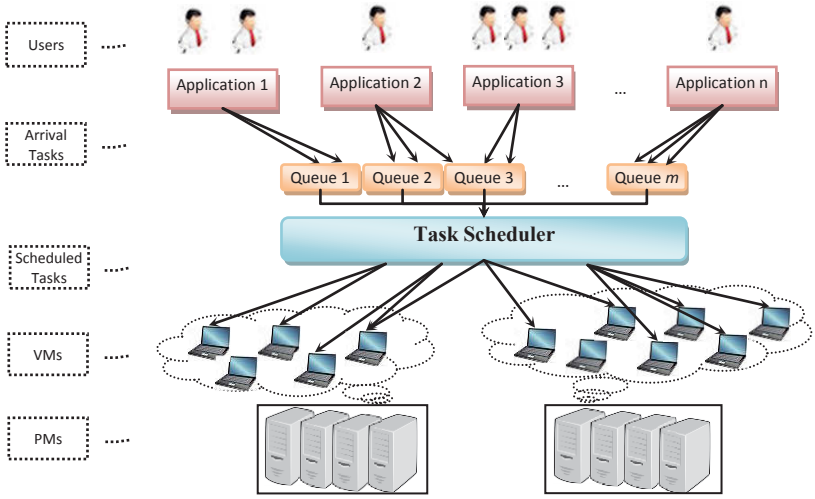


Figure 5.1: Cloud objects and their relations

To the best of our knowledge, reducing the task queue length of VMs has not been investigated in proposed task scheduling optimization models. In addition, the current job/task scheduling models are single or bi-objectives and have focused on at most two aspects of optimization.

A longer task queue results in more waiting time for tasks and a longer response time. In current task scheduling optimization models that apply evolutionary algorithms with

predefined objective functions, the optimal task scheduling schemas are suggested based on task and VM properties. In such models, optimizing objective functions usually occurs by assigning tasks to high performance VMs and neglecting low performance VMs. This leads to the creation of a long task queue for some VMs, while some other VMs remain idle. Although the properties of low performance VMs do not optimize the value of objective functions, as they have the lowest number of CPUs and smallest amount of memory, they can decrease response time by executing waiting tasks in the queues.

To improve previous research work and address these shortcomings in the existing literature, this study determines an objective function to achieve optimal load balance between resources and control the length of task queues in a cluster. This objective function considers changes in resource capacity (memory and number of CPUs) to avoid assigning multiple tasks to one VM's processors and creating long task queues. In addition, this study proposes a multi-objective model for task scheduling in a cloud environment that concurrently considers four aspects of optimizing cloud utilization including: task transfer time, task execution cost, power consumption, and task queue length. The model aims to enhance QoS based on the points of view of both cloud users and providers by minimizing service response time and price (to raise customer satisfaction), and minimizing power consumption (to reduce providers' expenditure). The MOTS-PSO/GA algorithm is then developed to find the optimal solution for the proposed model.

This chapter focuses on scheduling two types of highly parallel computations in a cloud environment namely: (1) a Bag-of-Tasks (BoT) applications for SaaS, and (2) a set of several dependent tasks that belong to a moldable job–described by a DAG. Jobs are generated by users and are submitted to schedulers to be executed by Computational Nodes (CNs) or VMs (Taheri, Choon Lee, et al. 2013). In BoT applications, completion of one task does not affect the completion of other tasks, and only one task is executed on a computer processor (CPU) at a time. BoT applications are used for data mining, massive searches, parameter sweeps, simulations, fractal calculations, computational biology, and computer

imaging (Cirne et al. 2006; Tchernykh et al. 2014). In contrast, for the set of tasks in a job/DAG, execution of each task depends to completion of its previous/parent task. In addition, there is data dependency between tasks and data to execute each task is provided to it through (1) previously existed data files listed by the list of required data files and/or (2) output of the task's immediate predecessors in the DAG (either as local/temporary data file or inter-processing messages) (Taheri, Choon Lee, et al. 2013). In this study, two procedures are developed to consider task priority and dependency.

The MOTS sub-system is employed by AS-ORM for two different purposes: (1) as task scheduler to schedule arrival PaaS and SaaS tasks to VMs, (2) as part of FP-TBSLB sub-system to transfer tasks from a poorly performing VM to other sets of high performance VMs. In addition, the location of stored data correspond to tasks are not considered to develop the MOTS sub-system. Therefore, for the first purpose, the model is applicable for computationally intensive jobs/tasks with light data files, and data intensive jobs/tasks with input/output data located in the same data storage. The MOTS sub-system works wider for FP-TBSLB sub-system and has no limitation for data intensive jobs/tasks. This is because, in this case input data files of each task has been uploaded to primary VM and move along with the task to the destination VM, and the location of data storage does not affect the resulting transfer time.

5.4 The MOTS-PSO/GA Objective Functions

In this study, available resources in VMs that are determined to be possible destinations for the arrival tasks —or extra workload of a poorly performing VM— will be less than their allocated resources. Based on this fact, the MOTS-PSO/GA objective functions are formulated and explained in the following. The variables that are applied to formulate this multi-objective model are defined in Table 5.1.

Table 5.1: The MOTS_PSO variables

Symbol	Definition
n	The number of tasks that have accumulated in the task queue of a poorly performing VM (VM_l)
T_{set}^{l}	Set of tasks that have accumulated in the task queue of VM_l
DF_i	The task$_i$ file size (MB)
DO_i	The task$_i$ output file size (MB)
DI_i	The task$_i$ input file size (MB)
PR_i	$\{j \mid \text{task}_j \text{ is prerequisite for task}_i\}$
tm_i	The maximum level of memory required for executing task i (MB)
tc_i	The number of CPUs required for executing task i
tcu_i	The amount of CPU usage of task i (*GHz*)
$Texe_k$	The total task execution time on VM_k (Hour)
m	The number of VMs
VM_k	Virtual Machine k, k={1, 2, …, m}
VM_m^k	The amount of memory allocated to VM_k (MB ≈ 0.001 GB)
VM_c^k	The number of CPUs allocated to VM_k
VM_{ac}^k	The number of available CPUs allocated to VM_k
N_{aCPU}^k	The number of active CPUs on VM_k
VM_{bw}^k	The bandwidth of VM_k (Mb/s)
SN_{bw}	The bandwidth of storage node (Mb/s)
$VM_{CPUSpeed}^k$	The CPU computing speed of VM_k (*GHz*)
RVM_m^k	The amount of available memory on VM_k
RVM_c^k	The number of available CPUs on VM_k
N_{PM}	The number of PMs in cloud
N_{aPM}	The number of active PMs in cloud
Svm_z	$\{k \mid VM_k \epsilon\ zth\ PM, z \in \{1,2,\dots,N_{PM}\}\}$=The set of indexes of VMs which are located on zth PM
cp	The number of could providers
SP_p	$\{k \mid VM_k \epsilon\ Pth\ \ cloud\ provider, P \in \{1,2,\dots,cp\}\}$= The set of VMs belong to pth provider
$Pcost_p$	The cost of one CPU for pth provider (AUD/hour)
x_{ij}	1 if task i is assigned to VM_k and 0, otherwise
VM_{set}^l	Set of VMs that are under-loaded and compatible with VM_l

5.4.1 Tasks Transfer Time

When a task is assigned to a VM to be executed, the input data of the task and the output data of its prerequisite tasks are uploaded to the VM from the corresponding storage node to the VM. Therefore, when MOTS-PSO/GA is applied as part of FP-TBSLB sub-system —to transfer task from one VM to another— the task and the output data produced by its prerequisite tasks should be transferred from overloaded VM (VM_l)to the destination VM (VM_k) to be executed. In this case, the total task transfer time —for both computing and data intensive tasks— is estimated as follows where the coefficient 1/8 is used to convert Mb to MB:

$$Ttrans = \sum_{k=1}^{m}(\sum_{i=1}^{n} \frac{x_{ik} * (DF_i + DI_i + \sum_{j\in PR_i} DO_j)}{\min(VM_{bw}^{l}, VM_{bw}^{k}) * (\frac{1}{8})} \tag{5.3}$$

To apply MOTS-PSO/GA as task scheduler module of AS-ORM, the value of DF_i and $\sum_{j\in PR_i} DO_j$ are uploaded from a storage node to the destination VM. In addition, in this case the location of stored data is assumed to be the same as the model is designed for computationally intensive tasks and data intensive tasks with stored data in share storage node. Therefore, the total task transfer time for this purpose is calculated using the following equation:

$$Ttrans = \sum_{k=1}^{m}(\sum_{i=1}^{n} \frac{x_{ik} * (DF_i + DI_i + \sum_{j\in PR_i} DO_j)}{\min(SN_{bw}, VM_{bw}^{k}) * (\frac{1}{8})} \tag{5.4}$$

5.4.2 Task Execution Cost and Time

The destination VMs to execute scheduled tasks may belong to independent cloud providers who offer different prices for their leased infrastructure. Considering this, the task execution cost (AUD per hour) for provider p is calculated as follows:

$$Cexe_p = \sum_{k\in SP_p} Pcost_p * Texe_k \tag{5.5}$$

where $Pcost_p$ is the cost of one CPU for pth provider in AUD per hour, and $Texe_k$ is the estimated execution time (in hours) of the tasks assigned to each VM_k belonging to provider p.

The $Texe_k$ is estimated based on the fact that the VMware ESXi divides the CPU speed of VM_k ($VM^k_{CPUSpeed}$) by the number of its active CPUs. Therefore, to determine the execution time of a set of tasks scheduled to VM_k, the number of active CPUs during the execution time of these tasks is approximated. To estimate this number, first the total number of tasks that are scheduled to VM_k is determined as:

$$No^k_t = \sum_{i=1}^{n} x_{ik} \tag{5.6}$$

Then, the integer division of No^k_t divided by VM^k_{ac} is calculated as follows:

$$Div^k_t = \ No^k_t \setminus VM^k_{ac} \tag{5.7}$$

This value is used to estimate how many tasks (i.e.$Div^k_t * VM^k_{ac}$) will be executed while all the CPUs of VM_k are active and the CPUs' speed is equal to $\frac{VM^k_{CPUSpeed}}{VM^k_c}$.

The remainder after the division of No^k_t by VM^k_{ac} (modulo) is calculated in Equation 5.8 to estimate how many CPUs will be used to execute the rest of the tasks. In this situation, the CPU speed of VM_kwould be divided by the number of its active CPUs as calculated in Equation 5.9.

$$Mod^k_t = \ No^k_t \,\%\, VM^k_{ac} \tag{5.8}$$

$$N^k_{aCPU} = Mod^k_t + CN^k_{aCPU} \tag{5.9}$$

where CN^k_{aCPU} is the number of CPUs in VM_kthat have been busy before tasks are scheduled to it and is estimated as:

$$CN^k_{aCPU} = VM^k_c - VM^k_{ac} \tag{5.10}$$

For example, when there are 13 tasks scheduled to VM_k with four CPUs, three of which are available, then $Div^k_t = 13 \setminus 3 = 4$. This means that 12 tasks would be executing while all the CPUs of VM_k are busy, and CPU speed is $\frac{VM^k_{CPUSpeed}}{4}$. In this

example, $Mod_t^k = 13\ \%\ 3 = 1$. This means that one of these tasks will be executed while the number of busy CPUs in VM_k is $Mod_t^k + CN_{aCPU}^k = 1 + (4 - 3) = 2$, and therefore CPU speed equals $\frac{VM_{CPUSpeed}^k}{2}$.

$Texe_k$ is then estimated in hours by applying the aforementioned assumptions as follows:

$$Texe_k = \left(\frac{\sum_{i=1}^{Div_t^k * VM_{ac}^k} tcu_i}{\left(\frac{VM_{CPUSpeed}^k}{VM_c^k}\right)} + \frac{\sum_{i=(Div_t^k * VM_{ac}^k)+1}^{No_t^k} tcu_i}{\left(\frac{VM_{CPUSpeed}^k}{Mod_t^k + CN_{aCPU}^k}\right)}\right) * \frac{\rho}{60} \tag{5.11}$$

The value of $\rho = 0.33$ has been empirically determined.

The same price is assumed for all CPUs in a VM; therefore, task execution sequence and scheduling schema in each VM are not considered in this function. For instance, in a VM with three CPUs, the cost of assigning three tasks to three different CPUs will be the same as the cost of assigning the execution of three tasks to one CPU. The total task execution cost for all providers is then determined as:

$$Cexe = \sum_{p=1}^{cp} Cexe_p \tag{5.12}$$

In situations where minimizing the execution time of tasks is more important than cost, the task execution time needs to be estimated. To do this, the value of coefficient $Pcost_p$ in Equation 5.5 for all providers is considered equal to 1. Using this, Equation 5.12 will be an estimation for the total task execution time in hours.

5.4.3 Power Consumption

Several power-aware multi-objective task scheduling models have been proposed for multi-core processors, grid and cloud environments (Shieh & Pong 2013; Wang, Wang & Cui 2014). A variety of linear and non-linear objective functions have been suggested in these models to estimate power consumption based on task scheduling patterns by considering the fact that energy will be reduced when the PM is either off or in idle mode (Priya, Pilli & Joshi 2013; Tchernykh et al. 2014; Zhang & Guo 2013). It has also been proved by Buyya et al. (Buyya, Beloglazov & Abawajy 2010; Priya, Pilli & Joshi 2013) that

an idle server consumes around 70% of the power consumed by a fully utilized server. In summary, previous studies show that having fewer active CPUs and PMs leads to lower power consumption in a cluster. Considering this fact, the ratio of active PMs and CPUs to all available PMs and CPUs has been minimized in this study to reduce power consumption. By applying this as an objective function, the optimization model avoids the selection of VMs on idle PMs as destinations for scheduled tasks and consequently reduces the power consumed in the corresponding cloud cluster. To calculate this ratio, first the power consumption for each fully utilized PM is determined as follows, based on the fact that the power consumed by PM increases by the number of active CPUs:

$$Pw_{ful}(PM_z) = Pw_0^z + \alpha N_{CPU}^z \tag{5.13}$$

where Pw_0^z is the amount of power that PM_z consumes when all of its CPUs are idle, N_{CPU}^z is the number of CPUs of PM_z, and α is the amount of extra power consumed above Pw_0^z for every active CPU of PM_z. Using this, the value of α is:

$$\alpha = \frac{Pw_{ful}(PM_z) - Pw_0^z}{N_{CPU}^z} \tag{5.14}$$

For a cluster with homogenous PMs, the value of α and Pw_0 are the same for every PM. Therefore, the total amount of consumed power in this cluster can be determined as follows:

$$Pw_{ful}(Cluster) = \sum_{z=1}^{N_{PM}} Pw_{ful}(PM_z) = \sum_{z=1}^{N_{PM}} Pw_0^z + \alpha \sum_{z=1}^{N_{PM}} N_{CPU}^z \tag{5.15}$$

then

$$Pw_{ful}(Cluster) = Pw_0^z * N_{PM} + \alpha * N_{CPU} \tag{5.16}$$

where N_{CPU} is the total number of CPUs in the cluster. On the other hand, $N_{CPU} = \sum_{k=1}^{m} VM_c^k$. Using this $Pw_{ful}(Cluster)$ is calculated as:

$$\mathrm{Pw}_{ful}(Cluster) = Pw_0^z * N_{PM} + \alpha * \sum_{k=1}^{m} VM_c^k \tag{5.17}$$

By applying the same logic, the value of the power consumed by active PMs and CPUs in this cluster is calculated as follows:

$$Pw_{\text{active}}(Cluster) = Pw_0^z * N_{aPM} + \alpha * \sum_{k=1}^{m} N_{aCPU}^k \quad (5.18)$$

where N_{aCPU}^k is the number of active CPUs in VM_k. To calculate N_{aCPU}^k, the number of CPUs of VM_k that are already busy is calculated by applying Equation 5.10, and the number of activated CPUs is considered. The number of tasks assigned to VM_k may be less than the number of available CPUs in VM_k, or may exceed this number. Therefore, the number of activated CPUs in VM_k —after scheduling the tasks to it— is equal to the minimum value of the total number of tasks assigned to VM_k and the total number of available CPUs in this VM. The total number of active CPUs in VM_k can be estimated as:

$$N_{aCPU}^k = CN_{aCPU}^k + min(\sum_{i=1}^{n} x_{ik}, VM_{ac}^k) \quad (5.19)$$

N_{aPM} is the number of active PMs in all iterations and is estimated based on the fact that PM_z will be activated if at least one of its allocated VMs is active. The activation status of VMs is determined based on the current number of busy CPUs (CN_{aCPU}^k) and the number of tasks assigned to them. VM_k is already activated if at least one of its CPUs is busy before a task is scheduled to it. The current status of VM_k is then calculated as:

$$CVM_{status}^k = \min(CN_{aCPU}^k, 1) \quad (5.20)$$

In the situation where all CPUs of VM_k are idle (available), VM_k will be activated by having at least one task assigned to it, and the status of VM_k is determined by the following formula:

$$VM_{status}^k = \min(\sum_{i=1}^{n} x_{ik}, 1) \quad (5.21)$$

where $\sum_{i=1}^{n} x_{ik}$ is the number of tasks assigned to the available CPUs of VM_k. By applying Equations 5.20 and 5.21, the number of active VMs located on PM_z is calculated as follows:

$$N_{aVM}^z = \sum_{k \in Svm_z} (CVM_{status}^k + (1 - CVM_{status}^k) * VM_{status}^k) \quad (5.22)$$

and the number of active PMs is determined using the following formula:

$$N_{aPM} = \sum_{z=1}^{Npm} \min(N_{aVM}^{z}, 1) \tag{5.23}$$

Using Equations 5.17 and 5.18, the ratio of consumed power in the cluster for each task scheduling pattern can be calculated as:

$$\frac{Pw_{active}(Cluster)}{Pw_{ful}(Cluster)} = \frac{Pw_0^z * N_{aPM} + \alpha * \sum_{k=1}^{m} N_{aCPU}^{k}}{Pw_0^z * N_{PM} + \alpha * \sum_{k=1}^{m} VM_c^k} = \frac{Pw_0^z \left(N_{aPM} + \frac{\alpha}{Pw_0^z} * \sum_{k=1}^{m} N_{aCPU}^{k}\right)}{Pw_0^z (N_{PM} + \frac{\alpha}{Pw_0^z} * \sum_{k=1}^{m} VM_c^k)} \tag{5.24}$$

As a result, the following formula is developed as an objective function of the task scheduling model to control power consumption in the cluster:

$$PowerC = \frac{(N_{aPM} + \mu \sum_{k=1}^{m} N_{aCPU}^{k})}{(N_{PM} + \mu \sum_{k=1}^{m} VM_c^k)} \tag{5.25}$$

where $\mu = \frac{\alpha}{Pw_0^z}$. For instance, for one of the applied servers in this study (Altix XE320 (SGI 2008)) with $Pw_{ful}(PM) = 115^{kw}$, $Pw_0 = 40^{kw}$(Top 500 Supercomputing Sites 2014), and 16 number of CPUs, the value of μ is equal to 0.1.

5.4.4 Length of VM Task Queue

For each possible best solution of the multi-objective task scheduling pattern, if the number of tasks assigned to VM_k exceeds the number of its CPUs, extra tasks will be allocated to the task queue of VM_k (see Figure 5.2). Considering this fact that the longer task queue causes the longer time for the tasks to be completed, another objective function is determined in this model to optimize the task scheduling pattern by minimizing the length of VM task queues. This will reduce the makespan of the corresponding jobs to the tasks, and consequently reduce response time, as fewer tasks will be located in queues and in waiting mode.

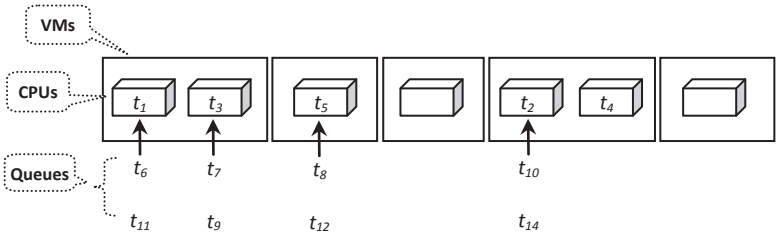


Figure 5.2: A sample task scheduling pattern among VMs

The following objective function is proposed to minimize the number of tasks in queues that will be created in any possible optimal task scheduling pattern:

$$\gamma_k = \sum_{i=1}^{n} x_{ik} * \frac{1}{\left(RVM_m^k\right) * \left(RVM_c^k\right)} \tag{5.26}$$

where the value of RVM_m^k and RVM_c^k is the portion of remaining memory and CPU capacity of VM_k respectively after VM_k receives the scheduled tasks, calculated as:

$$RVM_m^k = \frac{VM_{am}^k - \left(\sum_{j=1}^{i} x_{jk} * tm_j\right)}{VM_{am}^k} \tag{5.27}$$

$$RVM_c^k = \frac{VM_{ac}^k - \left(\sum_{j=1}^{i} x_{jk} * tc_j\right)}{VM_{ac}^k} \tag{5.28}$$

where $\left(\sum_{j=1}^{i} x_{jk} * tm_j\right)$ and $\left(\sum_{j=1}^{i} x_{jk} * tc_j\right)$ are the total amount of memory and number of CPUs required to execute tasks previously assigned to VM_k.

To calculate γ_k, the formula that was proposed in (Guo et al. 2012; Lei et al. 2008; Ramezani, Lu & Hussain 2012) is improved by considering changes in the capacity of available VMs after new tasks have been assigned to them. Equation 5.26 indicates that scheduling task *i* to VM_k —which reduces the amount of available CPU and memory of the corresponding VM— negatively affects VM_k's performance and the execution time of its tasks. In this formula, if $VM_{am}^k \leq \left(\sum_{j=1}^{i} x_{jk} * tm_j\right)$ and/or $VM_{ac}^k \leq \left(\sum_{j=1}^{i} x_{jk} * tc_j\right)$, the implication is that there is no available memory or CPU to execute extra tasks. In this situation, if RVM_m^k and/or RVM_c^k have a negative value, they will be converted to $(RVM_m^k)^{-1}$ and/or $(RVM_c^k)^{-1}$ then multiplied to a small value (-10^{-3}), and if each of them is equal to zero, the value of 10^{-3} will be added to them. Since the values of RVM_m^k and

RVM_c^k are multiplied by each other, this will increase the value of γ_k dramatically whenever one of them equals zero or has a negative value. Therefore, the probability of choosing this pattern as a possible optimal solution will be decreased. This prevents the assignation of tasks to VMs without available resources. The following formula is then used as an optimization objective function in the proposed MOTS model to minimize VM task queue length and optimize the load balance:

$$\Gamma = \sum_{k=1}^{m} \gamma_k \tag{5.29}$$

5.5 MOPSO AND MOGA IMPROVEMENT

In this section the suggested improvement in the functionality of MOPSO and MOGA algorithms are described. The first improvement is adopted from Taheri et al. (2013) to optimize the initialization of first population by the evolutionary algorithms. The second improvement is developed to complete the functionality of the evolutionary algorithms to consider task priority and data dependency.

5.5.1 OPTIMIZE FIRST INITIALIZATION

The first population in evolutionary algorithms is usually initialized randomly. In this study, the first population is determined using VMs and task properties to accelerate the performance of MOPSO and MOGA. Evolutionary algorithms are therefore expected to find the best solution faster because they start from the near best solution pattern. To achieve this, the VMs with a greater number of CPUs and a large amount of memory on active PMs are selected as the new hosts for excess tasks. The Roulette Wheel (RW) technique is applied as it is suggested in (Taheri, Zomaya, et al. 2013) to produce the first population for both MOPSO and MOGA. The RW selection method randomly selects a given choice from several options, based on the value of their winning probability. In this technique, the slots of a roulette wheel are first filled with the winning chance of the options, then the wheel is spun and an option is selected (Taheri, Zomaya, et al. 2013).

Equations 5.30 and 5.31 are proposed to determine the probability of choosing a task, and a host VM respectively. Then, based on the normalized winning chance, two roulette wheels are generated for tasks and VMs.

$$Chance_{VM_k} = \frac{VM^k_{CPUSpeed}}{VM^k_c} + VMm_k + (\rho_1 * VM^k_c) + \frac{VM^k_{bw}}{\rho_2} \quad (5.30)$$

$$Chance_{Task_i} = \frac{DF_i + DI_i + \sum_{j \in PR_i} DO_j}{\rho_3} \quad (5.31)$$

where coefficients ρ_1, ρ_2, and ρ_3 have been determined based on the value of task and VM properties to make the same impact for all properties in determining the winning chance. In this study, based on the data given in Tables 5.4 and 5.5 the value of these coefficients are determined as ρ_1= ρ_2=100 and ρ_3=1000. Figure 5.3 shows a RW for VMs that is illustrated based on the information in Table 5.4.

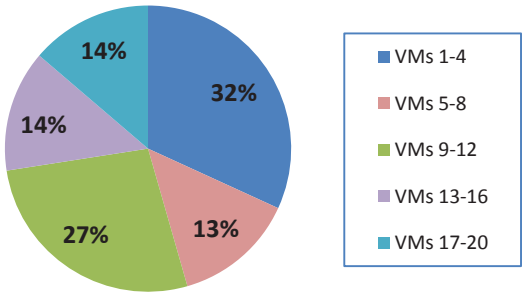


Figure 5.3: VM roulette wheel (winning probability)

After creating RWs, a task and a VM with the highest biggest slice in the RWs correspond to tasks and VMs respectively are selected as t_i and VM_j. Then the selected task(t_i) is deleted from the task set, and a new RM is generated for the new set of tasks. In addition, VM_j properties are updated by applying the following equations:

$$Available\_memory_{VM_j} = VM^k_m - tm_i \quad (5.32)$$

$$Available\_CPU_{VM_j} = VM^k_c - tc_i \quad (5.33)$$

If the value of $Available\_memory_{VM_j}$ or $Available\_CPU_{VM_j}$ is less than zero, this means VM_j has no available capacity to execute additional tasks. This VM is therefore deleted from the VM set and a new RW is generated for the new set of VMs based on their

available capacity. After selecting all VMs as the new destination for executing a set of tasks, and deleting those from the available set of VMs, all VMs are applied again to execute the remaining tasks. These tasks are allocated to the VM task queues in waiting mode.

The corresponding steps for implementing the RW technique to initialize the first population for MOGA and MOPSO are described in Steps 2 of the MOTS-PSO/GA algorithm that is presented in Section 5.7.

5.5.2 Job Priority and Dependency

Evolutionary algorithms like MOPSO and MOGA create a random scheduling pattern. This study develops two procedures to direct these algorithms to follow task priorities in their suggested solutions. The developed procedures modify the suggested scheduling

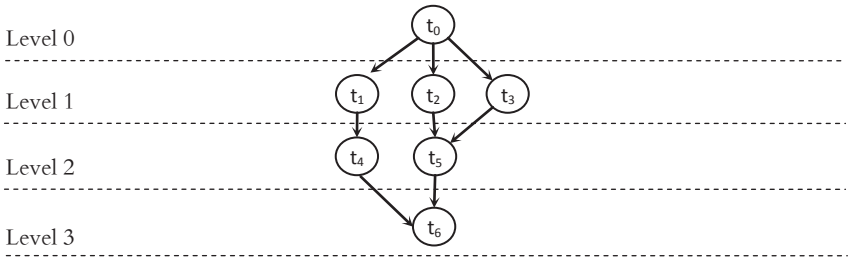


Figure 5.4: A DAG of a job

pattern by these algorithms and take the task priorities into account. These procedures are applied after the new scheduling pattern has been suggested by the evolutionary algorithm, and before the fitness functions are calculated, and use the information about task priorities and their position (level) in their corresponding job (see Figure 5.4).

The procedures are applied to solve two problems of the original optimal scheduling pattern. The first problem arises when several tasks with the same level are scheduled to one VM in a task scheduling pattern. This prevents executing tasks in parallel and consequently increases job makespan. Therefore, a procedure is designed to check the level of tasks assigned to VMs in each suggested solution before the value of the related objective functions is determined. If there are n tasks with the same level L_i assigned to one VM, this procedure will exchange $n-1$ of them with the subsequent tasks that are scheduled to other

VMs. The destination VMs for these *n-1* tasks should be chosen from a set of VMs that have tasks belonging to other levels of the DAG of job. This process will be repeated to fix the task levels in all VMs. For example, the suggested pattern will be modified as illustrated in Table 5.2.

Table 5.2: Modifying task scheduling pattern based on task levels in DAG of job

Tasks	t_0	t_1	t_2	t_3	t_4	t_5	t_6
Suggested Solution (SS)	*7 (vm_7)*	*7 (vm_7)*	*7 (vm_7)*	*5 (vm_5)*	*7 (vm_7)*	*4 (vm_4)*	*4 (vm_4)*
Modified Solution	*7 (vm_7)*	*7 (vm_7)*	*4 (vm_4)*	*5 (vm_5)*	*7 (vm_7)*	*4 (vm_4)*	*7 (vm_7)*

Procedure 5.1: Prevent tasks at the same level from being allocated to the same VM

Input: Suggested Solution$(SS) = \{ss_i, i = 1, .., n \mid ss_i$is the index of the VM that task i is scheduled to$\}$,

$T_{set}^{l} = \{t_1, \dots, t_n\}$, $L = \{L_i, i = 1, .., b | L_i$is the level number in the DAG of job$\}$

1. Determine a set of tasks in each level of job as $LT = \{lt_i, i = 1, \dots, p \mid lt_i$is a set of tasks in level $i\}$
2. Determine a set of tasks scheduled to each VM in the suggested solution as $ST_{set}^{k} = \{t_i, i = 1, \dots, q \mid t_i \epsilon T_{set}^{l}$ and is scheduled to $VM_k\}$
3. For each task, specify a set of tasks that are at the same level as this task and are also scheduled to the same VM as $Mt_i = \{t_1, \dots, t_p | \forall t_z, t_j : L_z = L_j$ and $ss_z = ss_j\}$
4. For each t_j in Mt_i find a $VM_k \in VM_{set}^{l}$ where $\forall t_z \in ST_{set}^{k} : L_z \neq L_j$

The second problem occurs when the sequence/priority of tasks in the DAG of their corresponding job does not match with the suggested pattern. The second procedure is developed to solve this problem by changing the task sequence if there are scheduled tasks with wrong priorities in a VM. For instance if tasks t_0, t_6 and t_2 are scheduled to VM_1, this schedule will be converted to t_0, t_2 and t_6. Considering the fact that changing the priority of scheduled tasks in each VM will not change the value of the optimal objective functions, this procedure will be carried out after the best possible scheduling solution has been suggested by the PSO algorithm.

Procedure 5.2: Modify task sequence in each VM based on task priorities

Input: SS, T_{set}^{l}, and L

1. Determine task paths in the DAG of job as $TP = \{tp_i, i = 1, \dots, p |\ tp_i \text{ is a task path}\}$
2. For each $VM_k \in VM_{set}^{l}$ if there are $t_z, t_q \in ST_{set}^{k}$ and $t_z, t_q \in tp_i$ then match the sequence of t_z and t_q with their priorities.

These procedures are applied to develop the MOTS-PSO algorithm. The first procedure is also considered as a condition in the multi-objective problem.

5.6 The Multi-Objective Problem

In the suggested MOTS optimization model, the data files required by the task and the priorities of the tasks are also considered to determine the best scheduling pattern. Computationally intensive tasks are assumed to require such light data files that file transfer time could be neglected. However, for computationally intensive tasks with large output file size, and for data intensive tasks, the data required by the task is not considered as a separate item from the task to be scheduled. In this case, the summation of the task file, its input data files, and the output data files of its parent task are considered in calculating the task transfer time (see Equations 5.3 and 5.4). In addition, Procedure 5.1 is considered as a condition in the multi-objective problem as follows where L_i is the level of t_i in the DAG of the correspond job:

The MOTS Problem:

$$min f_{TransferTime} = Ttrans \tag{5.34}$$

$$min\ f_{Excecution} = Cexe \tag{5.35}$$

$$min\ f_{PowerCunsumption} = PowerC \tag{5.36}$$

$$min\ f_{TaskQueue} = \Gamma \tag{5.37}$$

Subject to

$$\sum_{k=1}^{m} x_{ik} = 1, \forall i = 1, \dots, n$$

$$x_{ik} \in \{0,1\}, \forall i = 1, .., n\ \&\ k = 1, \dots, m$$

$$0 < N_{aPM} \leq N_{PM}$$

$$N_{aCPU}^{k} \leq VM_{ac}^{k} \leq VM_{c}^{k}$$
$$VM_{am}^{k} \leq VM_{m}^{k}$$
$$\forall \{t_x, t_y\} \in ST_{set}^{k} \subset T_{set}^{l},\ L_x \neq L_y$$

5.7 THE MOTS-PSO/GA ALGORITHM

This section explains the developed MOTS-PSO/GA algorithm that is designed to solve the formulated MOTS optimization problem presented in Section 5.6.

The MOTS-PSO/GA algorithm specifies the most appropriate VMs to which the arrival tasks or extra load of the poorly performing VMs can be allocated, and finds the optimal task scheduling schema by applying improved MOPSO and MOGA algorithms with original versions adopted from (Poli, Kennedy & Blackwell 2007) and (Zhang et al. 2011) respectively. This algorithm also applies the data and information determined in Table 5.1 as input variables.

It is assumed that in the task scheduling model, there are n tasks $\{t_1, t_2, \dots, t_n\}$ that should be assigned to m VMs $\{vm_1, vm_2, \dots, vm_m\}$ to be executed (see Table 5.3). MOPSO and MOGA methods are used to find the optimal task scheduling pattern, while minimizing task transfer time, task execution cost/time, power consumption and task queue length. All optimal suggested solutions (particle position/gene pattern) determined by MOPSO/MOGA techniques, are illustrated as $\vec{X}_{\mathrm{i}} = (x_1, x_2, \dots, x_n)$ vectors with continuous values, but their corresponding discrete values are needed to determine the index of the VM chosen for executing tasks. Therefore, these continuous vectors $\overrightarrow{X_{\mathrm{i}}}$ are converted to discrete vectors $d(\vec{X}_{\mathrm{i}}) = (d_1, d_2, \dots, d_n)$ by applying the Small Position Value (SPV) rule (Guo et al. 2012).

Table 5.3: **Task scheduling pattern**.

Tasks	t_1	t_2	t_3	t_4	t_5	…	t_n
VM number (Particle position/gene pattern)	vm_7	vm_4	vm_5	vm_7	vm_3	…	vm_m

The particle position (or gene pattern) in Table 5.3 are a possible solution, i.e., $d(\vec{X}_i) = (d_1, d_2, \ldots, d_n) = (7, 4, 5, 7, 3, .., m)$, after converting the continuous values to discrete values. According to this, VMs: $vm_7, vm_4, vm_5, vm_7, \ldots$, and vm_m are chosen to execute $t_1, t_2, t_3, t_4, \ldots$, and t_n respectively. Considering this fact, every particle/gene in our MOPSO/MOGA model has n dimensions to assign n tasks to m VMs. Every particle/gene will be assessed considering the predefined objective functions and all Pareto optimal solutions stored in an archive. In this study, it is assumed that the QoS obtained from each suggested solution can be estimated as:

$$QoS(\vec{X}_i) = -\sum_{j=1}^{q} W_j f_j(\vec{X}_i), \{\forall \vec{X}_i \in \text{Archive}\} \quad (5.38)$$

where q is the number of objective functions and W_j is the preference weight for every objective function $(f_j(\vec{X}_i))$. Pareto optimal solutions (archive members) are then ranked on the basis of the number of functions they minimize, and the maximum value of QoS. The top-ranking solution is chosen as the possible optimal solution $(\vec{X}_{gbest_i})$.

In summary, the following steps should be conducted by MOTS-PSO/GA algorithm:

Algorithm 5.1: The MOTS–PSO/GA Algorithm

Input: All variables in Table 5.1

[Begin MOTS-PSO/GA algorithm]

1. Collect data and information about a possible set of host VMs and set of arrival tasks

//Step 2 optimize the initialization of the first population

2. Initialize population: Determine new population (position and velocity of particles in MOPSO, or genes' pattern in MOGA) based on VM and task properties by applying roulette wheel technique as follows:
   - 2.1. Create the task and VM roulette wheels based on their properties by applying Equations 5.30 and 5.31
   - 2.2. $Count_{VM} = m$
   - 2.3. **For i=1 to n**
     - 2.3.1. Select a task (t_i) from task roulette wheel
     - 2.3.2. Select a VM (VM_j) from VM roulette wheel
     - 2.3.3. Delete t_i from the task set

2.3.4. Create the task roulette wheel based on new task set and their properties by applying Equation 5.31.

2.3.5. Calculate available VM memory and CPU by applying Equations 5.32 and 5.33

2.3.6. **If** $Available\_memory_{VM_j} \leq 0$ or $Available\_CPU_{VM_j} \leq 0$ then

{

Delete VM_j from VM set,

$Count_{VM} = Count_{VM} - 1,$

If $Count_{VM} = 0$ then //i.e. Current capacity of VMs are allocated to assigned tasks

{

$Count_{VM} = m$

Create new VM roulette wheel using original VM properties by applying Equation 5.30// Tasks will be assigned to task queue of these VMs

}

}

Else (if Step 2.3.6):

Create new VM roulette wheel —for $Count_{VM}$ number of VMs— based on available VM capacities by applying Equation 5.30

Next *i*

3. Initialize an archive in which members are non-dominated solutions (n dimensions particles/genes whose position/pattern is a Pareto optimal solution)

4. Determine the value of DO_i, DI_i, DF_i, VM_m^k, VM_c^k and VM_{bw}^k based on $d(\vec{X}_i)$ to calculate the value of every fitness function.

5. For each particle, calculate fitness functions $f_{TransferTime}$, $f_{Excecution}$, $f_{PowerConsumption}$ and $f_{TaskQueue}$ applying Equations 5.34, 5.35, 5.36 and 5.37.

6. For each particle, evaluate the desired optimization fitness functions.

7. Update the archive content by deleting dominated members from archive and store the Pareto optimal (non-dominated) solutions in the archive.

8. Sort archive members based on the number of minimized functions and the maximum value of $\mathrm{QoS}(\vec{\mathrm{X}})$

//Step 9 is conducted by MOTS-PSO

9. Produce new population using MOPSO:

9.1. Compare each particle's fitness evaluation with its personal best fitness function value $(\overrightarrow{x}_{pbest_i})$. If the current value is better than $\overrightarrow{x}_{pbest_i}$, then set $\vec{x}_{pbest_i}$ equal to the current value, and the best position *pi* equal to the current location $\vec{x}_i$ in *n*-dimensional space.

9.2. Identify the particle in the neighborhood with the best global success from top sorted members in the archive so far as $\vec{x}_{gbest_i}$, and assign its index to the variable *g* as the best global position.

9.3. Compute inertia weight and learning factors

9.4. Compute the velocity and position of the particle according to Equations 5.1 and 5.2.

//Step 10 is conducted by MOTS-GA

10. Produce new population using MOGA:

10.1. Reproduce best individuals using crossover and mutation.

11. Convert continuous position values vector of $\vec{X}_i$ to discrete vector $d(\vec{X}_i)$ using SPV rule to determine allocated VM for every arrival task.

12. **If** a criterion is met (usually a sufficiently good fitness or a maximum number of iterations) then

12.1. If tasks are dependent, modify the suggested solution based on the task level in the DAG of job by applying **Procedure 5.1**.

12.2. If tasks are dependent, modify the task priorities in each VM by applying **Procedure 5.2**.

12.3. **Output** the best particle position in n-dimensional space $d(\vec{X}_{gbest})$ for applied evolutionary algorithm (MOPSO or MOGA) as its best task scheduling patterns

Else (if Step12)

12.4. Go to Step 4

13. Calculate and update the current VM properties according to the optimal task scheduling solution.

14. Transfer tasks and their corresponding data to the destination VMs

[End MOTS-PSO/GA algorithm]

5.8 EVALUATION

This section analyzes the efficiency of the proposed model. In Section 5.8.1 the simulation environment is firstly determined. Then, in Section 5.8.2 it is explained how the CloudSim package (Calheiros et al. 2011) is extended to implement the method, and lastly, the performance and evaluation is presented in Section 5.8.3.

5.8.1 ENVIRONMENT DESCRIPTION

The simulation environment is designed by assuming that there are 15 PMs (data centers in CloudSim), 20 VMs, 3 cloud providers and 200 arrival bag-of-tasks. Data and information about VMs and tasks properties —that are determined based on required specifications by the cloud simulator— are summarized in Tables 5.4 and 5.5:

Table 5.4: **Properties of VMs**.

VM Id	CPU speed in GHz ($VM_{CPUSpeed}$)	Available memory in MB (VM_m)	Bandwidth in Mb/s (VM_{bw})	Number of CPUs (VM_c)	VMM name
1-4	2.6	512	1024	4	Xen
5-8	1.3	256	512	1	Xen
9-12	2.6	512	1024	2	Xen
13-16	1.3	256	512	1	Xen
17-20	1.3	256	512	1	Xen

Table 5.5: **Properties of tasks**.

Task Id	File Size in kB (*DF*)	Output Size in Byte (*DO*)	Input size in MB (*DI*)	Required CPUs (t_c)	CPU usage in GHz (t_{cu})	Max level of memory usage in MB (t_m)
1-20	87	190	1.5	1	65.216	63.800
21-40	85	46	0	1	21.186	42.512
41-60	87	190	1.5	1	65.216	63.800
61-80	85	46	0	1	21.186	42.512
81-100	87	190	1.5	1	65.216	63.800
101-120	87	190	1.5	1	65.216	63.800
121-140	85	46	0	1	21.186	42.512
141-160	87	190	1.5	1	65.216	63.800
161-180	87	190	1.5	1	65.216	63.800
181-200	85	46	0	1	21.186	42.512

5.8.2 IMPLEMENTATION

To implement the proposed method, first the open source Jswarm package (Cingolani 2009) is extended to support multi-objectives problems by converting the PSO algorithm to MOPSO algorithm. To achieve this goal, the evaluation method in Swarm class is modified by adding four new functions to: (1) determine non-dominated (Pareto optimal) solutions, (2) insert non-dominated solutions in the archive, (3) determine the dominated solutions in the archive, and (4) update achieve. In the first function, for every iteration, the positions of the particles (possible solutions) are assessed considering all the fitness functions (objectives) and the non-dominated solutions are determined, then they are inserted in the archive by applying the second function. Non-dominated solutions in the archive are assessed by the third function to find dominated solutions in the archive, and in the fourth function, dominated solutions in the archive are deleted. The archive members are then sorted. The methods to calculate the velocity and position that are applied in ParticleUpdate class, are also modified. The Particle, Neighborhood, SwarmRepulsive,

VariableUpdate classes are also made compatible with the new multi-objective calculations. The new produced package is called MO-Jswarm.

In the developed MOTS-PSO/GA, MO-Jswarm is applied as the foundation of MOTS-PSO algorithm (Steps 3 to 9), and NSGA-II (Hadka 2014) is applied to developed MOTS-GA algorithm (Step 10).

After creating and modifying the related packages to implement MOTS-PSO/GA algorithms, the CloudSim package is also extended to by using the MOTS-PSO and MOTS-GA algorithms as the task scheduling optimization algorithms for the simulation environment. The bindCloudletToVm() function in the DatacenterBrocker class of CloudSim is responsible for allocating tasks (cloudlets) to VMs according to the optimal task arrangement that results from the developed MOTS-PSO/GA algorithms.

The objective functions in the proposed multi-objective task scheduling model are applied as the fitness functions in MOTS-PSO and MOTS-GA. In our model, there are 200 particles and the optimal results are obtained after the 2000th iteration of the applied evolutionary algorithms.

5.8.3 Evaluation and Results Analysis

To evaluate the proposed method, the simulation is performed under the environment that is defined in Section 5.8.1. The proposed multi-objective method of solving task scheduling problems with conflicting objectives is then evaluated by considering optimization time, cost, power consumption and workload from the following aspects:

- Compare the efficiency of the proposed four-objectives model with current bi-objective models in terms of optimizing cloud utilization, QoS and Job makespan.
- Compare the efficiency of MOPSO and MOGA in speed and reliability to find the highest value of QoS in different iterations.

To make the first comparison, the method is compared with the optimization methods proposed in previous works (e.g. Guo et al. (2012) and Liu et al. (2012)) in which just two aspects of QoS optimization were considered: (1) task transfer time, and (2) execution cost.

The VM workload situation is also considered in this study to avoid multiple tasks being assigned to one VM, reducing its performance and increasing service response time. The optimization model also has the objective of reducing the number of active PMs and busy CPUs to decrease power consumption and providers' costs.

The graphs for task transfer time ($f_{TransferTime}$), task execution time/cost ($f_{Excecution}$), power consumption ratio ($f_{PowerConsumption}$), and VM task queue length ($f_{TaskQueue}$) obtained from the MOTS-PSO/GA algorithm in 2000 iterations are illustrated in Figures 5.5 and 5.6. The values of the axis on the right show the range of values of $f_{TaskQueue}$.

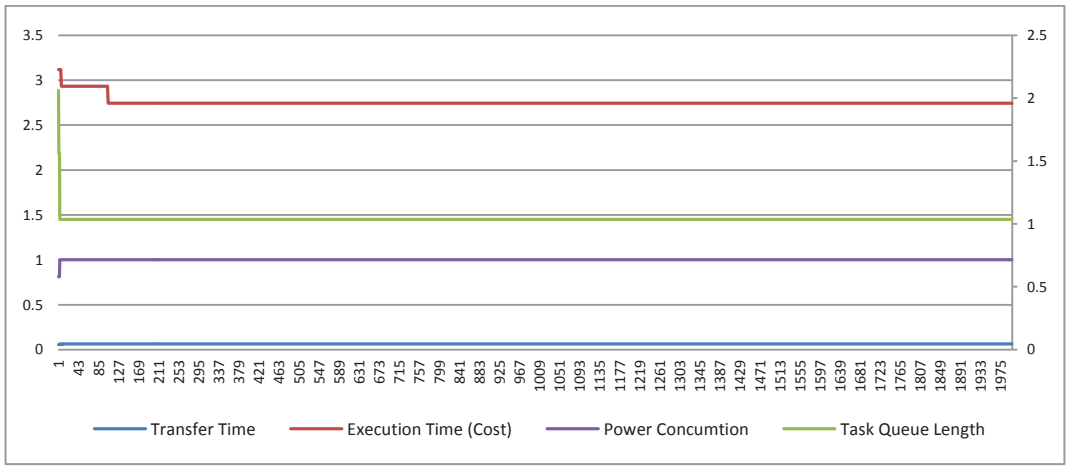


Figure 5.5: The value of objective functions using MOPSO

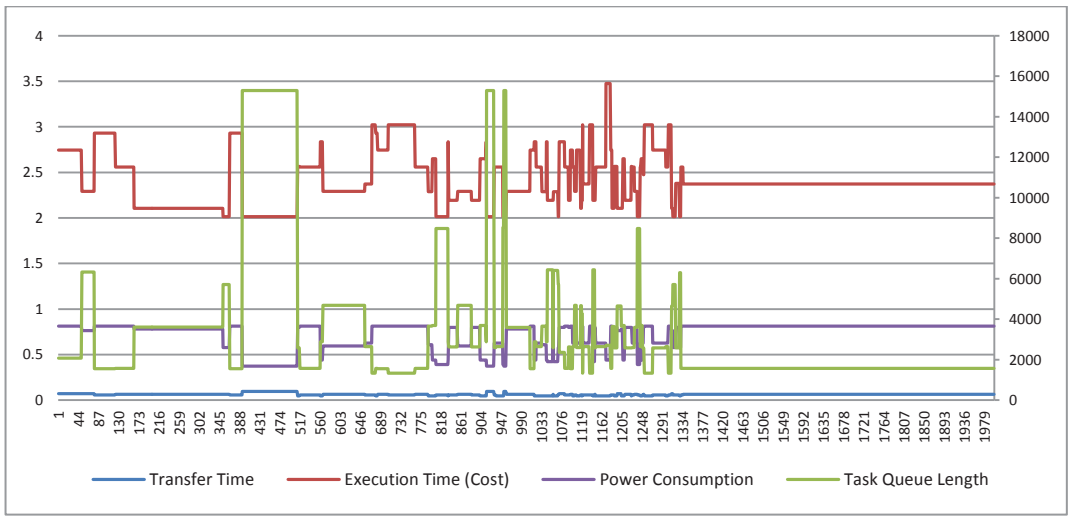


Figure 5.6: The value of objective functions using MOGA

To estimate the QoS that results from every method, Equation 5.39 is utilized by assuming the same preference weight ($w_1= w_2= w_3=1$) for transfer time, execution cost and power consumption. The preference weight ($w_4=10^{-3}$) or ($w_4=1$) is assigned for a task queue determinator function based on its value. As discussed in Section 5.4.4, a small value -10^{-3} is multiplied by RVM_m^k or RVM_c^k to increase the value of γ_k when the required capacity by arrival tasks exceeds the amount of available capacity in the target VM. This prevents sending multiple tasks to a single VM. Therefore, in cases where a suggested possible optimal task scheduling pattern assigns multiple tasks to certain VMs, the capacity required for executing the tasks assigned to those VMs would exceed the capacity of the VMs, then the value of $f_{TaskQueue}$ would be more than 10^3. In these situations $w_4=10^{-3}$ will be used in Equation 5.39, otherwise $w_4=1$. The value 10^{-3} for preference weight of $f_{TaskQueue}$ is determined as an aligner to make its value coherent with the other objective values and neutralize the effect of 10^{-3} coefficient that would be used in the denominator of Equation 5.26. As the optimal values of these conflicting objectives are independent of one another, there is no need to normalize their weights in the QoS equation (Equation 5.38). However, as objective functions have different measurement units, it is needed to normalize their values to be able to calculate the QoS. To do this, the range of objective functions are converted to (0,1). For instance, the maximum and minimum values for the objective function $f_{TaskQueue}$ produced by MOPSO and MOGA for four-objective model are 1.035 and 15284.7 respectively (see Figures 5.5 and 5.6). Therefore, $f_{TaskQueue}$ ranging (1.035, 1.035) is converted to (0,1). As a result, the normalized value of $f_{TaskQueue}$ i.e. $Nf_{TaskQueue}$ for MOGA is 0.1. The normalized value for all objective functions is calculated as $Nf_{TransferTime}$, $Nf_{Excecution}$, $Nf_{TaskQueue}$ and $Nf_{PowerConsumption}$ for all scenarios to determine the estimated value of QoS. In addition, considering the fact that this model minimizes the objective functions that have a negative impact on QoS, the resulted QoS is evaluated by applying the following equation:

$$QoS(\vec{X}) = -\left[w_1 * Nf_{TransferTime} + w_2 * Nf_{Excecution} + w_3 * Nf_{PowerConsumption} + w_4 * Nf_{TaskQueue}\right] \quad (5.39)$$

MOPSO and MOGA are applied to optimize both four-objective and bi-objective models. In a bi-objective model, the optimal value of $f_{TransferTime}$ and $f_{Excecution}$ is determined by MOPSO and MOGA, then the corresponding value of $f_{TaskQueue}$and $f_{PowerConsumption}$ is calculated by applying the optimal pattern of distribution tasks over VMs that results from these optimization algorithms. The algorithms have been run 25 times for each comparison section, and the results are almost the same. The optimal results of all methods are summarized in Table 5.6.

Table 5.6: Comparison results

Model	Algorithms	Transfer Time (s)	Execution Time (hrs) (Cost)	Power Consumption Ratio	Task Queue Length Coefficient	Estimated QoS	Job Makespan (hrs)	Iteration	Number of Idle VMs
4 objectives	MOPSO	0.063	2.74 (2:44:24)	1	1.03	-3.8	0.9 (0:54:00)	106	0%
	MOGA	0.063	2.37 (2:22:12)	0.81	1570.2	-4.8	1.1 (1:06:00)	1337	20%
2 objectives	MOPSO	0.043	2.01 (2:06:00)	0.39	8494.1	-10.94	1.74 (1:44:24)	58	60%
	MOGA	0.043	2.01 (2:06:00)	0.39	8494.1	-10.94	1.74 (1:44:24)	545	60%

As can be seen from the comparison results in Table 5.6, the estimated QoS in the proposed model with four objectives achieves the highest QoS compared to bi-objective models that apply both MOPSO and MOGA. In addition, in the four objective models,

the MOPSO determines the highest QoS in the lowest number of iterations (in 106th iteration) compared to MOGA, which determines its best possible solution in 1337th iteration. MOPSO in bi-objective models is also faster than MOGA. As can be seen, the optimal load balancing between resources determined by the four-objective model also has the shortest makespan. This means the model has the ability to offer a task scheduling pattern with the lowest response time. In addition, the four-objective models achieve lower task queue length than the bi-objective models. However, the suggested solution by the four-objective model consumes more power, because this model reduces the number of idle PMs, VMs and CPUs in the cluster in order to benefit from parallel task execution. This leads to reduce job makespan and optimize load balancing among recourses. In this case, preference weights of the conflicting objectives $f_{TaskQueue}$ and $f_{PowerConsumption}$ can be changed according to the agreement made between customer and provider to determine optimal task scheduling that satisfies the corresponded SLA criteria.

As a result, the proposed model that considers more aspects of task scheduling optimization, determines an optimal trade-off solution for the multi-objective task scheduling problem with objective functions that are in conflict with one another, and this results in the best possible compromise between objectives based on SLA, thereby increasing QoS.

The capability of the proposed objective functions are also examined in a real cloud environment as shown in Chapter 7.

5.9 SUMMARY

Optimizing task scheduling in a distributed heterogeneous computing environment, which is a nonlinear multi-objective NP-hard problem, plays a critical role in decreasing service response time and cost, and boosting QoS. This study considers four conflicting objectives, namely minimizing task transfer time, task execution time/cost, power consumption, and task queue length (VMs' workload), to develop a comprehensive multi-objective optimization model for task scheduling. This model reduces costs from both the

customer and provider perspectives by considering execution and power cost. A MOPSO/MOGA-based algorithm is also designed in this study to find the optimal solution for the proposed multi-objective task scheduling problem called MOTS-PSO/GA. To implement the algorithm, MO-Jswarm package is produced by modifying the original package to cover solving multi-objective problems. To evaluate this method the CloudSim toolkit is extended by applying MOTS-PSO/GA as its task scheduling algorithm. The optimal solution determined by MOTS-PSO/GA algorithm is applied by the bindCloudletToVm() function in the DatacenterBrocker class of CloudSim to assign tasks to VMs in an optimal way. The experimental results in the simulation environment show that the proposed optimization model has the ability to determine the best trade-off solutions compared to recent task scheduling approaches; it provides the best possible compromise between objectives which significantly reduces the job response time and makespan, achieves the highest QoS, and decreases the cost to providers. The experimental results also show that MOPSO is a faster and more accurate evolutionary algorithm than MOGA for solving such problems. It not only determines the optimal task scheduling pattern with highest QoS, but also obtains the solution in the shortest possible time. The multi-objective model could be made part of the virtualization layer. This would enable data center operators to make use of this model for optimal load balancing. It would also give cloud providers the opportunity to boost their benefits by optimizing their preferred goals based on their determined objective weights in a unique optimization model. They also have the ability to conduct sensitivity analysis on the value of optimized objective functions by changing their preferred weights. This sensitivity analysis process helps them to find the model with the highest level of benefit.

Chapter 6.

FUZZY PREDICTABLE TASK BASED SYSTEM LOAD BALANCING SUB-SYSTEM

6.1 INTRODUCTION

Cloud computing delivers scalable on-demand services include SaaS, PaaS, and IaaS over the Internet. A cloud provides the IaaS, PaaS, and/or SaaS through its own virtualized resources which are created over its underlying physical resources. Typically, a cloud virtualized resource is a set of specification and configuration files, called a VM (Buyya, Broberg & Goscinski 2011; Celesti et al. 2012).

In uncertain and complex environment of a cloud, VMs' workloads alternate with unsteady customer demands which lead to fluctuation in VMs resource utilization and imbalanced load over physical resources. VM migration —that is the process of moving a VM from an overloaded PM to another with more capacity— has been a solution for system load balancing and optimizing resource utilization. VM migration enables flexible resource allocation and increases the computation power and communication capability in a cloud environment. However, this process costs and is memory- and time-consuming for

large size VMs, and has the risk of losing the most recent activities of cloud customers during the VM migration process.

To deal with these challenges, this chapter presents a Fuzzy Predictable Task-based System Load Balancing (FP-TBSLB) approach that eliminates the need for VM migration and balances the load by employing the idle physical and virtual resources in cloud clusters to collaborate in executing extra workload.

The proposed approach is implemented in CloudSim (Calheiros et al. 2011) that is a cloud simulator toolkit to evaluate the proposed method. The simulation results indicate that the proposed model has significant influence on reducing time and memory consumption by the process of load balancing, and consequently reduces cost and service response time. As a result a higher QoS is expected by applying this approach.

The rest of this paper is organized as follows. In Sections 6.2 and 6.3, FP-TBSLB approach and sub-system are presented respectively. The FP-TBSLB algorithm is proposed in Section 6.4. The model is evaluated in Section 6.5. Lastly, this chapter is summarized in Section6.6.

6.2 The FP-TBSLB Approach

Due to the dynamic nature of cloud environments, the workload of VMs fluctuates dynamically, leading to imbalanced loads and utilization of virtual and physical resources of cloud clusters. VM migration is used in this situation to relax the workload by moving the VM from overloaded to under-loaded PM. In addition, VM migration is applied when IaaS customers request to scale up their assigned VMs, while the original host PM has not any idle resources available (Sallam & Li 2014).

VM migration which is done by suspend/resume or live migration strategy, is the process of copying the complete state of a VM from one PM to another for stronger computation power, larger memory, fast communication capability, or energy saving (Jin et al. 2011). This part of this study is constructed based on this believe that if a VM is small, it would be reasonable to migrate it to a new physical host. However, when the VM is large,

VM migration is not the optimal solution. Live VM migration process results in dirty memory as a result of the pre-copy process, utilizes a large amount of memory in the primary PM and new host PM, causes the VM to slow down during the migration process and carries the risk of losing last customer activities. These issues get worse in live storage migration when a VM changes its original host PM and storage which means its disk files also need to be migrated. Therefore, although cloud customers rarely notice these issues, VM migration is cost- and time- consuming for cloud providers. The resume/suspend migration strategy not only has live migration shortcomings but also causes lengthy VM downtime and lower QoS. Add security issue from chapter 2 section 2.13

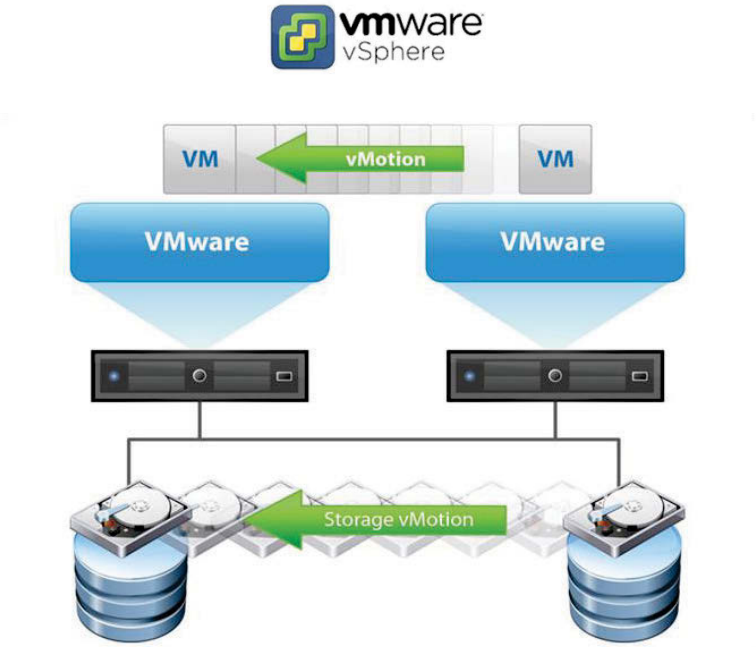


Figure 6.1: VM migration using vMotion

In this study, a Fuzzy Predictable Task-based System Load Balancing (FP-TBSLB) approach is proposed to overcome these shortcomings. The FP-TBSLB achieves system load balancing by migrating tasks from poorly performing VMs on an overloaded PM to a set of determined compatible VM instances located on other under-loaded PMs, instead of migrating VMs in their entirety. A compatible VM is a VM with the same OS as the

poorly performing VM, and has the required application/software stack to execute the scheduled tasks (see Figure 6.2).

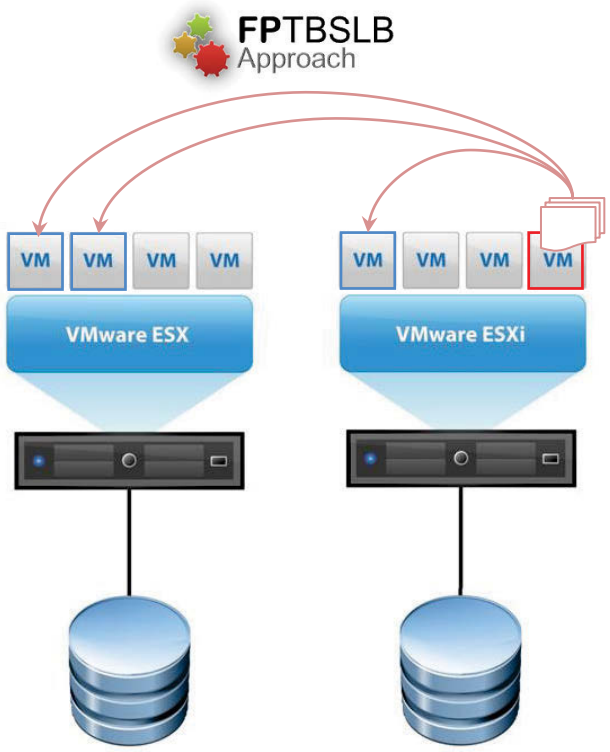


Figure 6.2: Fuzzy predictable task based system load balancing approach

In addition, this study considers the fact that cloud providers need to solve the problem of poorly performing VMs in the shortest possible time to satisfy the Service Level Agreement (SLA). Therefore, this model also has a VM's workload prediction model to predict PM hotspots before low performance occurs. This prediction model also has the ability to predict a set of compatible under-loaded VMs that can execute the tasks that have accumulated in the task queues of the possible poorly performing VMs. This helps the hypervisor layer to make an accurate decision and solve the problem before it arises. This model also provides a solution to distribute these tasks from the poorly performing VMs to the set of under-loaded VMs with minimum task transfer time, task execution cost, power consumption, and the length of task queues (see Figure 6.3).

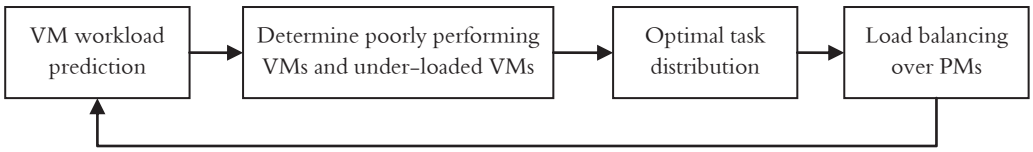


Figure 6.3: FP-TBSLB overview

Considering the fact that a poorly performing VM —irrespective of whether the VM is located on overloaded or under-loaded PM— leads to longer response time and lower Quality of Service (QoS), this study also suggests the same solution to enhance the performance of VMs with maximum resource utilization (overloaded VMs) that are located on under-loaded PMs.

6.3 THE FP-TBSLB SUB-SYSTEM

The FP-TBSLB sub-system has three main parts that communicate with each other and report to the FP-TBSLB decision center to conduct load balancing:

- Global Blackboard (GB)
- A fuzzy Workload Prediction (WP) sub-system by applying ES&NN-WP
- Central Task Scheduler (CTS) using MOTS-PSO algorithm of MOTS sub-system

The first part (GB) is a blackboard on which all VMMs and task schedulers share their data and information about VM features, jobs and scheduled tasks in a cluster. This data includes the number of CPUs, free memory and bandwidth allocated to VMs. Additional information about the VMs provided for SaaS and PaaS includes: the number of tasks to be executed, the task execution time, and the resources required for the tasks (the number of required processors, CPU and memory usage). In addition, this blackboard contains online information about PMs (physical resources), such as the number and speed of their processors (CPUs), the amount of free memory and hard disk they have, their situation (idle or active), and their associated VMM. All the information about SLA constraints are also gathered on this blackboard to monitor the QoS criteria. The stored variables on GB are also summarized in Table 5.1.

The second part of the FP-TBSLB sub-system is a fuzzy Workload Prediction (WP) model. The WP model applies blackboard data and information related to the VMs' workload situation and their CPU usage to predict which VMs are likely to meet their maximum utilization and exhibit low performance. This method also determines a set of compatible high performance VMs that can be used to execute the extra workload imposed on the poorly performing VM. The ES&NN-WP model that has been described in Chapter 4 is applied in WP sub-system that is employed by FP-TBSLB sub-system.

Lastly, the CTS is proposed as the third main part of the FP-TBSLB sub-system. The CTS applies all the information created by the other two parts of the sub-system (GB and WP) to determine the optimal pattern for scheduling extra tasks from the poorly performing VM to a set of compatible VMs that are determined. The optimal scheduling pattern will be discovered by a developed Multi-Objective Task Scheduling model applying PSO (MOTS-PSO) that considers the minimization of task transfer time, task execution cost, power consumption in the corresponding data center, and task queue length in the destination VMs. The CTS then transfers those tasks and their corresponding data to the pre-determined set of VMs based on an optimal pattern suggested by the MOTS-PSO algorithm. The MOTS-PSO algorithm has been described in Chapter 5. The PSO algorithm is chosen to be applied by MOTS sub-system that is employed by FP-TBSLB sub-system because it has been found faster and more reliable than GA (see Chapter 5.8).

The proposed FP-TBSLB sub-system is illustrated in Figure 6.4.

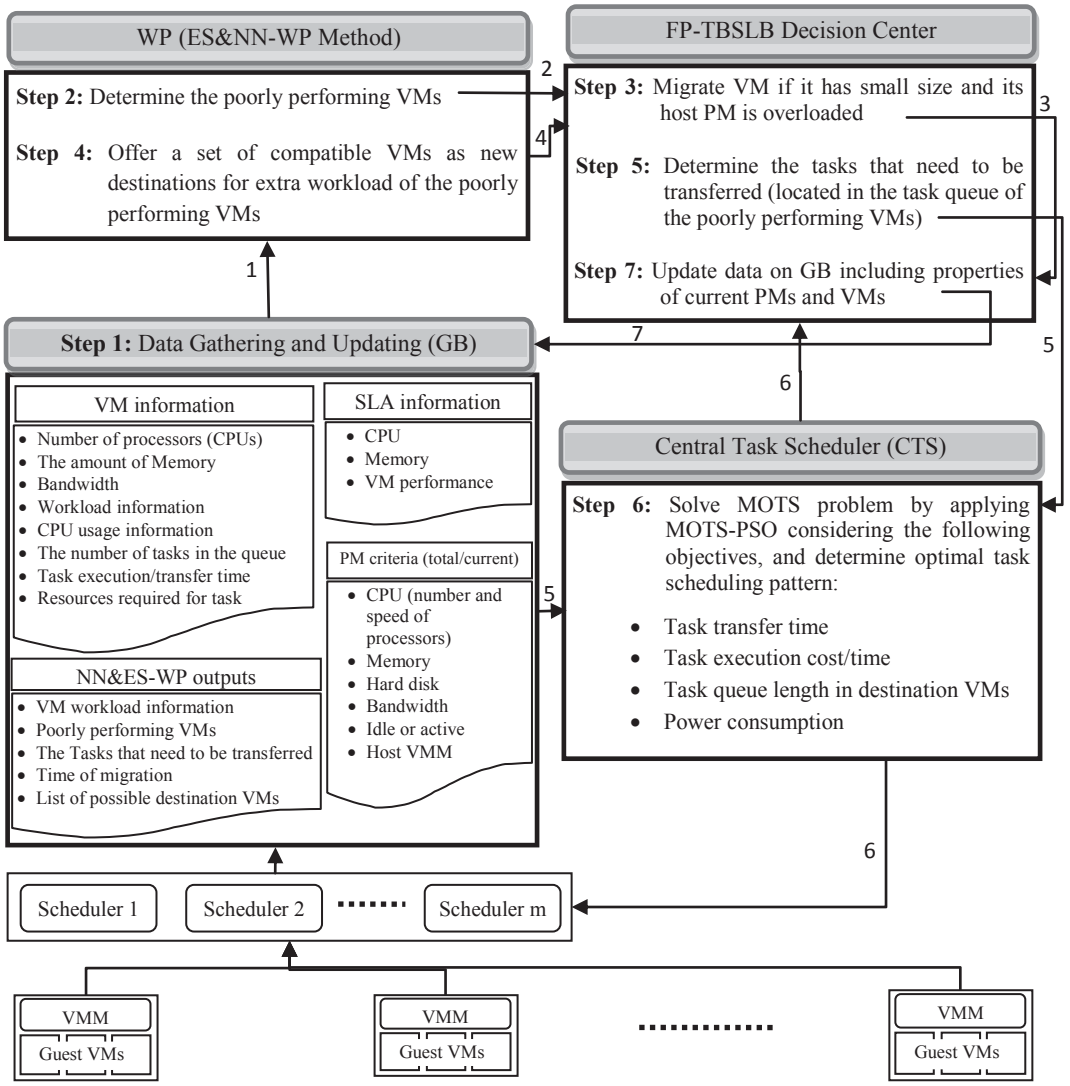


Figure 6.4: The FP-TBSLB sub-system

6.4 THE FP-TBSLB ALGORITHM

In this stage, the output variables of WP sub-system (using ES&NN-WP algorithm) are generated as part of input variables for MOTS-PSO algorithm of MOTS sub-system. These input variables are a set of poorly performing VMs, and a set of VMs as $VM_{set} = \{vm_1, \dots, vm_l\}$ which have available memory and CPUs for executing extra tasks. For each poorly performing VM (VM_l), a set of compatible VMs from VM_{set} should be chosen as new destinations for the extra workload (VM_{set}^l). The set of tasks to be transferred from VM_l are determined as $T_{set}^l = \{t_1, \dots, t_n\}$. Lastly, the MOTS-PSO algorithm is applied to

find the best solution for the multi-objective task scheduling optimization problem (see Section 5.4) to schedule tasks in T_{set}^{l} to VMs in VM_{set}^{l} and complete Step 8 of FP-TBSLB algorithm. The MOTS-PSO algorithm applies the data and information determined in Table 5.1, and the output variables obtained by Steps 1-7 of the FP-TBSLB algorithm as its input variables. Ultimately, the current value of VM properties (CPU, memory, etc.) will be updated on the GB. In summary, the following steps should be conducted by FP-TBSLB for each VM in the target cloud cluster:

Algorithm 6.1: The FP-TBSLB Algorithm

Input: All variables on the global blackboard (Summarized in Tables 4.2 and 5.1 applied by ES&NN-WP and MOTS-PSO respectively).

[Begin FP-TBSLB algorithm]

1. Monitor FP-TBSLB blackboard data to collect VM information including: VM tasks, memory and CPU usage, the amount of virtual resources (CPU and memory), etc.
2. Predict the CPU usage and workload status of VMs to determine poorly performing VMs by applying ES&NN-WP algorithm (WP sub-system).
3. Choose a VM from a set of poorly performing VMs suggested by ES&NN-WP algorithm (WP sub-system) as VM_l.
4. **If** VM has a small size and its host PM is also overloaded, then migrate it to new PM with available resources and go to Step 9.

   Else
5. **If** VM need OS and light installed applications **then** create new compatible VMs on other PM with available resources
6. Choose a set of compatible VMs as $VM_{set}^{l} = \{vm_1, \dots, vm_m\}$ form the set of candidate destination VMs ($VM_{set} = \{vm_1, \dots, vm_l\}$) determined by ES&NN-WP Algorithm (WP sub-system).
7. Determine the set of tasks accumulated in the task queue of each poorly performing VM which need to be transferred to new destination VMs as $T_{set}^{l} = \{t_1, \dots, t_n\}$
8. Determine the optimal task scheduling pattern to reschedule tasks (T_{set}^{l}) from poorly performing VM onto new determined destination VMs (VM_{set}^{l}) by applying MOTS_PSO algorithm (MOTS sub-system). minimizing task transfer time, task execution cost/time, length of VM task queue, and power consumption.
9. Transfer tasks and their corresponding data to the determined VMs.

10. Update the information of blackboard and schedulers including current properties of PMs and VMs.

[End FP-TBSLB algorithm]

6.5 EVALUATION

A system prototype is developed based on the proposed AS-ORM and two of its subsystems (WP using ES&NN-WP algorithm, and FP-TBSLB) are evaluated in a real cloud environment against the determined features in Chapter 7. However, in this chapter, the evaluation is presented by comparison of the proposed TBSLB approach with traditional load balancing approaches (VM migration) in a cloud simulation environment called CloudSim (Calheiros et al. 2011). The TBSLB approach is part of FP-TBSLB without WP sub-system. It has the same algorithm as FP-TBSLB algorithm that does not include Step 2.

6.5.1 ENVIRONMENT DESCRIPTION

The simulation environment is designed by assuming that there are three PMs (data centers), five VMs, and ten arrival tasks (cloudlets) to an overloaded VM in a cloud environment. Data and information about VMs and tasks that are applicable in the simulation environment of CloudSim are summarized in Tables 6.1 and 6.2. In addition, the MOTS-PSO algorithm in this model has 20 particles, and the optimal results are obtained after the 2000th iteration of the MOTS-PSO algorithm in MO-Jswarm.

Table 6.1: **VM properties**.

VM Id	CPU speed in GHz ($VM_{CPUSpeed}$)	Available memory in MB (VM_m)	Bandwidth in Mb/s (VM_{bw})	Number of CPUs (VM_c)	VMM name
1	2.6	4096	1024	4	Xen
2	2.6	4096	1024	2	Xen
3	1.3	2048	1024	2	Xen
4	1.3	1024	1024	1	Xen
5	1.3	512	1024	1	Xen

Table 6.2: Task properties.

Task Id	File Size in kB (*DF*)	Output Size in Byte (*DO*)	Input size in MB (*DI*)	Required CPUs (t_c)	CPU usage in GHz (t_{cu})	Max level of memory usage in MB (t_m)
1	8.7	47	0	1	186.372	125.828
2	8.5	46	0	1	21.186	62.912
3	8.5	46	0	1	21.186	62.912
4	8.5	46	0	1	8.261	73.4
5	8.5	47	0	1	21.759	62.912
6	8.5	46	0	1	21.186	62.912
7	9.2	170	0	1	41.097	943.716
8	8.5	46	0	1	21.186	62.912
9	8.5	46	0	1	21.186	62.912
10	8.5	46	1.4	1	2.833	128.825

6.5.2 IMPLEMENTATION

To evaluate proposed TBSLB approach, the CloudSim (Calheiros et al. 2011) toolkit has been extended by applying the MO-Jswarm package. CloudSim is a new, generalized, and extensible simulation sub-system that allows seamless modelling, simulation, and experimentation of emerging cloud computing infrastructures and application services. By using CloudSim, researchers and industry-based developers can test the performance of a newly developed application service in a controlled and easy to set-up environment. The bindCloudletToVm() function in the DatacenterBroker class of CloudSim is responsible for assigning tasks to VMs. MO-Jswarmhas the ability to determine the optimal task arrangement among VMs according to the PSO algorithm. In our proposed model, the bindCloudletToVm() assigns tasks to VMs according to the optimal results of MO-Jswarm.

6.5.3 EVALUATION AND RESULTS ANALYSIS

To achieve system load balancing in this situation according to TBSLB approach, first a set of eligible VMs that can host arrival tasks, has been determined by taking into account the information on the global blackboard. In the simulation environment, it has been assumed that five eligible VMs are found. The CloudSim is then applied to allocate extra tasks from the overloaded VMs to eligible VMs in an optimal way, to minimize task execution time, task transfer time, power consumption and the task queue length of each destination VM. In this stage, the MOTS-PSO algorithm —that is implemented by MO-

Jswarm— is used to determine this optimal pattern for scheduling these tasks over the eligible VMs to be executed. This pattern is the best particle position suggested by MOTS-PSO that is achieved according to Step 8 of the FP-TBSLB algorithm.

This optimal pattern is used by bindCloudletToVm() function in CloudSim. It takes less than 0.02 second to schedule these tasks by CloudSim. The algorithm has been run 25 times and the results are almost the same. The output results of CloudSim are illustrated in Table 6.3. Particle position in Table 6.3 is the optimal solution $d(\vec{X}_i) = (d_1, d_2, \ldots, d_{10})$= (2, 4, 5, 1, 1, 3, 2, 3, 1, 1) after converting the continuous position values to discrete. According to this solution, VMs: $vm_2, vm_4, vm_5, vm_1, vm_1, vm_3, vm_2, vm_3, vm_1$ and vm_1 are chosen to execute $t_1, t_2, t_3, \ldots,$ and t_{10} respectively.

Table 6.3: Simulation results.

Task Scheduling Pattern (CloudSim Outputs)										
Tasks	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
VM number = Particle position	vm_2	vm_4	vm_5	vm_1	vm_1	vm_3	vm_2	vm_3	vm_1	vm_1
Total task scheduling time <0.02 second										

Figure 6.5 illustrates the first 200 iterations of MOTS-PSO algorithm. The optimal values for the objective functions $f_{TransferTime}, f_{Execution}, f_{TaskQueue}$ and $f_{PowerConsumption}$ are determined in 127th iteration as 0.008 seconds, 1.95 hours, 16.8 and 1 respectively. The values on the right axis show the range of $f_{TaskQueue}$.

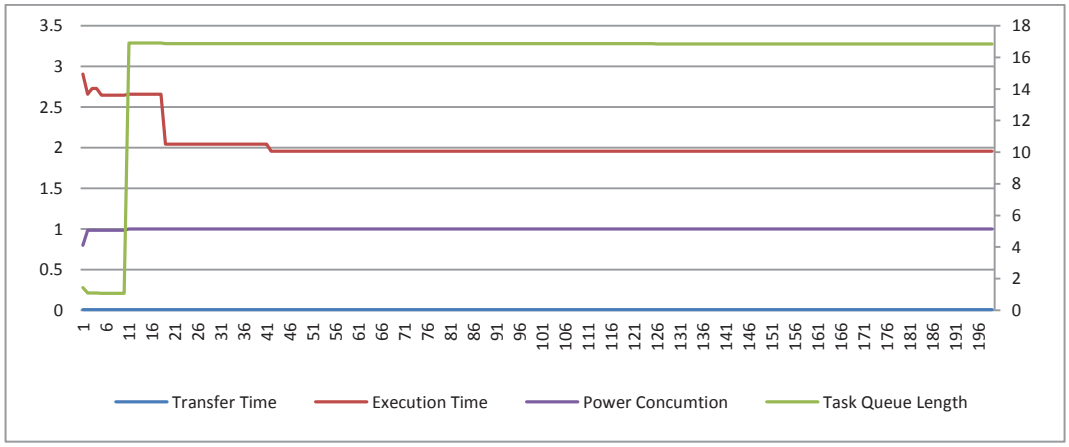


Figure 6.5: The optimal values for objective functions by MOTS-PSO algorithm

The implementation results are analysed based on two parameters: (1) the time consumption for load balancing, and (2) amount of produced idle memory.

(1) Load Balancing Time Consumption

A part of load balancing time consumption is equal to VM migration time and preparation time for determining new PM host. In offline and online VM migration the total migration time is equal to the time taken for migrating entire state of a VM. This time in TBSLB approach is reduced to the time spent for transferring some tasks from poorly performing VM.

The migration time mainly depends on the type of migration (host or storage migration), bandwidth capacity, and the VM size. Storage migration takes much more time than live host migration as the disk files of a VM need to be migrated. For instance, storage migration of a VM with 290 GB disk file size using 1024 Mb/s bandwidth capacity takes around 5 minutes. Live host migration for a small size VM by VMware vMotion takes around two seconds (Liu et al. 2014). In some cases, to achieve system load balancing by applying traditional system load balancing approaches, a new VM should first be set up to be used as a new host, and in a cloud environment, the setup time of a VM is typically around 5 to 15 minutes (Islam et al. 2012b).

However, as it can be seen from the results, system load balancing using TBSLB approach —regardless the size of the VM— takes less than 0.02 second, which is much less time consuming compare to all types of traditional system load balancing approaches. Therefore, if $T_{lb}(t)$ be the value of load balancing time:

$$T_{lb}(t)_{Offline\,VM} > T_{lb}(t)_{Online\,VM} > T_{lb}(t)_{FP-TBSLB\ Approach} \quad (6.1)$$

(2) Produced Idle Memory

To compare the efficiency of TBSLB approach, the amount of idle memory that is prepared during the load balancing process is defined as:

$$M_{im}(t) = PrimaryPM_m(t) + DestinationPM_m(t) \quad (6.2)$$

where $PrimaryPM_m$ and $DestinationPM_m$ are the amount of consumed memory in primary and destination PMs respectively.

The next process in both traditional and proposed approaches is the migration process. In offline VMs migration, the primary VM should be suspended during VM migration time, and the amount of the VM memory space on the original host PM and the amount of memory allocated to this VM on the destination host PM will be idle. In online VMs migration, although VM will not be suspended during migration process, the amount of memory in both the original and destination host PMs will be idle during the process of VM migration. In contrast, there is no VM migration in TBSLB approach, and only certain tasks should be migrated to destination VMs that have been determined as eligible. Therefore, the VM migration will consume much more memory than TBSLB approach during migration process and:

$$M_{im}(t)_{Offline\,VM} = M_{im}(t)_{Online\,VM} > M_{im}(t)_{TBSLB\ Approach} \quad (6.3)$$

In summary, the result obtain from implementing TBSLB approach in simulation environment shows that the proposed method will significantly reduce the time and memory consumption of the system load balancing process in the cloud environment. However, as has been mentioned before, the FP-TBSLB sub-system is also evaluated as part of AS-ORM in a real VMware-vSphere based cloud environment with VMware ESXi hypervisor in Chapter 7, and the results illustrate more advantages for this sub-system.

6.6 SUMMARY

VM migration has been applied to achieve system load balancing in the cloud environment by moving a poorly performing VM from one PM to another for stronger computation power, larger memory, fast communication capability, and/or energy savings. Although this approach reduces VM downtime and enables flexible resource allocation in a cloud environment, it is time- and cost-consuming. Additionally, a large amount of

memory is involved and there is the risk of losing the most recent activities of cloud customers during the VM migration process.

To address these shortcomings, the FP-TBSLB is proposed and TBSLB —that is FP-TBSLB sub-system without the prediction part— is validated in this chapter to achieve system load balancing in the cloud environment by migrating arrival tasks from a poorly performing VM to another homogeneous VM using MOTS sub-system, instead of migrating the VM in its entirely. The MOTS-PSO algorithm of MOTS sub-system is applied as part of the FP-TBSLB sub-system to find an optimal solution to the scheduling of tasks from poorly performing VMs to a set of new destination VMs. This optimization model has four conflicting objectives, namely: task transfer time, task execution cost/time, length of VM task queue, and power consumption.

In addition, the CloudSim package is extended and combined with the MO-Jswarm package to apply our PSO-based task scheduling model as the task scheduling algorithm in CloudSim to evaluate the TBSLB sub-system.

The simulation results show that the TBSLB approach significantly reduces the time consumption of the load balancing process compared to other traditional methods for load balancing. Furthermore, the proposed method reduces VM slowdown and memory usage because there is no need for the VM migration process and much less idle capacity in the primary and destination host PMs is required.

Chapter 7.

IMPLEMENTATION AND EVALUATION

7.1 INTRODUCTION

In this chapter the efficiency of the AS-ORM is evaluated in comparison with the VMware ESXi auto load balancing method by implementing them in a private cloud. This evaluation is done by investigating the performance of the AS-ORM sub-systems including: WP (ES&NN-WP), MOTS (MOTS_PSO) and FP-TBSLB, in a real environment. These sub-systems evaluation results have been shown in Chapters 4.5.5, 5.8 and 6.5 respectively.

Following on, the rest of this paper is organized as follows. The tools that are applied for evaluation tools are described in Sections 7.2. Section7.3 presents the description of the evaluation environment, followed by Section 7.4 that represents the scenarios under which the evaluation is done. In Section 7.5 the proposed system is implemented. Section 7.6 analyses the evaluation results and evaluation is summarized in Section 7.7.

7.2 EVALUATION TOOLS DESCRIPTION

This section describes a preliminary description of the application of hypervisor layer and tools in virtualized cloud cluster. The tools that are applied for implementing and

evaluating the AS-ORM and its sub-systems including VMware ESXi, VMware vMotion and Condor are then introduced.

Virtualization technology makes it very flexible and easy to manage resources in cloud computing environments, because they improve the utilization of such resources by multiplexing many VMs on one physical host (server consolidation). The virtualization layer will partition the physical resource of the underlying physical server into multiple VMs with different workloads (see Figure 7.1). The fascinating thing about this virtualization layer is that it schedules, allocates the physical resource, and makes each VM think that it totally owns the whole underlying hardware's physical resource (processor, disks, rams, etc.). These machines can be scaled up and down on demand with a high level of resources' abstraction. Virtualization enables high, reliable, and agile deployment mechanisms and management of services, providing on-demand cloning and live migration services which improve reliability. Accordingly, having an effective management suite for managing VMs' infrastructure is critical for any cloud computing infrastructure as a service (IaaS) vendor (Buyya, Broberg & Goscinski 2011).

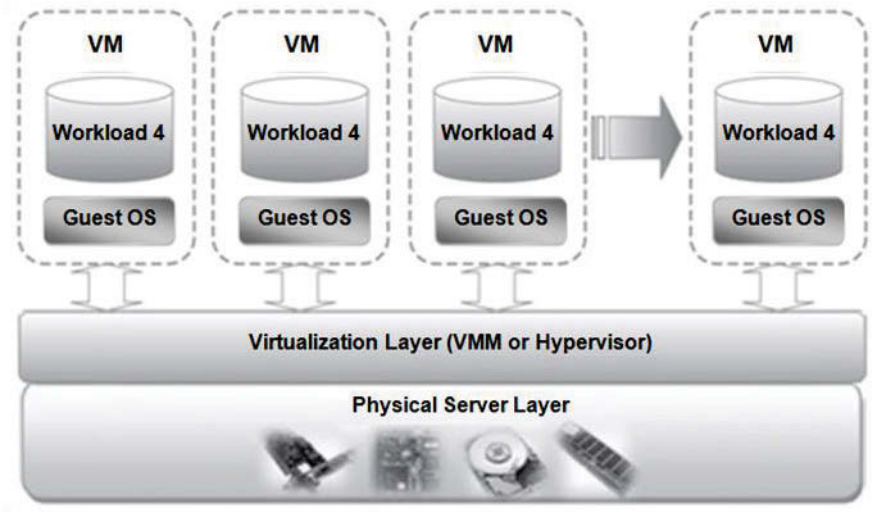


Figure 7.1: A layered virtualization technology architecture

VMware vSphere is VMware's cloud computing virtualization operating system that aims at transforming IT infrastructures into private clouds. It is distinguished from other Virtual Infrastructure Managers (VIMs) as one of the most feature-rich, due to the

company's several offerings in all levels the architecture. In the vSphere architecture, servers run on the ESXi hypervisor platform. A separate server runs vCenter Server, which centralizes control over the entire virtual infrastructure. Through the vSphere Client software, administrators connect to vCenter Server to perform various tasks. The Distributed Resource Scheduler (DRS) makes allocation decisions based on predefined rules and policies. It continuously monitors the amount of resources available to VMs and, if necessary, makes allocation changes to meet VM requirements. In the storage virtualization realm, vStorage Virtual Machine File System (VMFS) is a cluster file system to provide an aggregate of several disks in a single volume. VMFS is especially optimized to store VM images and virtual disks. It supports storage equipment that use Fibre Channel or iSCSI SAN. In its basic setup, vSphere is essentially a private administration suite. Self-service VM provisioning to end users is provided via the vCloud Application Programming Interface (API), which interfaces with vCenter Server. In this configuration, vSphere can be used by service providers to build public clouds. In terms of interfacing with public clouds, vSphere interfaces with the vCloud API, thus enabling cloud-bursting into external clouds (Buyya, Broberg & Goscinski 2011).

In general, vSphere provides the following features: Windows-based controller (vCenter Server); CLI, GUI, Web portal, and Web services interfaces; VMware ESX, ESXi backend; VMware vStorage VMFS storage virtualization; interface to external clouds (VMware vCloud partners); virtual networks (VMWare Distributed Switch); dynamic resource allocation (VMware DRM);high availability; data protection (VMWare Consolidated Backup) (Buyya, Broberg & Goscinski 2011).

7.2.1 VMware ESXi

The hypervisor architecture of VMware vSphere® 5.0 plays a critical role in the management of the virtual infrastructure. The introduction of the bare-metal VMware ESX® architecture in 2001 significantly enhanced performance and reliability, which in turn enabled customers to extend the benefits of virtualization to their mission-critical

applications. The introduction of the VMware ESXi™ architecture represents a similar leap forward in reliability and virtualization management. Less than five percent of the footprint of ESX, ESXi runs independently of a host operating system (OS) and improves hypervisor management in the areas of security, deployment and configuration, and ongoing administration (Wang, Qiu & Guo 2015).

In the original ESX architecture illustrated in Figure 7.2, the virtualization kernel (VMkernel) is augmented by a management partition known as the Console Operating System (COS) or service console. The primary purpose of the COS is to provide a management interface with the host. Various VMware® management agents are deployed in the COS, along with other infrastructure service agents (for example, name service, time service, logging, and so on). In this architecture, many customers deploy other agents from third parties to provide a particular functionality, such as hardware monitoring and systems management. Furthermore, individual administrative users log in to the COS to run configuration and diagnostic commands and scripts (Wang, Qiu & Guo 2015).

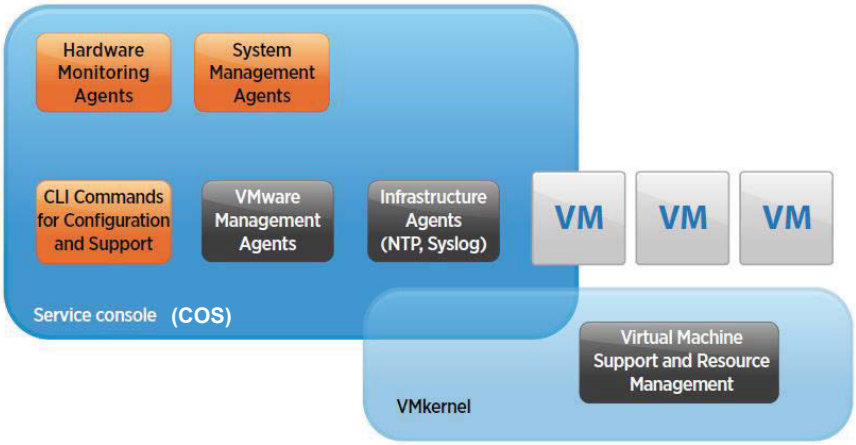


Figure 7.2: VMware EXS architecture

In the ESXi architecture, the COS has been removed, and all of the VMware agents run directly on the VMkernel (see Figure 7.3). Infrastructure services are provided natively through modules included in the VMkernel. Other authorized third party modules, such as hardware drivers and hardware monitoring components, can run in the VMkernel as well (Wang, Qiu & Guo 2015).

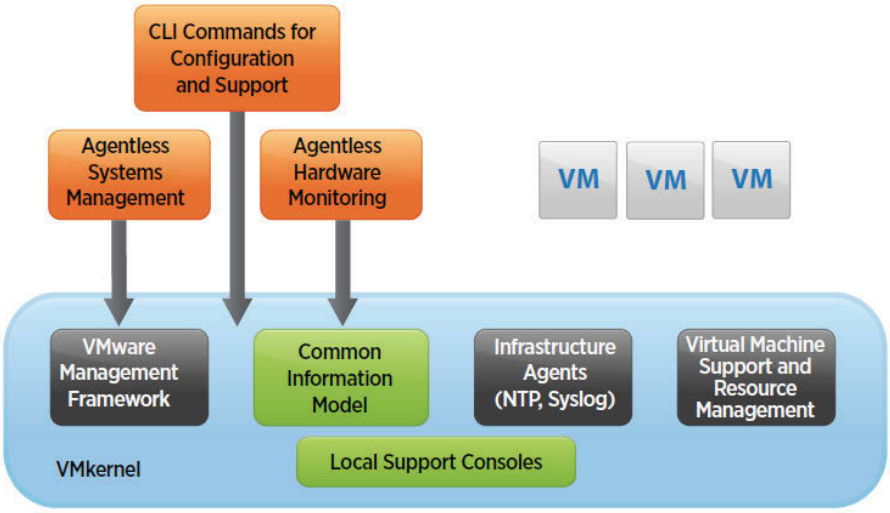


Figure 7.3: VMware EXSi architecture

7.2.2 VMware vMotion

VMware vSphere live migration allows you to move an entire running VM from one physical server to another, without downtime. The VM retains its network identity and connections, ensuring a seamless migration process. Transferring the VM's active memory and precising execution state over a high-speed network allow the VM to switch from running on the source vSphere host to the destination vSphere host. This entire process takes less than two seconds on a gigabit Ethernet network (Liu et al. 2014).

VMware vSphere® vMotion® enables the live migration of VMs from one VMware vSphere® 5 host to another, with no perceivable impact to the end user. vMotion is a key enabler of a number of VMware technologies, including vSphere Distributed Resource Scheduler (DRS) and vSphere Distributed Power Management (DPM). vMotion brings invaluable benefits to administrators —it helps prevent server downtime, enables troubleshooting and provides flexibility. Although vMotion has been used successfully since the earliest versions of VMware ESX®, vSphere 5 incorporates a number of performance enhancements to make it easier than ever to enable vMotion on even the largest VMs running heavy-duty, enterprise-class applications, with minimal overhead. (Antonescu & Braun 2016). VMware vMotion allows users to (a) automatically optimize and allocate an

entire pool of resources for maximum hardware utilization, flexibility, and availability and (b) perform hardware maintenance without scheduled downtime along with migrating VMs away from failing or underperforming servers (Buyya, Broberg & Goscinski 2011). It has the ability to:

- Automatically optimize VMs within resource pools.
- Perform hardware maintenance without scheduling downtime or disrupting business operations.
- Move VMs away from failing or underperforming servers.

7.2.3 HTCondor

HTCondor is a software system that creates a High-Throughput Computing (HTC) environment. It effectively utilizes the computing power of workstations that communicate over a network. HTCondor can manage a dedicated cluster of workstations. Its power comes from the ability to effectively harness non-dedicated, pre-existing resources under distributed ownership (Barati & Sharifian 2015).

When a user submitsa job to HTCondor, HTCondor finds an available machine on the network and begins running the job on that machine. HTCondor has the capability to detect that a machine running an HTCondor job is no longer available. It can checkpoint the job and move (migrate) the jobs to a different machine which would otherwise be idle. HTCondor continues the job on the new machine from precisely where it left off. In those cases where HTCondor can checkpoint and migrate a job, HTCondor makes it easy to maximize the number of machines which can run a job. In this case, there is no requirement for machines to share file systems, so that machines across an entire enterprise can run a job, including machines in different administrative domains. HTCondor can be a real time saver when a job must be run many (hundreds of) different times, perhaps with hundreds of different data sets. With one command, all of the hundreds of jobs are submitted to HTCondor. Depending upon the number of machines in the HTCondor

pool, dozens or even hundreds of otherwise idle machines can be running the job at any given moment.

HTCondor does not require an account (login) on machines where it runs a job. HTCondor can do this because of its remote system call technology, which traps library calls for such operations as reading or writing from disk files. The calls are transmitted over the network to be performed on the machine where the job was submitted. HTCondor provides powerful resource management by match-making resource owners with resource consumers. This is the cornerstone of a successful HTC environment. Other compute cluster resource management systems attach properties to the job queues themselves, resulting in user confusion over which queue to use as well as administrative hassle in constantly adding and editing queue properties to satisfy user demands. HTCondor implements ClassAds, a clean design that simplifies the user's submission of jobs.

ClassAds work in a fashion similar to the newspaper classified advertising want-ads. All machines in the HTCondor pool advertise their resource properties, both static and dynamic, such as available RAM memory, CPU type, CPU speed, virtual memory size, physical location, and current load average, in a resource offer ad. A user specifies a resource request ad when submitting a job. The request defines both the required and a desired set of properties of the resource to run the job. HTCondor acts as a broker by matching and ranking resource offer ads with resource request ads, making certain that all requirements in both ads are satisfied. During this match-making process, HTCondor also considers several layers of priority values: the priority the user assigned to the resource request ad, the priority of the user which submitted the ad, and the desire of machines in the pool to accept certain types of ads over others (Barati & Sharifian 2015)

In this study, the destination machines are determined based on optimal solution suggested by MOTS sub-system. The names of the determined machines are then specified in the submit file of their corresponding jobs. Using this, HTCondor sends the jobs to their specified machines.

7.3 Environment Description

The cloud environment has been designed by two data-stores, four PMs, twenty VMs, two cloud providers and 200 arrival computation, memory and data intensive tasks that are independent. The coefficients in Equation 5.5 are considered as $Pcost_1 = Pcost_2 = 1$ therefore objective function $f_{Execution}$ (Equations 5.12 and 5.35) is applied as an estimation for the task execution time. The information about VMs and tasks is summarized in Tables 7.1 and 7.2. The PMs are homogenous and each has five different VMs (see Table 7.3).

Table 7.1: Properties of VMs

VM Id	CPU speed in GHz ($VM_{CPUSpeed}$)	Available memory in MB (VM_m)	Bandwidth inMb/s (VM_{bw})	Number of CPUs (VM_c)	OS
1-4	2.6	4096	1024	4	Ubuntu Linux
5-8	2.6	4096	1024	2	Ubuntu Linux
9-12	1.3	2048	1024	2	Ubuntu Linux
13-16	1.3	1024	1024	1	Ubuntu Linux
17-20	1.3	512	1024	1	Ubuntu Linux

Table 7.2: Properties of tasks

Task Id	File Size in kB (DF)	Output Size in Byte (DO)	Input size in MB (DI)	Required CPUs (t_c)	CPU usage in GHz (t_{cu})	Total memory usage in MB	Max level of memory usage in MB (t_m)
Computationally intensive Tasks							
1-20	8.7	47	0	1	186.372	2055.06	125.828
21-40	8.5	46	0	1	21.186	985.616	62.912
41-60	8.5	47	0	1	67.166	754.924	62.912
61-80	8.5	46	0	1	8.261	471.848	73.4
81-100	8.5	47	0	1	21.759	524.26	62.912
101-120	8.5	46	0	1	2.263	125.82	41.94
Memory Intensive Tasks							
121-140	9.2	170	0	1	41.097	11534.304	943.716
141-160	9.2	169	0	1	38.367	9940.468	859.832
161-180	9.2	170	0	1	38.372	1992.268	167.772
Data Intensive Tasks							
181-200	8.5	46	1.4	1	2.833	377.484	125.828

Table 7.3: Resource allocation in the cluster

PM Id	VMs				
1	VM_1	VM_5	VM_9	VM_{13}	VM_{17}
2	VM_2	VM_6	VM_{10}	VM_{14}	VM_{18}
3	VM_3	VM_7	VM_{11}	VM_{15}	VM_{19}
4	VM_4	VM_8	VM_{12}	VM_{16}	VM_{20}

7.4 SCENARIOS

The proposed system is evaluated based on two different scenarios to cover all possible situations. These scenarios are explained as follows.

Scenario 1–A set of VMs in a host PM have low performance:

In the first scenario, the efficiency of the sub-systems of AS-ORM is evaluated in the situation of having several VMs located on a PM that are working with low performance due to high-utilization memory/CPU. This situation may arise for a set of VMs that are applied to deliver SaaS but which are struggling to finish their major scheduled tasks and also have a long scheduled task queue.

In this situation, schedulers stop sending more tasks to the poorly performing VMs, and the VMs continue to execute their current tasks and the tasks in their queue. This will create a longer makespan for the corresponding jobs to those tasks. In addition, this solution is applied after the VMs have started to work at low performance. However, the proposed solution is to predict which VMs are likely to be overloaded (work with their maximum capacity) to solve the problem before it occurs. Tasks that have accumulated in the task queue of the potentially poorly performing VMs will then be scheduled to other compatible VMs (i.e. VMs with the same OS and required applications) in a cluster that has the capacity to execute these tasks.

Scenario 2–A host PM is likely to be overloaded or a VM needs to be scaled up:

In the second scenario, the system is evaluated in cases when VM migration is required. This occurs when all the resources in the host are allocated to VMs, and all VMs are using

their maximum capacity to execute their allocated tasks. In this case —irrespective of whether the VMs are used for SaaS, PaaS or IaaS— the PM is at risk of being overloaded. The current solution for this problem is to migrate a set of VMs from the overloaded PM to other PMs with more available resources (Clark et al. 2005; Jin et al. 2011; Jun & Xiaowei 2011; Kozuch & Satyanarayanan 2002; Nelson, Lim & Hutchins 2005; Nicolae & Cappello 2013; Osman et al. 2002; Sallam & Li 2014; Sapuntzakis et al. 2002; Whitaker et al. 2004). This scenario also covers VMs that are delivered as IaaS when these VMs exhibit low performance and their client asks for VM scale-up while there is no available capacity in their host PM. In this case, these VMs will be migrated to other PMs that have the available capacity to allocate to these VMs and scale them up (Clark et al. 2005; Jin et al. 2011; Jun & Xiaowei 2011; Kozuch & Satyanarayanan 2002; Nelson, Lim & Hutchins 2005; Nicolae & Cappello 2013; Osman et al. 2002; Sallam & Li 2014; Sapuntzakis et al. 2002; Whitaker et al. 2004). In contrast, the solution of FP-TBSLB sub-system of AS-ORM for the first situation is to select a set of VMs located on the overloaded PM, stop them from working and transfer their tasks in progress and all tasks that have accumulated in their task queues to compatible VMs with available capacity located on other PMs. In addition, FP-TBSLB sub-system transfers all tasks that have accumulated in the task queue of the other poorly performing VMs located on this overloaded PM to a set of VMs located on other PMs with more available capacity. This reduces resource utilization on the overloaded PM and eases tension between its allocated VMs. For the second situation —where a VM is delivered as IaaS and needs to be scaled up— FP-TBSLB sub-system also applies the same process and transfers the pending and executing tasks of this VM to other compatible VMs with more available capacity.

Furthermore, the VMware auto load balancer will react after some of the VMs start to exhibit low performance and the PM utilizes more than the determined percentage of its capacity. However, the prediction model that is applied in the FP-TBSLB approach,

predicts VM performance and reacts before the corresponding PM meets the predefined utilization threshold.

In this scenario, the suggested solution is compared to both live and storage migration. In live migration, the execution state and active memory of a VM is migrated to a new ESX host. Live storage VM migration includes changing both the host and data store. In this case, the disk files of VM will also be migrated.

7.5 IMPLEMENTATION

To implement the sub-systems of AS-ORM, first the determined tasks are randomly scheduled to the VMs in the private cloud by applying Condor functionality for one week to create CPU usage historical data. The resulting data (VM^{k}_{Ucpu} that is defined in Table 4.2) is used along with VM^{k}_{c} and VM^{k}_{m} (that are defined in Table 5.1) as input historical data to train the designed NN and make it ready for prediction. The NN and ES architecture and ES&NN-WP algorithm have been implemented using MATLAB.

7.5.1 IMPLEMENTING THE AS-ORM IN THE FIRST SCENARIO

In the first scenario, the tasks listed in Table 7.2 are randomly scheduled to VMs, and the ES&NN-WP algorithm is implemented for every VM to check their performance every two minutes. According to the ES&NN-WP algorithm results, all VMs were under-loaded in the first round. In the second round (after four minutes), the ES&NN-WP algorithm results show that VM_{17} located on PM_1 will be overloaded, based on the value of its input variables. These variables are calculated using data that has been stored in the FP-TBSLB blackboard as follows, where $Ts = (2,4)$:

$$VM^{17}_{Ucpu}(ct) = 85\%\,,\ \ ct = 4$$

$$\underset{s_i \in lt}{\text{Avg}}\, VM^{17}_{Ucpu}(s_i) \geq 81\%, \qquad lt = (3,4)$$

$$fCA'^{17}_{cpu}(x)\ \geq 0$$

By applying these values in Equation 4.8, the value of $VM^{k}_{Huti}(Ts)$ indicates that VM_{17} has a high level of CPU utilization. In addition, $fCA'^{17}_{et}(x)\ \geq 0$ which means that VM_{17}

has a rising workload pattern and the value of $VM_{RisEt}^{17}(Ts)$ is high (see Equation 4.11). Furthermore, $PI^{17}(Ts)$ shows a decreasing trend during *Ts*, therefore the value of $PI_{Dec}^{17}(Ts)$ is also high (see Equation 4.13). The designed NN prediction results ($NN_R^{17}(Ts)$) also suggest that this VM had a high level of CPU usage during time slot *Ts* and will be overloaded and exhibit low performance (see Equation 4.14).

These results satisfy the first rule of the developed ES (see Table 4.4), therefore VM_{17} is suggested as being a poorly performing VM (see Equation 4.15). At this moment, $VM_{17}$was executingt_5 while it had $T_{set}^{17} = \{t_7, t_{21}, t_{45}, t_{66}, t_{181}, t_{119}, t_{126}, t_{158}, t_{183}, t_{198}\}$ in its task queue. ES&NN-WP also nominates $VM_2, VM_3, VM_5, VM_8, VM_9$ and VM_{15} as VMs that will be under-loaded. Of the under-loaded VMs, VM_3, VM_5, VM_9 and VM_{15} are compatible with VM_{17} and have been offered by the ES&NN-WP algorithm as VM_{set}^{17} , i.e., as possible destinations for tasks that have accumulated in the task queue of VM_{17} task queue. In addition, VM_3, VM_5, VM_9 and VM_{15} have 3, 1, 1 and 1 available CPUs respectively.

Table 7.4: The ES&NN-WP results

VM ID	$NN_R^k(Ts)$		$VM_{Huti}^k(Ts)$		$VM_{RisEt}^k(Ts)$		$PI_{Dec}^k(Ts)$		VM_{load}^k
17	O	and	H	and	H	and	H	then	Overloaded
2	U	and	L	and	L	and	L	then	Under-loaded
3	U	and	L	and	L	and	L	then	Under-loaded
5	U	and	L	and	L	and	H	then	Under-loaded
8	U	and	L	and	L	and	H	then	Under-loaded
9	O	and	L	and	L	and	L	then	Under-loaded
15	U	and	L	and	L	and	L	then	Under-loaded

After determining the poorly performing VM and a set of under-loaded VMs, the MOTS_PSO sub-system/algorithm schedules the tasks accumulated in the VM_{17} task queue (T_{set}^{17}) to VM_{set}^{17} members based on the MOTS-PSO optimal scheduling pattern as follows:

Table 7.5: The MOTS–PSO suggested solution for task scheduling

Tasks	t_7	t_{21}	t_{45}	t_{66}	t_{181}	t_{119}	t_{126}	t_{158}	t_{183}	t_{198}
Task CPU usage in GHz	186.372	21.186	67.166	8.261	2.833	2.263	41.097	38.367	2.833	2.833
VMs	VM_5	VM_3	VM_3	VM_3	VM_9	VM_3	VM_{15}	VM_9	VM_3	VM_3

The optimal values for the objective functions $f_{TransferTime}$, $f_{Execution}$, $f_{TaskQueue}$ and $f_{PowerConsumption}$ are 0.024 seconds, 2.19 hours (02:11:24), 3.18 and 1 respectively. The suggested optimal value of these objective functions in 2000 iterations of the MOTS-PSO algorithm of the MOTS sub-system is shown in Figure 7.4.

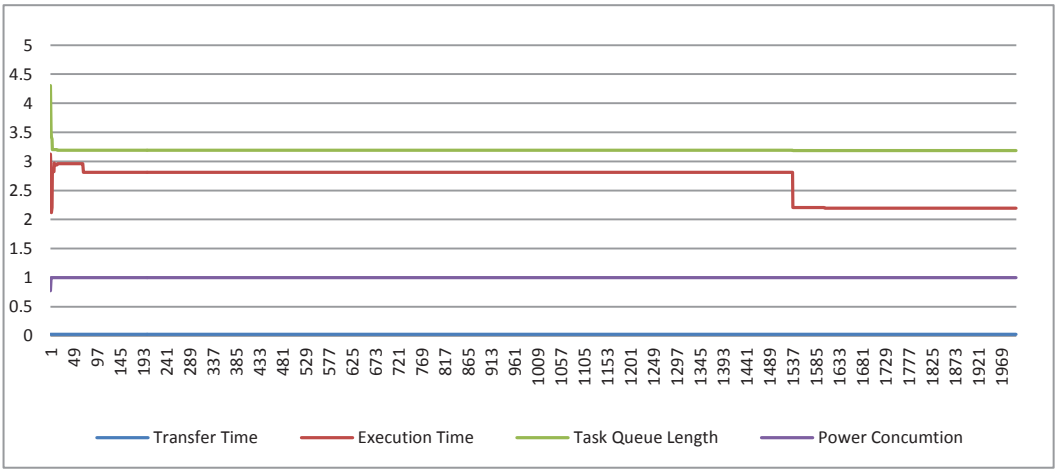


Figure 7.4: The optimal values for objective functions by MOTS–PSO algorithm in 2000 iterations

After executing all scheduled tasks by their corresponding VMs, and t_5 by VM_{17}, the evaluation metrics are calculated as shown in Table 7.6.

Table 7.6: The AS–ORM implementation results in Scenario 1

VM ID	Scheduled Tasks	Execution Time	Makespan
Scheduling T_{set}^{17} to VM_{set}^{17}			
3	$t_{21}, t_{45}, t_{66}, t_{119}, t_{183}, t_{198}$	00:50:30	00:14:39
5	t_7	00:49:22	00:49:22
9	t_{181}, t_{158}	00:05:43	00:05:43
15	t_{126}	00:04:59	00:04:59
Sum		**1:50:34**	**00:49:22**
Executing task by VM_{17}			
17	t_5	**00:49:24**	**00:49:24**
Total		**2:39:58**	**00:49:24**

The value of job makespan is the maximum execution time among VMs (VM_3, VM_5, VM_9, VM_{15} and VM_{17}), and is calculated as follows:

$$Jmakespan = max_{k=1}^{m+1}[Makespane_k] \quad (7.1)$$

where $Makespane_k$ is the maximum time taken by the claimed CPUs of VM_k to execute a sub-set of scheduled tasks, and is calculated as:

$$Makespane_k = max_{i=1}^{VM_{ac}^k}(Texe_i^k) \quad (7.2)$$

where $Texe_i^k$ is the task execution time on *ith* available CPU on VM_k.The value of $Makespane_k$ for each VM is determined by monitoring the VM performance and checking its task log files.

7.5.2 Implementing VMware Auto Load Balancer in the First Scenario

After implementing the FP_TBSLB sub-system, the cluster is set to apply the VMware Auto Load Balancer (ALB). To create the same situation for both models in the first scenario, tasks in the same order as in the previous round are scheduled again to the same VMs in the cluster. After five minutes, VM_{17} has used all its CPU capacity while it has $T_{set}^{17} = \{t_7, t_{21}, t_{45}, t_{66}, t_{181}, t_{119}, t_{126}, t_{158}, t_{183}, t_{198}\}$ in its task queue and is executing t_5. The values of evaluation measurements are calculated after VM_{17} has completed the execution of its scheduled tasks, and are summarized in Table 7.7.

Table 7.7: The implementation results in Scenario 1 using VMware ALB

VM ID	Scheduled Tasks	Execution Time	Makespan
17	$t_5, t_7, t_{21}, t_{45}, t_{66}, t_{181}, t_{119}, t_{126}, t_{158}, t_{183}, t_{198}$	**2:17:03**	**2:17:03**

7.5.3 Implementing AS-ORM in the Second Scenario

To evaluate the AS-ORM in the second scenario, a PM with lower capacity is applied as PM_1 and the resource allocation is changed as follows:

Table 7.8: Resource allocation in the cluster

PM Id	VMs				
1	VM_1	VM_5	–	–	–
2	VM_2	VM_6	VM_{10}	VM_{14}	VM_{18}
3	VM_3	VM_7	VM_{11}	VM_{15}	VM_{19}
4	VM_4	VM_8	VM_{12}	VM_{16}	VM_{20}

To emulate a situation in which a PM will be overloaded, random tasks were chosen from the computationally intensive tasks listed in Table 7.2 and scheduled to the VMs located on PM_1, as illustrated in Table 7.9.

Table 7.9: The list of tasks scheduled to VMs located on PM_1

VM ID	Tasks										
1	t_5	t_{11}	t_{41}	t_{45}	t_{47}	t_{49}	t_{30}	t_{21}	t_{25}	t_{27}	t_{50}
5	t_{44}	t_{46}	t_{51}	t_{27}	t_{39}	–	–	–	–	–	–

In addition, a random set of tasks is scheduled to other VMs in the cluster. The ES&NN-WP algorithm also checks the performance of every VM every two minutes. After two minutes, the algorithm specifies a list of possible overloaded and under-loaded VMs in the cluster as follows:

Table 7.10: The ES&NN-WP results

VM ID	$NN_R^k(Ts)$		$VM_{Huti}^k(Ts)$		$VM_{RisEt}^k(Ts)$		$PI_{Dec}^k(Ts)$		VM_{load}^k
1	O	and	H	and	H	and	H	then	Overloaded
5	O	and	H	and	L	and	H	then	Overloaded
3	U	and	L	and	L	and	L	then	Under-loaded
6	U	and	L	and	L	and	H	then	Under-loaded
7	U	and	L	and	L	and	H	then	Under-loaded
9	U	and	L	and	L	and	H	then	Under-loaded
10	H	and	L	and	L	and	L	then	Under-loaded
11	U	and	L	and	L	and	L	then	Under-loaded

As can be seen from Table 7.10, VM_1 and VM_5 , which are located on PM_1, are suggested by ES&NN-WP as being poorly performing VMs. VM_3, VM_9 and VM_{11} from the list of high performance VMs are compatible withVM_1 and have 2, 1 and 1 available CPUs respectively. VM_6, VM_7 and VM_{10} are compatible with VM_5 with 2,2 and 1 available CPUs.

In this situation, the VM with the lowest number of scheduled tasks is selected to stop executing, and all the tasks scheduled to this VM (pending and executing tasks) are transferred to reduce the tension between VMs on the overloaded PM. The executing tasks on this VM will resume and continue their execution from where they left off on the destination VM, using the Condor check point mechanism. The VMs with lower numbers of scheduled tasks are chosen to stop executing so that fewer tasks need to be transferred. Therefore, the tasks that have accumulated in the VM_1 task queue and all the tasks scheduled to VM_5 are determined as $T_{set}^1 = \{t_{47}, t_{49}, t_{30}, t_{21}, t_{25}, t_{27}, t_{50}\}$ and $T_{set}^5 = \{t_{44}, t_{46}, t_{51}, t_{27}, t_{39}\}$. The MOTS sub-system using MOTS-PSO algorithm is then used to schedule T_{set}^1 and T_{set}^5 to their compatible VMs. The scheduling results are illustrated in Tables 7.11 and 7.12.

Table 7.11: Optimal suggested pattern for scheduling T_{set}^1 to VM_{set}^1

Tasks	t_{47}	t_{49}	t_{30}	t_{21}	t_{25}	t_{27}	t_{50}
Task CPU usage (GHz)	67.166	67.166	21.186	21.186	21.186	21.186	67.166
VMs	VM_{11}	VM_3	VM_9	VM_3	VM_3	VM_9	VM_3

Table 7.12: Optimal suggested pattern for scheduling T_{set}^5 to VM_{set}^5

Tasks	t_{44}	t_{46}	t_{51}	t_{27}	t_{39}
Task CPU usage (GHz)	67.166	67.166	67.166	21.186	21.186
VMs	VM_6	VM_7	VM_7	VM_6	VM_{10}

The optimal values for the objective functions $f_{TransferTime}, f_{Execution}, f_{TaskQueue}$ and $f_{PowerConsumption}$ of MOTS sub-system for the optimal pattern to schedule T_{set}^1 to VM_{set}^1 are 0 second, 2.42 hours (02:25:12), 2.58 and 1 respectively. The corresponding values to these objective functions that are determined for scheduling T_{set}^5 to VM_{set}^5 are 0 seconds, 1.12 hours (01:07:12), 7.20 and 1 respectively.

The execution of tasks t_{44} and t_{46} on VM_5 are stopped, and T_{set}^1 and T_{set}^5 are scheduled according to the suggested optimal patterns to their specified destinations. After

the execution of these tasks has been completed, the related data about the evaluation measurements are determined and summarized in Table 7.13. The value of job makespan is calculated by applying Equation 7.1.

It can be seen from the results that the estimated execution time calculated as objective function $f_{Execution}$ for the tasks scheduled to $VM_1$and VM_5 are close to their corresponding real time consumption of 02:19:06 and 01:00:23.

Table 7.13: The AS–ORM implementation results in Scenario 2

VM ID	Scheduled Tasks	Execution Time	Makespan
Executing tasks by VM_1			
1	$t_5, t_{11}, t_{41}, t_{45}$	**04:16:00**	**01:34:00**
Scheduling $T^1_{set}$to VM^1_{set}			
3	$t_{49}, t_{50}, t_{21}, t_{25}$	01:26:44	00:43:20
9	t_{27}, t_{30}	00:21:30	00:21:30
11	t_{47}	00:30:52	00:30:52
Sum		**02:19:06**	**00:43:20**
Executing tasks by VM_5 before transferring its tasks			
5	t_{44}, t_{46}	**00:04:00**	**00:02:00**
Scheduling T^5_{set} to VM^5_{set}			
6	t_{44}, t_{27}	00:19:42	00:14:55
7	t_{46}, t_{51}	00:31:30	00:16:30
10	t_{39}	00:10:11	00:10:11
Sum		**01:00:23**	**00:16:30**
Total		**07:39:29**	**01:34:00**

7.5.4 Implementing VMware Auto Load Balancer in the Second Scenario

After implementing the AS-ORM, the cluster is set to apply the VMware ALB. The tasks are scheduled to VMs in the same order as scheduled in Section 7.5.3, and$VM_1$and VM_5allocated on PM_1start to use all the resource capacity of their host. After five minutes, when VM_1 has $T^1_{set} = \{t_{47}, t_{49}, t_{30}, t_{21}, t_{25}, t_{27}, t_{50}\}$ in its task queue and is executing t_5, t_{11}, t_{41} and t_{45}, and VM_5 has $T^5_{set} = \{t_{51}, t_{27}, t_{39}\}$ in its task queue and is executing t_{44} and t_{46}, VM_5 is migrated to PM_2 by the hypervisor in approximately 3

seconds. In this state, VM_5 has approximately 100 Mb active memory which is migrated to the new host. In the same situation, the live storage migration of VM_5 with 290 GB (296960 Mb) disk file size and 100 Mb active memory took approximately 5.5 minutes. Before VM_5 migration, the CPU speed of VM_1 and VM_5 were less than the guaranteed speed. Therefore, it takes longer than expected for the executing tasks to be completed. In addition, VM_5 was slowed down during storage migration time, and this also affected the execution time of its allocated task. The task execution time and job makespan in each VM are determined after the execution of these tasks has been completed by VM_1 and VM_5 before and after live and storage migrations, as summarized in Table 7.14.

Table 7.14: The implementation results in Scenario 2 using VMware ALB

VM ID	Location	Scheduled Tasks	Execution Time	Makespan
VM storage migration				
1	PM_1 Before migration PM_1 After migration	$t_5, t_{11}, t_{41}, t_{45}$ $t_5, t_{11}, t_{41}, t_{45}, t_{47}, t_{49}, t_{30}, t_{21}, t_{25}, t_{27}, t_{50}$	6:51:06	2:05:49
5	PM_1 Before migration PM_2 After migration	t_{44}, t_{46} $t_{46}, t_{46}, t_{51}, t_{27}, t_{39}$	1:04:40	0:35:17
Total			**7:55:46**	**2:41:06**
VM live migration				
1	PM_1 Before migration PM_1 After migration	$t_5, t_{11}, t_{41}, t_{45}$ $t_5, t_{11}, t_{41}, t_{45}, t_{47}, t_{49}, t_{30}, t_{21}, t_{25}, t_{27}, t_{50}$	6:46:45	2:04:30
5	PM_1 Before migration PM_2 After migration	t_{44}, t_{46} $t_{46}, t_{46}, t_{51}, t_{27}, t_{39}$	1:01:40	0:33:46
Total			**7:48:25**	**2:38:16**

7.6 Evaluation and Results Analysis

The final results of implementing the two approaches in both scenarios are summarized in Table 7.15. The total task execution time is the summation of time taken by each task to reach completion. The job makespan is considered as the time between the start and finish time for executing all scheduled tasks (T_{set}). In the first scenario, $T_{set} = T_{set}^{17}$ and in the second scenario, $T_{set} = T_{set}^{1} \cup T_{set}^{5}$.

As can be seen from the results, the total task execution time in Scenario 1 with the application of VMware ALB is shorter than the task execution time of the FP-TBSLB sub-system. This occurs because all tasks in the first method were executed by VM_{17} with CPU speed equal to 1.3 GHz, while using the second method, some of the scheduled tasks were sent in parallel to VMs with two or three available CPUs with half CPU speed. However, the job makespan was reduced dramatically by 64% using the proposed model. The optimal pattern for scheduling accumulated tasks in the VM_{17} task queue to a new set of destination VMs was determined in less than 0.5 second.

Table 7.15: Comparison of results

		Time (Seconds and Hours)					Power (kw)	Memory (MB)		
		MOTS-PSO time in	Total task execution time	Job makespan	Task transfer time	VM migration time		Transferred memory	Task size	Transferred data
S1	ALB	NA	8223 (2:17:03)	8223 (2:17:03)	NA	NA	9.12	NA	0.0953	4.2
	FP-TBSLB	0.5	9598 (2:39:58)	2964 (00:49:24)	0	NA	10.64	NA	0.0953	4.2
S2	VM-SM	NA	28546 (7:55:46)	9666 (2:41:16)	NA	330 (0:5:30)	31.71	297059	0.1536	0
	VM-LM	NA	28102 (7:48:25)	9486 (2:38:06)	NA	3 (0:0:3)	31.22	100	0.1536	0
	FP-TBSLB	0.5	27569 (7:39:29)	5640 (1:34:00)	0	NA	30.63	NA	0.1536	0

Note: ALB= VMware Auto Load Balancer, VM-SM= VM Storage Migration, VM-LM=VM Live Migration

On the other hand, the FP-TBSLB sub-system applied $VM_{17}, VM_3, VM_5, VM_9$ and VM_{15} with 1, 3, 1, 1 and 1 available CPUs respectively to execute the tasks, while ALB only applied VM_{17} with one CPU for execution. Therefore, six more CPUs were activated in the proposed method. The additional amount of power consumed per hour by an active CPU can be estimated using Equation 5.18 where $pw_0 = 40^{kw/h}$ and $\alpha = 0.1 * pw_0$:

$$Pw_{active}(CPU) = (pw_0 * 0) + [(0.1 * pw_0) * 1] = 4 * 1 = 4^{kw/h} \quad (7.3)$$

Using this calculation, the additional amount of power consumed by applying ALB for executing T_{set}—which takes 8223 seconds— is:

$$Additional\_Pw_{active}^{ALB}(T_{set}) = 4^{kw/h} * 2.28^{h} = 9.12^{kw} \quad (7.4)$$

In the FP-TBSLB sub-system, the number of activated PMs was not changed because the model applies VMs on active PMs. Therefore, the additional power consumption by the FP-TBSLB sub-system for executing T_{set} —which takes 9598 seconds— can be estimated as follows:

$$Additional\_Pw_{active}^{FP-TBSLB}(T_{set}) = 4^{kw/h} * 2.66^{h} = 10.64^{kw} \quad (7.5)$$

In conclusion, the FP-TBSLB sub-system in this example for the first scenario achieved a 64% reduction in job makespan (response time) while having 1.5^{kw} (14.2%) more power consumption in total. However, the FP-TBSLB sub-system gives users the opportunity to consider their preference by determining different weights for the objective functions of the MOTS problem, taking into account that greater power consumption leads to lower response time and vice versa.

In Scenario 2, when ALB was applied, VM_5 was migrated after PM_1 became overloaded and its allocated VMs experienced tension and low performance for a period. This affected the execution time of tasks on both VMs and it took longer for these VMs to complete their jobs. In contrast, the amount of execution time with the application of FP-TBSLB was reduced by 3.4% and 1.9% compared to VM storage and live migration respectively, as a result of executing tasks by VMs on another PM with lower resource utilization. In addition, makespan was reduced dramatically (41.6% and 40.5% compared to VM storage and live migration respectively) by FP-TBSLB as the tasks were distributed over eight VMs, and this method benefits from parallel task execution. Furthermore, ALB had 100 Mb more memory transferred during live VM migration compared to the FP-TBSLB sub-system. The ALB also had 297059 Mb (≈ 290 GB) more memory consumed —for 5.5 minutes— on both the original and destination PMs and data-stores, compared to the FP-TBSLB sub-system. ALB also used more bandwidth due to VM migration. If the tasks were memory

intensive as well, the amount of active memory transferred could be approximately 4 GB. This would be added to the amount of memory transferred in both live and storage migration, and would increase migration time.

The additional power consumption by ALB using storage migration, ALB using live migration, and FP-TBSLB sub-system for executing T_{set} are calculated as follows:

$$Additional\_Pw_{active}^{ALB-StorageM}(T_{set}) = 4^{kw/h} * 7.93^{h} = 31.71^{kw} \quad (7.6)$$

$$Additional\_Pw_{active}^{ALB-LiveM}(T_{set}) = 4^{kw/h} * 7.80^{h} = 31.22^{kw} \quad (7.7)$$

$$Additional\_Pw_{active}^{FP-TBSLB}(T_{set}) = 4^{kw/h} * 7.65^{h} = 30.63^{kw} \quad (7.8)$$

This shows that the proposed method has the lowest power consumption.

In addition, it takes 330 seconds (5.5 minutes) for ALB to perform VM storage migration, and 3 seconds for VM live migration. In contrast, FP-TBSLB sub-system achieves load balancing in at most 0.5 second, which is the time taken by the MOTS-PSO algorithm —of MOTS subsystem— to find the optimal scheduling pattern. The task transfer time taken in the FP-TBSLB sub-system is near zero. Therefore, the time taken for load balancing is reduced by 99.8% and 83.3% using the FP-TBSLB sub-system compared to VM storage and live migration respectively.

7.7 SUMMARY

Several VM migration techniques have been applied for load balancing and optimizing resource utilization in cloud environments, including suspend/resume and live migration. Live migration has been developed to migrate running VMs, and achieves lower VM downtime compared to the suspend/resume strategy. However, live migration for large size VMs is not an optimal solution because it consumes time, bandwidth, power, and memory space in both origin and destination PMs and data-stores. In addition, the live migration process carries the risk of losing last user activities.

In this study, an AS-ORM along with three sub-systems including WP (ES&NN-WP and DBN-WP), MOTS-PSO/GA and FP-TBSLB have been proposed. The WP sub-

system is applied for predicting resource requirement and VMs performance. The MOTS sub-system is designed for optimal task scheduling over VMs in a cloud cluster minimizing task transfer time, task execution cost/time, length of VM task queue, and power consumption. It also applied as load distributer by FP-TBSLB sub-system for transferring tasks to the selected destination VMs. The MOTS-PSO/GA is developed for optimal task scheduling. The FP-TBSLB sub-system is developed to eliminate VM migration for large size VMs to achieve load balancing as a sub-system of AS-ORM. The FP-TBSLB distributes accumulated tasks in the task queue of the poorly performing VMs over a set of compatible VMs with lower utilization. The FP-TBSLB sub-system also applies the ES&NN-WP algorithm of WP sub-system that not only forecasts poorly performing VMs, but also determines a set of VMs as candidate destinations for arrival tasks to poorly performing VMs. In addition, the poorly performing VMs were not slowed down by the application of the FP-TBSLB sub-system.

The proposed AS-ORM is evaluated by applying a VMware-vSphere based private cloud environment with VMware ESXi hypervisor in two scenarios. The evaluation results show that the AS-ORM achieves reduction in makespan, execution time, memory usage, and total time taken for load balancing in comparison to ALB by vMotion in both live and storage VM migration. As a result, the proposed system dramatically increases VM performance and reduces service response with optimal resource utilization —which have been the main objectives for optimal resource management. The AS-ORM can be utilized by the hypervisor for optimal load balancing and resource management in cloud environments.

Chapter 8.

CONCLUSIONS AND FUTURE WORKS

8.1 CONCLUSIONS

The main objective of cloud providers is to determine optimal resource allocation to various services in order to best maximize their revenue while minimizing their costs. In addition, designing optimal pattern for resource usage considering arrival tasks is the key to several crucial system design and deployment decisions such as, workload management, system sizing, capacity planning and dynamic rule generation in the cloud. Thus, both resource discovery and selection, and resource allocation, which are the most frequently encountered combinatorial optimization problems, play a key role to improve flexible and reliable systems.

It is also intuitive that if the dynamic resource scaling system is a reactive one, it might not be able to scale proportionally with the Slashdot effect or sudden traffic surge resulting from special offers or market campaigns; thus turning out to be catastrophic for application performance, leading to an unacceptable delay in response time and in the worst case, application unavailability (Ghanbari et al. 2012). Therefore, proactive prediction-based resource management is required in order to estimate required resources and cope with the ever fluctuating resource usage pattern of e-commerce applications.

Considering these facts, this study investigates three main topics in resource management including: (1) resource estimation, (2) resource discovery and selection in terms of optimizing task scheduling, and (3) physical resource allocation to VM or load balancing. However, while predicting required resources, task scheduling and resources allocation optimization, subject to minimizing the execution time and cost, and maximizing the QoS have been extensively studied, none of them consider the interaction between them. Furthermore, most of the studies in the load balancing area work on automated VM migration. that is the underlying technique of virtualization that refers to the process of moving a running VM between different PMs without disconnecting the client or application (Huang et al. 2016).

This research has been motivated by this fact that predicting VM workload and resource usage, task scheduling, load balancing have been recognized as important contributing factors in optimal resource management in cloud environments. It has also been realized that creating a management system from the combination of these factors considering their interaction is critical in managing abnormal situations when some PMs are overloaded and some VMs are exhibiting low performance. In addition, predicting required resources and VMs workload are investigated in this study to develop proper prediction methods and accelerate decision making in resource management in cloud environments. Furthermore, this study proposed a new technique to promote current load balancing approaches. In summary, this research makes the following main contributions:

1) **Develop a novel Autonomic System for Optimal Resource Management (AS-ORM)**: This research developed a new AS-ORM for optimizing cloud resource management that optimizes task scheduling and resources allocation dynamically based on predicted required resources to meet optimal cloud utilization. Using this, cloud providers have the opportunity to maximize their benefits and prepare higher quality services for cloud customers. The AS-ORM is applied in the hypervisor layer of cloud architecture for general or particular resource management purpose. This system is

flexible about the number of optimization objectives. It has the ability to optimize resource management based on all the objects of the multi-objective problem (general) or do optimization on the basis of a set of selected objects according to the customer preferences (particular). This system has three subsystems called:

- WP (ES&NN-WP and DBN-WP) sub-system for VM workload prediction (resource estimation)
- MOTS sub-system for optimizing task scheduling (resource discovery and selection)
- FP-TBSLB sub-system for optimal load balancing (resource allocation).

These sub-systems work together to achieve the main goals including: maximizing the quality of service (QoS), minimizing delay in cloud service response time, preventing network overload and minimizing costs. By developing AS-ORM, Objectives 1 and 2 of this study has been achieved.

2) **Develop VM Workload Prediction models by applying Expert System and Neural Network (ES&NN-WP), and Dynamic Bayesian Network (DBN-WP)**: This research proposed two fuzzy Workload Prediction (WP) methods that monitor both historical and current VM CPU utilization and workload to predict probable poorly performing VMs. This model is also applied to predict PM resource utilization and hotspots, and virtual resource discovery. ES&NN-WP is also applied in FP-TBSLB approach that not only predicts the poorly performing VMs but also determines a set of appropriate under-loaded VMs that have the potential to execute the extra workload imposed on the low performance VMs. This system is developed to satisfy Objective 3 of this study.

3) **Develop a Multi-Objective Task scheduling optimization (MOTS) sub-system**: Cloud computing enables the sharing of computing resources that are distributed all over the world. Scheduling optimization techniques can reduce delay in clouds' service response time to enhance clouds customers' satisfaction and quality of

the delivered service. In addition, these techniques will help cloud providers to dynamically allocate their resources to various clients in order to best maximize its revenue while minimizing its costs. This study develops a MOTS sub-system. To do this, a multi-objective optimization algorithm for task scheduling applying MOPSO and MOGA is designed that is called MOTS-PSO/GA algorithm. This algorithm have four objectives: (1) minimizing tasks execution time/cost, (2) minimizing tasks transfer time, (3) minimizing the length of the task queues of VMs, and (4) minimizing power consumption. In the proposed AS-ORM, tasks are allocated to the virtual resources according to predicted required resources and optimized task scheduling pattern. MOTS sub-system using MOTS-PSO is also applied as a part of FP-TBSLB sub-system to suggest optimal pattern to distribute the extra workload of poorly performing VMs to the determined compatible VMs. This system is developed to cover Objective 4 of this study.

4) **Develop a Fuzzy Predictable Task Based System Load Balancing (FP-TBSLB) technique**: This research presents a novel load balancing approach to overcome the drawbacks of current load balancing solutions that have relied on VM migration strategy.

This model avoids VM migration for large size VM and is designed based on the idea of reducing workload of poorly performing VMs by transferring the accumulated tasks in their task queue. FP-TBSLB sub-system employs two other sub-systems including: (1) the WP sub-system(using ES&NN-WP algorithm) that monitors CPU usage by VMs and forecasts poorly performing VMs, a set of VMs with available resources, and PMs hotspots (2) the MOTS sub-system (using MOTS-PSO algorithm) to transfer tasks from selected VMs to a set of determined under-loaded VMs. FP-TBSLB sub-system has also a new central scheduler and a blackboard where the VMM schedulers collect their information about VMs and PMs. The central scheduler communicates

with other schedulers via this blackboard and applies their information about VMs performance, the set of tasks that are scheduled to VMs, and CPU usage by VMs, as input variables for MOTS-PSO and ES&NN-WP algorithms.

In this proposed approach there is no need to pause VM during load balancing process and VMs are not slowed down. In addition, as VM live migration in contrast to tasks migration will take a long time to complete and needs more idle capacity in both original and destination PMs, the proposed approach will significantly reduce time, memory consumption and cost. The FP-TBSLB satisfies Objective 5 of this study.

5) **Develop an intelligent multi-objective system prototype for optimizing resource management in cloud environments**: This research develops an AS-ORM system prototype in the form of a software which provides all necessary information and suggests the proper actions and gives the opportunity to optimize cloud utilization. This software can be applied in the virtualization layer of cloud clusters by the hypervisor. Objectives 6 and 7 of this study are achieved by developing and evaluating this prototype.

6) **Evaluation Results**: The evaluation results show that the AS_ORM achieves lower execution time, job makespan, memory transferred, bandwidths, power consumption in comparison with live and storage migration strategies applied by VMware-ESXi. In addition, the AS_ORM consumes lower amount of time to conduct load balancing over PMs compared to ALB of VMware-ESXi. The amount of execution time is reduced by 3.4% and 1.9% compared to VM storage and live migration respectively. The makespan was reduced dramatically by 64% in the Scenario 1, and it is reduced by 41.6% and 40.5% compared to VM storage and live migration respectively in the Scenario 2. The time taken for load balancing is reduced by 99.8% and 83.3% using the FP-TBSLB sub-system of AS_ORM compared to VM storage and live migration respectively.

8.2 FUTURE WORKS

Future directions of this research in the area of prediction, task scheduling and load balancing can be summarized in the following perspectives:

1) Considering the fact that the reliable prediction results support AS-ORM to make decisions properly for resource management, it is necessary to improve the performance of the WP sub-system. To develop DBN-WP model of the WP sub-system, fuzzy Bayesian networks are implemented based on experts' experience to handle some of the uncertainty. In this model, the data about the status of cloud resources over time is stored. Therefore, the future work will be to consider the use of stored data in the training of the DBN models to achieve more accurate prediction results.

2) To increase the QoS, the customer preferences should be taken into account in resource management and allocation. The more preferences considered the higher QoS will be achieved. To reach this goal, this study will be improved by considering more criteria of SLA as new objectives to develop multi-objective task scheduling optimization model. This needs developing new formulas to estimate optimal value of objectives, developing new multi-objective algorithms and preparing related software to execute.

3) In FP-TBSLB accumulated tasks in the task queue of a poorly performing VM are scheduled to under-loaded VMs. In some cases task are scheduled to under-loaded VMs with lower CPU speed. In this situation, although the system benefits from executing tasks in parallel by a larger number of under-utilized CPUs, the execution of a task by a lower speed CPU will take more time and consequently more power consumption. However, in total, job makespan will be significantly reduced and the increase in power consumption will be small, this study aims to solve this matter. To do this, in the future work additional constraints will be added to the MOTS problem —applied in FP-TBSLB— to consider scheduling tasks to VMs that have a CPU speed

equal to or more than the CPU speed of the primary VM. Using this strategy, there would not be even a slight increase in total execution time and power consumption.

4) There are still several aspects in resource management that can be considered to enhance the functionality of the developed AS-ORM. Another important topic in resource management is resource mapping in terms of mapping of virtual resources over physical resources to develop the fundamental structure of the cloud infrastructure based on customer demand. Resource mapping is a system-building process that enables identifying available resources and matching those resources to a specific purpose. The issue here is to maximize cloud utilization in IaaS by calculating the required capacity so that optimal resources will be procured and maintained (Manvi & Krishna Shyam 2014). Therefore, it is considered as a future work of this study to develop a multi-objective model for re-mapping VMs over physical resources in cloud environments according to the predicted required resources, and determine optimized resource allocation. The goal is not only providing the opportunity to employ spare resources and achieve optimal resource utilization, but also to reduce the cloud providers' cost and increase the QoS.

REFERENCES

Angel Diaz & Chris Ferris 2013, *IBM's open cloud architecture*, IBM, <http://www.ibm.com/>.

Antonescu, A.F. & Braun, T. 2016, 'Simulation of SLA-based VM-scaling algorithms for cloud-distributed applications', *Future Generation Computer Systems*, vol. 54, pp. 260-73.

Anuradha, V. & Sumathi, D. 2014, 'A survey on resource allocation strategies in cloud computing', *International Conference on Information Communication and Embedded Systems (ICICES), 2014*, IEEE, pp. 1-7.

Ardagna, D., Casolari, S., Colajanni, M. & Panicucci, B. 2012, 'Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems', *Journal of Parallel and Distributed Computing*, vol. 72, no. 6, pp. 796-808.

Ardagna, D., Panicucci, B., Trubian, M. & Li, Z. 2012, 'Energy-aware autonomic resource allocation in multitier virtualized environments', *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 2-19.

Arlot, S. & Celisse, A. 2010, 'A survey of cross-validation procedures for model selection', *Statistics Surveys*, vol. 4, pp. 40–79.

Atif, M. 2010, 'Adaptive Resource Relocation in Virtualized Heterogeneous Clusters', Doctor of Philosophy thesis, The Australian National University.

Atif, M. & Strazdins, P. 2014, 'Adaptive parallel application resource remapping through the live migration of virtual machines', *Future Generation Computer Systems*, vol. 37, pp. 148-61.

Barati, M. & Sharifian, S. 2015, 'A hybrid heuristic-based tuned support vector regression model for cloud load prediction', *Journal of Supercomputing*, vol. 71, no. 11, pp. 4235-59.

Bashar, A. 2013, 'Autonomic scaling of Cloud Computing resources using BN-based prediction models', *2nd IEEE International Conference on Cloud Networking (CloudNet)*, San Francisco, CA, pp. 200-4.

Buyya, R., Beloglazov, A. & Abawajy, J. 2010, 'Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges', *arXiv preprint arXiv:1006.0308*.

Buyya, R., Broberg, J. & Goscinski, A. (eds) 2011, *Cloud Computing, Principles and Paradigms*, vol. 81 John Wiley & Sons, Hoboken, NJ.

Buyya, R., Calheiros, R.N. & Li, X. 2012, 'Autonomic cloud computing: Open challenges and architectural elements', *Emerging Applications of Information Technology (EAIT), 2012 Third International Conference on*, IEEE, pp. 3-10.

Buyya, R., Pandey, S. & Vecchiola, C. 2009, 'Cloudbus toolkit for market-oriented cloud computing', *Cloud Computing*, Springer, pp. 24-44.

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F. & Buyya, R. 2011, 'CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms', *Software: Practice and Experience*, vol. 41, no. 1, pp. 23-50.

Celesti, A., Fazio, M., Villari, M. & Puliafito, A. 2012, 'Virtual machine provisioning through satellite communications in federated Cloud environments', *Future Generation Computer Systems*, vol. 28, no. 1, pp. 85-93.

Cingolani, P. 2009, *Jswarm package.* http://jswarm-pso.sourceforge.net/, <http://jswarm-pso.sourceforge.net/>.

Cirne, W., Brasileiro, F., Andrade, N., Costa, L.B., Andrade, A., Novaes, R. & Mowbray, M. 2006, 'Labs of the world', *Journal of Grid Computing*, vol. 4, no. 3, pp. 225-46.

Clark, C., Fraser, K., Hand, S. & Jacob, G.H. 2005, 'Live migration of virtual machines', *Proceedings of the 2nd ACM/USENIX Symposium on Network Systems, Design and Implementation (NSDI)*, pp. 273-86.

CloudPatterns 2015, <http://cloudpatterns.org/mechanisms/resource_management_system>.

Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R. & Von Eicken, T. 1993, 'LogP: Towards a realistic model of parallel computation', vol. 28, ACM Sigplan Notices, pp. 1-12.

Devi, P., Gupta, S. & Choudhary, S. 2014, 'Parallel Efficiency for Graph Search', *International Journal of Enhanced Research in Science Technology & Engineering*, vol. 3, no. 4, pp. 278-82.

Dietterich, T. 2002, 'Machine learning for sequential data: A review, in: T. Caelli, A. Amin, R. Duin, D. de Ridder, M. Kamel (Eds.), Structural, Syntactic, and Statistical Pattern Recognition', *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, pp. 227–46.

Dougherty, B., White, J. & Schmidt, D.C. 2012, 'Model-driven auto-scaling of green cloud computing infrastructure', *Future Generation Computer Systems*, vol. 28, no. 2, pp. 371-8.

Efron, B. & Gong, G. 1983, 'A leisurely look at the bootstrap, the jackknife, and crossvalidation', *The American Statistician*, vol. 37, pp. 36–48.

Engelbrecht, A.P. 2005, *Fundamentals of Computational Swarm Intelligence*, vol. 67, John Wiley & Sons, Hoboken, NJ.

Engelbrecht, A.P. 2007, *Computational intelligence: an introduction*, John Wiley & Sons, Ltd, Hoboken.

Gallupe, R.B. 2007, 'The tyranny of methodologies in information systems research', *SIGMIS Database*, vol. 38, no. 3, pp. 20-8.

Gao, Y., Zhang, G., Lu, J. & Wee, H.-M. 2011, 'Particle swarm optimization for bi-level pricing problems in supply chains', *Journal of Global Optimization*, vol. 51, no. 2, pp. 245-54.

Ghanbari, H., Simmons, B., Litoiu, M. & Iszlai, G. 2012, 'Feedback-based optimization of a private cloud', *Future Generation Computer Systems*, vol. 28, no. 1, pp. 104-11.

Grzonka, D., Kołodziej, J., Tao, J. & Khan, S.U. 2015, 'Artificial Neural Network support to monitoring of the evolutionary driven security aware scheduling in computational distributed environments', *Future Generation Computer Systems*, vol. 51, pp. 72-86.

Guo, L., Zhao, S., Shen, S. & Jiang, C. 2012, 'Task Scheduling Optimization in Cloud Computing Based on Heuristic Algorithm', *Journal of Networks*, vol. 7, no. 3, pp. 547-53.

Hadka, D. 2014, *MOEA Framework A Free and Open Source Java Framework for Multiobjective Optimization*, 2014, <http://www.moeaframework.org/>.

Halavais, A.C. 2001, 'The slashdot effect: Analysis of a large-scale public conversation on the World Wide Web', 3022843 thesis, University of Washington, United States - Washington.

Hu, Y., Wong, J., Iszlai, G. & Litoiu, M. 2009, 'Resource provisioning for cloud computing', *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, ACM, pp. 101-11.

Huang, T., Zhu, Y., Wu, Y., Bressan, S. & Dobbie, G. 2016, 'Anomaly detection and identification scheme for VM live migration in cloud infrastructure', *Future Generation Computer Systems*, vol. 56, pp. 736-45.

Islam, S., Keung, J., Lee, K. & Liu, A. 2012a, 'Empirical prediction models for adaptive resource provisioning in the cloud', *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155-62.

Islam, S., Keung, J., Lee, K. & Liu, A. 2012b, 'Empirical prediction models for adaptive resource provisioning in the cloud', *Future Generation Computer Systems*, vol. 28, pp. 155–62.

Jain, N., Menache, I., Naor, J. & Shepherd, F. 2012, 'Topology-aware VM migration in bandwidth oversubscribed datacenter networks', paper presented to the *39th International Colloquium on Autonomic, Language, and Programming*.

Jin, H., Gao, W., Wu, S., Shi, X., Wu, X. & Zhou, F. 2011, 'Optimizing the live migration of virtual machine by CPU scheduling', *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1088-96.

Juhnke, E., D¨ornemann, T., B¨ock, D. & Freisleben, B. 2011, 'Multi-objective scheduling of BPEL workflows in geographically distributed clouds', *4th IEEE International Conference on Cloud Computing*, pp. 412-9.

Jun, C. & Xiaowei, C. 2011, 'IPv6 virtual machine live migration framework for cloud computing', *Energy Procedia*, vol. 13, pp. 5753-7.

Kennedy, J. & Eberhart, R. 1995, 'Particle swarm optimization', *Proceedings of the IEEE International Conference on Neural Networks* vol. 4, pp. 1942-8

Konak, A., Coit, D.W. & Smith, A.E. 2006, 'Multi-objective optimization using genetic algorithms: A tutorial', *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992-1007.

Kong, X., Lin, C., Jiang, Y., Yan, W. & Chu, X. 2011, 'Efficient dynamic task scheduling in virtualized data centers with fuzzy prediction', *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1068-77.

Kozuch, M. & Satyanarayanan, M. 2002, 'Internet suspend/resume', *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications* pp. 40-6.

Kwakernaak, H. 1978, 'Fuzzy random variables I: definitions and theorems', *Information Sciences*, vol. 15, no. 1, pp. 1-29.

Laboratory, D.S. 1998, 'SMILE (Structural Modeling, Inference, and Learning Engine)', library of C++ University of Pittsburgh, <http://genie.sis.pitt.edu/>.

Lei, Z., Yuehui, C., Runyuan, S., Shan, J. & Bo, Y. 2008, 'A task scheduling algorithm based on PSO for grid computing', *International Journal of Computational Intelligence Research*, vol. 4, no. 1, pp. 37-43.

Li, J., Peng, J., Cao, X. & Li, H.-y. 2011, 'A task scheduling algorithm based on improved ant colony optimization in cloud computing environment', *Energy Procedia*, vol. 13, pp. 6833-40.

Li, J., Qiu, M., Ming, Z., Quan, G., Qin, X. & Gu, Z. 2012, 'Online optimization for scheduling preemptable tasks on IaaS cloud systems', *Journal of Parallel and Distributed Computing*, vol. 72, no. 5, pp. 666-77.

Liao, X., Jin, H. & Liu, H. 2012, 'Towards a green cluster through dynamic remapping of virtual machines', *Future Generation Computer Systems*, vol. 28, no. 2, pp. 469-77.

Lin, W., Wang, J.Z., Liang, C. & Qi, D. 2011, 'A threshold-based dynamic resource allocation scheme for cloud computing', *Procedia Engineering*, vol. 23, pp. 695 – 703.

Liu, H., Abraham, A., SnÃ¡Å¡el, V.c. & McLoone, S.n. 2012, 'Swarm scheduling approaches for work-flow applications with security constraints in distributed data-intensive computing environments', *Information Sciences*, vol. 192, no. 0, pp. 228-43.

Liu, Y., Gong, B., Xing, C. & Jian, Y. 2014, 'A virtual machine migration strategy based on time series workload prediction using cloud model', *Mathematical Problems in Engineering*, vol. 2014, pp. 1-11.

Lopez, V. & Minana, G. 2012, 'Modeling the stability of a computer system', *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 20, no. supp01, pp. 81-90.

Lu, J., Zhang, G. & Ruan, D. 2007, *Multi-objective Group Decision Making: Methods, Software and Applications with Fuzzy Set Techniques*, Imperial College Press, London.

Mahabadi, A., Zahedi, S.M. & Khonsari, A. 2013, 'Reliable energy-aware application mapping and voltage–frequency island partitioning for GALS-based NoC', *Journal of Computer and System Sciences*, vol. 79, no. 4, pp. 457-74.

Mahmoodabadi, M.J., Bagheri, A., Nariman-zadeh, N. & Jamali, A. 2012, 'A new optimization algorithm based on a combination of particle swarm optimization, convergence and divergence operators for single-objective and multi-objective problems', *Engineering Optimization*, vol. 44, no. 10, pp. 1-20.

Mamdani, E.H. 1977, 'Application of fuzzy logic to approximate reasoning using linguistic synthesis', *IEEE Transactions on Computers*, vol. C-26, no. 12, pp. 1182-91.

Mangaiyarkarasi, K., Sureshkumar, K. & Elango, N. 2013, 'Comparative study on performance test methodologies-traditional and cloud', *International Journal of Advanced Research in Computer Science and Applications*, vol. 1, no. 2, pp. 1-11.

Manvi, S.S. & Krishna Shyam, G. 2014, 'Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey', *Journal of Network and Computer Applications*, vol. 41, no. 0, pp. 424-40.

Markowski, A.S., Mannan, M.S., Kotynia, A. & Pawlak, H. 2011, 'Application of fuzzy logic to explosion risk assessment', *Journal of Loss Prevention in the Process Industries*, vol. 24, no. 6, pp. 780-90.

Mendes, R., Weingartner, R., Geronimo, G., Bräscher, G., Flores, A., Westphall, C. & Westphall, C. 2014, 'Decision-theoretic planning for cloud computing', *The Thirteenth International Conference on Networks (ICN)*, pp. 191-197.

Meng, X., Isci, C., Kephart, J., Zhang, L., Bouillet, E. & Pendarakis, D. 2010, 'Efficient resource provisioning in compute clouds via VM multiplexing', *Proceeding of the 7th International Conference on Autonomic Computing*, ACM, pp. 11-20.

Mian, R., Martin, P. & Vazquez-Poletti, J.L. 2013, 'Provisioning data analytic workloads in a cloud', *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1452-8.

Moltó, G., Caballer, M. & de Alfonso, C. 2016, 'Automatic memory-based vertical elasticity and oversubscription on cloud platforms', *Future Generation Computer Systems*, vol. 56, pp. 1-10.

Naderpour, M. 2015, 'Intelligent situation awareness support system for safety-critical environments', *PhD thesis,* University of Technology Sydney, Australia.

Naderpour, M. & Lu, J. 2012a, 'A fuzzy dual expert system for managing situation awareness in a safety supervisory system', *2012 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Australia, Brisbane, pp. 1-7.

Naderpour, M. & Lu, J. 2012b, 'Supporting situation awareness using neural network and expert system', *International Conference on uncertainty Modeling in Knowledge*

Engineering and Decision Making (FLINS 2012) World Scientific, Turkey, Istanbul, pp. 993-8.

Naderpour, M. & Lu, J. 2014, 'A situation analysis decision support system based on dynamic object oriented Bayesian networks', *Journal of Software*, vol. 9, no. 8, pp. 2194-9.

Naderpour, M., Lu, J. & Zhang, G. 2014a, 'The explosion at Institute: Modeling and analyzing the situation awareness factor', *Accident Analysis & Prevention*, vol. 73, no. 0, pp. 209-24.

Naderpour, M., Lu, J. & Zhang, G. 2014b, 'An intelligent situation awareness support system for safety-critical environments', *Decision Support Systems*, vol. 59, pp. 325-40.

Naderpour, M., Lu, J. & Zhang, G. 2014c, 'A situation risk awareness approach for process systems safety', *Safety Science*, vol. 64, pp. 173-89.

Nagothu, K., Kelley, B., Prevost, J. & Jamshidi, M. 2010a, 'On prediction to dynamically assign heterogeneous microprocessors to the minimum joint power state to achieve ultra low power cloud computing', *Conference Record of the 44th Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, IEEE, pp. 1269-73.

Nagothu, K.M., Kelley, B., Prevost, J. & Jamshidi, M. 2010b, 'Ultra low energy cloud computing using adaptive load prediction', *Proceedings of the World Automation Congress (WAC)*, IEEE, pp. 1-7.

Nelson, M., Lim, B.H. & Hutchins, G. 2005, 'Fast transparent migration for virtual machines', *USENIX Annual Technical Conference*, pp. 391-4.

Nicolae, B. & Cappello, F. 2013, 'BlobCR: Virtual disk based checkpoint-restart for HPC applications on IaaS clouds', *Journal of Parallel and Distributed Computing*, vol. 73, no. 5, pp. 698-711.

Niu, L., Lu, J. & Zhang, G. 2009, *Cognition-driven decision support for business intelligence: Models, techniques, systems and applications*, vol. 238, Springer-Verlag, Berlin Heidelberg.

Osman, S., Subhraveti, D., Su, G. & Nieh, J. 2002, 'The design and implementation of ZAP: A system for migrating computing environments', *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 361-76.

Oztekin, A. 2009, 'A generalized hybrid fuzzy-Bayesian methodology for modeling complex uncertainty', PhD thesis, The State University of New Jersey, New Jersey.

Pillai, K. & Ozansoy, C. 2012, 'Web-based digital habitat ecosystems for sustainable built environments', *Green Technologies and Business Practices: An IT Approach*, p. 185.

Platform as a Service Magazine 2015, <http://www.paasmag.com/about/>.

Poli, R., Kennedy, J. & Blackwell, T. 2007, 'Particle swarm optimization', *Swarm Intelligence*, vol. 1, no. 1, pp. 33-57.

Priya, B., Pilli, E.S. & Joshi, R.C. 2013, 'A survey on energy and power consumption models for Greener Cloud', *IEEE 3rd International Conference on Advanced Computing (IACC)*, IEEE, pp. 76-82.

Ramezani, F., Lu, J. & Hussain, F. 2012, 'Tasks based system load balancing approach in cloud environment', *International Conference on Intelligent Systems and Knowledge Engineering*, Springer pp. 31-42.

Rao, J. 2011, 'Autonomic management of virtualized resources in cloud computing', PhD thesis, Wayne State University, Detroit, Michigan.

Riesen, R. & Maccabe, A. 2011, 'Job Scheduling', in D. Padua (ed.), *Encyclopedia of Parallel Computing*, Springer US, pp. 997-1002.

Rizvandi, N.B., Taheri, J. & Zomaya, A.Y. 2011, 'Some observations on optimal frequency selection in DVFS-based energy consumption minimization', *Journal of Parallel and Distributed Computing*, vol. 71, no. 8, pp. 1154-64.

Sadeka, I., Jacky, K., Kevin, L. & Anna, L. 2012, 'Empirical prediction models for adaptive resource provisioning in the cloud', *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155-62.

Sahu, Y. & Pateriya, R. 2013, 'Cloud Computing overview with load balancing techniques', *International Journal of Computer Applications*, vol. 65, no. 24, pp. 0975-8887.

Sallam, A. & Li, K. 2014, 'A multi-objective virtual machine migration policy in cloud systems', *The Computer Journal*, vol. 57, no. 2, pp. 195-204.

Salman, A., Ahmad, I. & Al-Madani, S. 2002, 'Particle swarm optimization for task assignment problem', *Microprocessors and Microsystems*, vol. 26, no. 8, pp. 363-71.

Sapuntzakis, C.P., Chandra, R., Pfaff, B., Chow, J., Lam, M.S. & Rosenblum, M. 2002, 'Optimizing the migration of virtual computers', *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 377-90.

Saripalli, P., Kiran, G., Shankar, R.R., Narware, H. & Bindal, N. 2011, 'Load prediction and hot spot detection models for autonomic cloud computing', *Proceedings of 4th IEEE International Conference on Utility and Cloud Computing (UCC)*, IEEE, pp. 397-402.

Schmidt, K.W. & Boutalis, Y.S. 2012, 'Fuzzy discrete event systems for multiobjective control: Framework and application to mobile robot navigation', *IEEE Transactions on Fuzzy Systems*, vol. 20, no. 5, pp. 910-22.

Schwoebel, J. 2015, *Mind the Virtualization*, <http://www.mindthevirt.com/>.

SGI 2008, 'Altix® XE320 System User's Guide', <http://techpubs.sgi.com/library/manuals/5000/007-5466-001/pdf/007-5466-001.pdf>.

Shapiro, A.F. 2009, 'Fuzzy random variables', *Insurance: Mathematics and Economics*, vol. 44, no. 2, pp. 307-14.

Shieh, W.-Y. & Pong, C.-C. 2013, 'Energy and transition-aware runtime task scheduling for multicore processors', *Journal of Parallel and Distributed Computing*, vol. 73, no. 9, pp. 1225-38.

Shu-Hsien, L. 2005, 'Expert system methodologies and applications: a decade review from 1995 to 2004', *Expert Systems with Applications*, vol. 28, no. 1, pp. 93-103.

Song, B., Hassan, M.M. & Huh, E. 2010, 'A novel heuristic-based task selection and allocation framework in dynamic collaborative cloud service platform', *2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 360-7.

Srinivas, N. & Deb, K. 1994, 'Muiltiobjective optimization using nondominated sorting in genetic algorithms', *Evolutionary Computation*, vol. 2, no. 3, pp. 221-48.

Su, S., Li, J., Huang, Q., Huang, X., Shuang, K. & Wang, J. 2013, 'Cost-efficient task scheduling for executing large programs in the cloud', *Parallel Computing*, vol. 39, no. 4–5, pp. 177-88.

Sugeno, M. 1985, *Industrial applications of fuzzy control*, Elsevier Science Inc.

Taheri, J., Choon Lee, Y., Zomaya, A.Y. & Siegel, H.J. 2013, 'A Bee Colony based optimization approach for simultaneous job scheduling and data replication in grid environments', *Computers & Operations Research*, vol. 40, no. 6, pp. 1564-78.

Taheri, J., Zomaya, A.Y., Bouvry, P. & Khan, S.U. 2013, 'Hopfield neural network for simultaneous job scheduling and data replication in grids', *Future Generation Computer Systems*, vol. 29, pp. 1885-900.

Taheri, J., Zomaya, A.Y., Siegel, H.J. & Tari, Z. 2014, 'Pareto frontier for job execution and data transfer time in hybrid clouds', *Future Generation Computer Systems*, vol. 37, pp. 321-34.

Tayal, S. 2011, 'Tasks Scheduling optimization for the Cloud Computing Systems', *International Journal of Advanced Engineering Sciences and Technologies*, vol. 5, no. 2, pp. 111-5.

Tchernykh, A., Pecero, J.E., Barrondo, A. & Schaeffer, E. 2014, 'Adaptive energy efficient scheduling in peer-to-peer desktop grids', *Future Generation Computer Systems*, vol. 36, pp. 209-20.

Theodoridis, S. & Koutroumbas, K. 2008, *Pattern Recognition*, Academic Press.

Tong, Z., Xiao, Z., Li, K. & Li, K. 2014, 'Proactive scheduling in distributed computing-A reinforcement learning approach', *Journal of Parallel and Distributed Computing*, vol. 74, no. 7, pp. 2662-72.

Top 500 Supercomputing Sites 2014, <http://www.top500.org/system/176223>.

Vakilinia, S., Ali, M.M. & Qiu, D. 2015, 'Modeling of the resource allocation in cloud computing centers', *Computer Networks*, vol. 91, pp. 453-70.

Van, H.N., Tran, F.D. & Menaud, J.-M. 2009, 'SLA-aware virtual resource management for cloud infrastructures', *Ninth IEEE International Conference on Computer and Information Technology (CIT'09)*, vol. 1, IEEE, pp. 357-62.

Varia, J. 2008, 'Cloud architectures', *White Paper of Amazon*, p. 16.

vmware 2015, *VMware vSphere 5.1 Documentation Center*, <https://www.vmware.com/support/pubs/>.

Vvirtual's Blog 2011, <https://vvirtual.wordpress.com/2011/10/17/how-to-enable-evc-on-a-cluster-when-vcenter-is-a-virtual-machine/>.

Wang, J., Qiu, M. & Guo, B. 2015, 'High reliable real-time bandwidth scheduling for virtual machines with hidden Markov predicting in telehealth platform', *Future Generation Computer Systems*, vol. 49, pp. 68-76.

Wang, L.-X. 1999, *A course in fuzzy systems*, Prentice-Hall press, USA.

Wang, L., Khan, S.U., Chen, D., Kołodziej, J., Ranjan, R., Xu, C.-z. & Zomaya, A. 2013, 'Energy-aware parallel task scheduling in a cluster', *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1661-70.

Wang, L., Ranjan, R., Chen, J. & Benatallah, B. (eds) 2012, *Cloud computing methodology systems and applications*.

Wang, X., Wang, Y. & Cui, Y. 2014, 'A new multi-objective bi-level programming model for energy and locality aware multi-job scheduling in cloud computing', *Future Generation Computer Systems*, vol. 36, pp. 91-101.

Wang, Y., Lu, P. & Kent, K.B. 2015, 'WaFS: A Workflow-Aware File System for Effective Storage Utilization in the Cloud', *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2716-29.

Weber, P. & Jouffe, L. 2006, 'Complex system reliability modelling with Dynamic Object Oriented Bayesian Networks', *Reliability Engineering & System Safety*, vol. 91, no. 2, pp. 149-62.

Wei, G., A.V.Vasilakos, Zheng, Y. & Xiong, N. 2010, 'A game-theoretic method of fair resource allocation for cloud computing services', *J Supercomput*, vol. 54, pp. 252-69.

Weingärtner, R., Bräscher, G.B. & Westphall, C.B. 2015, 'Cloud resource management: A survey on forecasting and profiling models', *Journal of Network and Computer Applications*, vol. 47, pp. 99-106.

Weisberg, S. 2005, 'Applied Linear Regression', *Wiley Series in Probability and Statistics*, John Wiley & Sons, Hoboken, NJ.

Whitaker, A., Cox, R.S., Shaw, M. & Gribble, S.D. 2004, 'Constructing services with interposable virtual hardware', *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 169-82

Wikipedia 2011, *Autonomic computing*, <http://en.wikipedia.org/wiki/Autonomic_computing>.

Wikipedia 2012, *Load balancing (computing)*, 2012, <http://en.wikipedia.org/wiki/Load_balancing_(computing)>.

Xiong, K. & Perros, H. 2006, 'Resource Optimization Subject to a Percentile Response Time SLA for Enterprise Computing', *IEEE GLOBECOM 2006 proceedings*.

Yang, D., Cao, J., Fu, J., Wang, J. & Guo, J. 2013, 'A pattern fusion model for multi-step-ahead CPU load prediction', *Journal of Systems and Software*, vol. 86, no. 5, pp. 1257-66.

Yeo, C.S. 2008, 'Utility-based Resource Management for Cluster Computing', University of Melbourne.

Zadeh, L.A. 1965, 'Fuzzy sets', *Information and Control*, vol. 8, no. 3, pp. 338-53.

Zadeh, L.A. 1975, 'The concept of a linguistic variable and its application to approximate reasoning—I', *Information Sciences*, vol. 8, no. 3, pp. 199-249.

Zhang, Y.-W. & Guo, R.-F. 2013, 'Power-aware scheduling algorithms for sporadic tasks in real-time systems', *Journal of Systems and Software*, vol. 86, no. 10, pp. 2611-9.

Zhang, Y., Lu, C., Zhang, H. & Han, J. 2011, 'Active vibration isolation system integrated optimization based on multi-objective genetic algorithm', *IEEE 2nd International Conference on Computing, Control and Industrial Engineering (CCIE)*, vol. 1, pp. 258-61.

Zhang, Y. & Xu, C. 2012, 'Service Replacement Algorithm Using Complex Network', *Transactions on Edutainment VIII*, vol. 7220, Springer Berlin Heidelberg, pp. 221-31.

Zhu, X., Young, D., Watson, B.J., Wang, Z., Rolia, J., Singhal, S., Mckee, B., Hyser, C., Gmach, D. & Gardner, R. 2009, '1000 islands: An integrated approach to resource management for virtualized data centers', *Cluster Computing*, vol. 12, no. 1, pp. 45-57.

Zomaya, A.Y. & Yee-Hwei, T. 2001, 'Observations on using genetic algorithms for dynamic load-balancing', *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 9, pp. 899-911.

APPENDIX: ABBREVIATIONS

ALB	Auto Load Balancer
API	Applications Programming Interface
AS-ORM	Autonomic System for Optimal Resource Management
BN	Bayesian Network
BoT	Bag-of-Tasks
CN	Computational Node
CPT	Conditional Probability Tables
CPU	Central Processing Unit
CTS	Central Task Scheduler
DAG	Directed Acyclic Graph
DBN	Dynamic Bayesian Network
DBN-WP	Dynamic Bayesian Network based Workload Prediction
DPM	Distributed Power Management
DRS	Distributed Resource Scheduler
DS	Data Storage
DSS	Decision Support System
ES	Expert System
ES&NN-WP	Expert System and Neural Network based Workload Prediction
FLS	Fuzzy Logic System
FP-TBSLB	Fuzzy Predictable Task-based System Load Balancing
GB	Global Blackboard
HPC	High-Performance Computing
HTC	High-Throughput Computing
IaaS	Infrastructure as a Service

MOGA	Multi-Objective Genetic Algorithm
MOPSO	Multi-Objective Particle Swarm Optimization
MOTS	Multi-Objective Task Scheduling
MOTS-PSO/GA	Multi-Objective Task Scheduling by applying Particle Swarm Optimization and Genetic Algorithm
NIST	National Institute of Standards and Technology
NN	Neural Network
OS	Operating System
PaaS	Platform as a Service
PM	Physical Machine
PPVM	Poorly Performed VM
PSO	Particle Swarm Optimization
QoS	Quality of Service
RMS	Resource Management System
RW	Roulette Wheel
SaaS	Software as a Service
SLA	Service Level Agreement
SMILE	Structural Modeling, Inference, and Learning Engine
SPV	Small Position Value
VIM	Virtual Infrastructure Managers
VM	Virtual Machine
VMM	Virtual Machine Monitor
WP	Workload Prediction