AUTONOMIC APPLICATION AND RESOURCE MANAGEMENT IN VIRTUALIZED DISTRIBUTED COMPUTING SYSTEMS

By

JING XU

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2011

To my parents, my husband, and my wonderful babies

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Jose Fortes for his endless patience, great motivation, excellent guidance and a tremendous amount of support throughout my PhD study at ACIS lab. I am greatly indebted to him for constantly supporting me and patiently giving me time and helping me grow.

I am very grateful to the other members of my supervisory committee, Prof. Renato Figueiredo, Prof. Pramod Khargoneka, and Prof. Zhigang Chen for taking time out of their busy schedules to review my work. Their insightful suggestions are very helpful for improving my work.

My heartfelt thanks is extended to my current and former fellow ACIS lab members. The lab provides me a pleasant place to conduct my research.

A special note of gratitude is due to Prof. Zhen'an Liu, my formal advisor at the University of Science and Technology of China, for encouraging and assisting me to pursue my Ph.D. overseas.

Finally, and most importantly, I would like to thank my parents. Their unconditional love are always with me. I owe everything that I have achieved to them.

There were happy times, there were tough times, there were exciting times, and there were unbearable times. All through this, my husband, Ming, has been with me, encouraging, supporting, and loving.

This dissertation work is supported in part by National Science Foundation grants No. OCI-0721867, CNS-0540304, CNS-0821622, CNS-0855123, IIP-0758596, IIP-0932023, the BellSouth Foundation, an Intel grant (IT R Council), and equipment awards from DURIP and IBM.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

AUTONOMIC APPLICATION AND RESOURCE MANAGEMENT IN VIRTUALIZED DISTRIBUTED COMPUTING SYSTEMS

By

Jing Xu

May 2011

Chair: Jose Fortes
Major: Electrical and Computer Engineering

Large-scale distributed computing systems, such as computational grids and enterprise data centers, present complex management challenges. Such systems experience inherent dynamism due to unpredictable resource availability and usage, or/and highly dynamic workloads. By introducing a layer of abstraction, virtualization technology provides ways of provisioning and customizing resource environments as needed, and migrating workloads to adapt to dynamic changes. However, the scale of such computing systems makes it extremely hard to control them manually by one or more human operators.

Our solution is to incorporate autonomic capabilities into the management of applications and resources in grid and data center environments to reduce direct human intervention. Such capabilities are accomplished through a two-level feedback-control framework in which local controllers at the application level have detailed information about the applications and allow independent adaptation and optimization. The global controller at the resource level collects resource information and optimizes the system behavior from a global perspective. It also acts as a coordinator when conflicts occur at different local controllers.

For grid environments, the proposed two-level control system is studied in the context of In-VIGO, a grid-computing system that provides application services on-demand using dynamically instantiated virtual machines, networks, data and

applications. Local controllers utilize application-specific information for tracking and predicting the performance of jobs executing on grid resources, which is then used to guide the scheduling/rescheduling decisions. Its effectiveness has been evaluated for CPU-intensive jobs with relatively short execution times (ranging from tens of seconds to less than an hour) on resources with highly variable loads. The results show that In-VIGO jobs managed by the two-level controllers consistently meet their execution deadlines under varying load conditions and gracefully recover from unexpected failures. Under the most dynamic and heavy loading environment created by the experiments, the average job runtime of the proposed approach is 10% and 20% shorter than two other competing scheduling strategies, one using round-robin and the other using the same scheduling as the proposed approach but without rescheduling actions. The percentage of jobs meeting their predefined deadlines is improved by 40% and 50%, respectively.

In a virtualized data center, the two-level control system is designed to deliver performance guarantees while optimizing resource usage, and also other important aspects of data centers such as power and cooling costs. At the application level, two fuzzy-logic-based methods - fuzzy modeling and fuzzy prediction - are proposed to estimate the resource demands for dynamic workloads. The global controller at the resource level tries to find the optimal resource allocation and virtual machine (VM) placement/replacement, with multiple objectives including the elimination of thermal hotspots, the minimization of total power consumption, and the efficient use of resources. The problem is posed as a multi-objective combinatorial optimization problem and an improved genetic algorithm with fuzzy multi-objective evaluation is proposed for efficiently searching the large solution space and conveniently combining possibly conflicting objectives. An online local search algorithm using multi-objective optimization and stabilization techniques is designed for dynamically changing virtual machine placement to quickly adapt to changes in system conditions or workloads. The proposed

approaches are implemented and evaluated on a virtualized testbed built upon an IBM BladeCenter. Under both synthetic and real-world Web workloads the local controller is validated to accurately estimate resource needs (the difference is within 5%) using fuzzy modeling and fuzzy prediction approaches. The global controller for determining virtual machine placement is tested with simulation-based experiments over a wide range of problem sizes and the results show that the multi-objective optimization using genetic algorithm achieve good balance among different objectives, resulting in relatively low values for power consumption, peak temperature, and resource wastage. For the dynamic virtual machine migration problem, experimental evaluations are conducted using a mix of types of workloads to emulate the variety and dynamics of data center workloads. The results indicate that the proposed multi-objective optimization with stabilization significantly reduces unnecessary VM migration and unstable host selection by up to 80% and also improves the application performance by up to 30% and the efficiencies of power usage by up to 20%.

The rapid growth of computing systems raises new challenges for centralized management at the global-control level in the proposed two-level architecture. A network of cooperative controllers is proposed in this work, each managing a subset of resources and collectively collaborating to manage the entire system. The proposed network model is validated on a testbed for In-VIGO and the results show that the decentralized and cooperative nature of the system yields a number of desirable properties, including efficiency, robustness, and scalability under a highly dynamic environment.

CHAPTER 1
INTRODUCTION AND BACKGROUND

Today's computing infrastructures are becoming increasingly large scale. Examples include scientific grids which harness resources from different domains for distributed and cooperative computing, and enterprise data centers which can potentially house thousands of physical servers. Common in these systems is the resource sharing among a variety of applications with different resource requirements and possibly dynamic workloads. Virtualization offers a new approach to sharing resources, by allowing the provisioning and customizing of computing environments as needed, and migrating workloads to adapt to changes. However, the growing management complexity due to the increased system size and the inherent dynamism experienced by both systems pose great challenges in application and resource management of such systems.

There is a growing interest in integrating autonomic capabilities into computer systems, aiming to develop self-management capability to overcome the growing complexity of system management. The work presented in this dissertation aims to address the challenges of autonomic management in large-scale distributed computing systems, targeting grid and data center environments. In this chapter, we first introduce the background of these two types of computing systems, and their management challenges are also discussed. The second part briefly introduces the basic idea of autonomic computing and our proposed approach to achieve autonomic application and resource management in grid and data center environments. The last part shows the roadmap of the whole dissertation.

1.1 Grids

A grid is a type of distributed system that enables the sharing, selection, and aggregation of resources distributed across "multiple" administrative domains based on their availability, capacity, performance, cost and users' quality-of-service requirements

[1]. Due to the lack of central control over the resources, the availability of grid resources cannot be guaranteed and resource utilization is hard to predict, which prevent the use of grid systems as one large cohesive set of resources for the users and the provision of performance guarantees. The following summarizes the key characteristics of grid environments.

Heterogeneity: In grid systems, resources from different domains tend to be heterogeneous, in terms of their platform, capacities and performance. The applications also have diverse characteristics and resource needs.

Dynamism: The computing environment in grids is continuously changing during the lifetime of an application, including the availability and the states of resources. Due to the large decentralized and asynchronous nature of grid environments, it is hard to obtain a complete knowledge of global system state.

Uncertainty: The dynamic nature of grid resources causes unpredictable and changing behaviors that can only be detected and managed at runtime. Furthermore, as the scale of system and application increases, the probability and frequency of failures increase dramatically.

Many applications running in grids have stringent end-to-end performance requirements across multiple computational resources that are possibly geographically separated. For example, educational usage scenarios of grid systems like PUNCH [2] and In-VIGO [3], offer many examples of large numbers of users running many relatively small jobs over concentrated periods of times (e.g. before homework deadlines). Unexpected long running times for supposedly quick tasks do discourage further use of a grid-computing system, and generate user discontent and customer losses. In the context of grid systems, the problem addressed in this dissertation is how to enable such workloads to adapt to highly dynamic and fault-prone grid environments, for obtaining consistent performance and satisfying application requirements automatically.

1.2 Data Centers

A data center is defined as ”an environmentally controlled centralized facility providing business services by securely delivering applications and data across a network to remote users” [4]. In a traditional data center environment [5], applications are deployed at different servers to provide necessary security and performance isolation. As more applications are deployed, the number of servers also grows rapidly. This leads to what is referred to as “server sprawl”, i.e., a large number of underutilized, and heterogeneous servers.

Applications hosted in data center are usually business-critical applications with quality-of-service (QoS) requirements. Such applications typically have time-varying workloads with high peak-to-average ratio, resulting in dynamically changing resource demands. Traditional over-provisioning approaches used for meeting peak demand usually lead to low resource utilization. In addition, the power consumption and cooling costs [6][7][8][9] become great concerns in recent years. According to a report in [10], the amount of energy used to power the world’s data center servers doubled in a five-year span due mainly to an increase in demand for Internet services, such as music and video downloads.

All these difficulties in data center resource management promoted the usage of virtualization technology aiming to produce more cost-efficient data centers, hereon referred as virtualized data centers. Server virtualization [11][12] entails the possibility of one physical server hosting multiple independent virtual machines, and the ability of transparently moving workloads from one physical server to another through virtual machine migration. On the other hand, these capabilities create great demands on system management, especially for large-scale enterprise data centers that contain thousands of servers and even more virtual machines. This dissertation addresses the questions on how to dynamically allocate resources among virtual machines and their applications, and how to map/remap those virtual machines to physical servers, in order

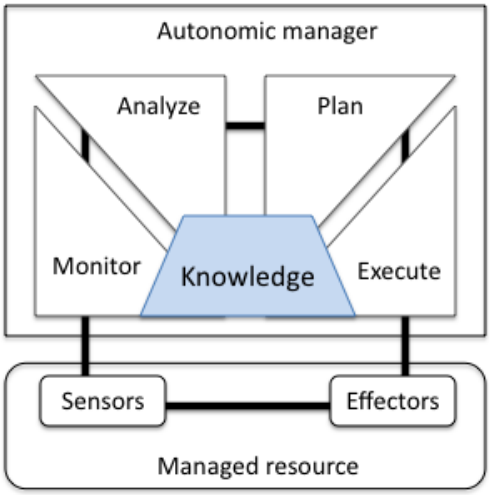

Figure 1-1. MAPE control loop in an autonomic manager.

to deliver performance guarantees while simultaneously optimizing resource usage, and other important features of data centers such as power consumption and cooling costs.

1.3 Autonomic Computing

As discussed in preceding sections, application and resource management in both data centers and grids brings great challenges due to the increasing scale and inherited dynamism. Using ad-hoc manual tuning performed by human operators is impractical. Autonomic Computing, as defined by IBM in 2001 [13], aims to build computer systems that manage themselves much in the same way our autonomic nervous system [14] regulates and protects our bodies. By integrating monitoring, decision processing and actuation into system components, a control loop provides the basic backbone structure for an autonomic computing system.

IBM represents this control loop as the MAPE (Monitor-Analyze-Plan-Execute) loop [15]. Figure 1-1 depicts the components and key interactions of MAPE architecture. The managed resource represents what is being managed and it could be any software or hardware resource. Sensors are used to measure usage and performance of managed resources. For example, for a web server, that could include response time of client requests, and utilization of a server's CPU and memory. Effectors provide a way

to change the behavior of the managed resources. An autonomic manager is the component that implements an intelligent control loop. The monitored data is first filtered and correlated by the monitoring component, and the refined data is processed by the analysis component for the purposes such as forecasting and problem determination. Planning component constructs an order of actions to accomplish high-level goals. The execute component controls the execution of the actions. Knowledge about the managed resources is accessible to all the four components.

The following section introduces the proposed two-level control system to address the challenges of the application and resource management in large-scale computing system. The autonomic managers (a.k.a controllers) implemented in the system utilize methods and techniques from machine learning, optimization, and control theory to optimize application performance, improve resource use and reduce related costs.

1.4 Proposed Approach

In the scenarios considered in our work, the management of computing systems typically involves two parties: applications (and corresponding users) and resources (and corresponding resource owners). The management goals for these two parties are typically different. From applications' point of view, the performance requirements should be satisfied. For example, "the response time of a web request should not exceed 5 seconds". From the viewpoint of resources or resource owners, the resource pool is shared among applications and the use of resources and returned investment should be maximized. This dissertation proposes a two-level control system to address the management objectives at both the application level and the resource level, which allows flexible and independent optimizations for both parties.

1.4.1 Two-Level Control System

In the two-level control system, a local controller at the application level has detailed information about the application, which can be used to optimize its own performance, make resource requests and adapt to the dynamically changing environment. The global

controller at the resource level collects resource information and responds the requests from local controller. It also acts as a coordinator when conflicts occur at different local controllers. Local management allows applications to control their behaviors and optimize their performance independently while global management optimizes the system behavior from a global perspective. This two-level resource control system is preferred over the more obvious centralized approach in which all the control functions are implemented at one centralized location. The internal complexities of control functions are compressed by local controllers into straightforward resource requests using locally available information. Moreover, it is easy to add, change or remove local controllers without affecting the global controller.

The proposed two-level control architecture is developed in both grid and data center environments. Chapter 3 explains how the local controllers cooperate with the global controller to optimize application performance and recover from performance fault caused by dynamically changing resource usage in a grid environment. Chapter 4 and 5 discusses the proposed two-level autonomic resource management developed in a virtualized data center. A local controller implemented at each virtual machine uses adaptive online learning approaches to predict resource demands for its application and sends resource requests to the global controller to meet its application performance requirements and reduce resource cost. The global controller determines the optimal virtual machine placement and resource allocation by taking several factors into consideration including resource usage, power consumption and cooling costs.

1.4.2 Optimization

Optimization of management related measures is one of the most important goals of the proposed two-level control system. For example, in virtualized data center environment, the main task of the local controller is to optimize the set of resources needed by an application running in the virtual machine in order to minimize the resource cost of each individual application. The global controller tries to maximize

the total profit from running users' applications and to minimize the costs from power consumption, cooling, and inefficient use of resources. The following lists the optimization problems addressed in each chapter followed by a brief discussion of the main optimization approaches used in the two-level control system.

1. Optimize application performance and improve fault tolerance under a dynamically changing resource environment, as exemplified by autonomic application management in the In-VIGO grid system (Chapter 3: Autonomic Application Management in Grid Systems).

2. Optimize resource allocation in a virtualized data center so as to maximize the obtained profit while meeting application SLA under dynamically changing workloads (Chapter 4: Autonomic Resource Management in Virtualized Data Centers).

3. Optimize virtual machine placement and migration considering multiple objectives in a virtualized computing environment (Chapter 5: Multi-objective Optimization in Virtual Machine Management).

Fuzzy-logic-based Optimization

To enable automatic and adaptive resource provisioning under dynamically chancing workloads, two fuzzy-logic-based methods - fuzzy modeling and fuzzy prediction - are proposed to guide resource allocation based on online measurements. Both approaches have an adaptive-learning ability to capture transient or unexpected workload changes. Specifically, the fuzzy modeling approach characterizes the relationship between the workloads and the corresponding resource demands, while the fuzzy prediction builds a mapping from recent resource usage to future resource needs. The knowledge obtained by the fuzzy-logic-based system can be easily updated when new information is available to adapt to the system changes and reflect the most recent system conditions. These approaches make no underlying assumption of the workload characteristics, and can learn any type of relationship very fast. Especially, the fuzzy-logic-based system can efficiently model a nonlinear system with dynamically changing operating conditions [16][17][18].

Multi-objective Combinatorial Optimization

The global controller at the resource level of virtualized data centers manages all the virtual machines and determines where to place them, when and where to migrate them as needed. To address the complex tradeoffs between system performance, reliability, and cost of data centers, the global controller utilizes a multi-objective optimization approach to simultaneously optimize multiple objectives including efficiently using multidimensional resources, avoiding thermal hotspots, and reducing energy consumption.

The problem of virtual machine placement can be reduced to multi-dimensional bin-packing, which is NP-hard combinatorial optimization problem. To efficiently search potentially large solution space for large-scale data centers, an improved genetic algorithm for mapping virtual machines to physical hosts is proposed in this dissertation. The genetic algorithm (GA) is a search heuristic which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. Solutions in search space are represented as strings using an encoding schema. In our approach, encoding and genetic operators for crossover and mutation are modified to suit the structure of virtual machine placement problem and improve the searching performance. Another novelty of our proposed approach is the use of a fuzzy multi-objective approach for evaluating solutions obtained by the genetic algorithm. Fuzzy logic allows the mapping of values of different objectives into linguistic values characterizing levels of satisfaction and provides a convenient way of combining conflicting objectives without specifying weights or preference among different objectives [19].

1.4.3 Decentralized Management Network

The ever-increasing scale of computing system raises challenges for the design of global management in the proposed two-level architecture. The centralized global manager introduces a single point of failure and can become a bottleneck in handling all

information and management tasks in large-scale systems. This dissertation adopted a solution that uses a network of cooperative local autonomic managers, each managing a subset of resources and applications and collectively collaborating to manage the entire system. A self-organizing management network is built through the building and rebuilding of a dynamic local neighborhood, to provide scalable, robust and efficient information sharing for local managers to control, monitor and optimize performance of the system.

1.5 Roadmap

The rest of this proposal is organized as follows:

Chapter 2: This chapter reviews the research work closely related to this dissertation.

Chapter 3: This chapter presents our solutions for autonomic application management in the context of the In-VIGO grid system to enable self-healing and self-optimization.

Chapter 4: This chapter presents autonomic resource management in virtualized data centers using a two-level control system. The fuzzy-logic-based approaches are used to adaptively model resource demands for time-varying workloads so as to optimize resource usage.

Chapter 5: This chapter presents our solution for optimizing virtual machine placement and migration considering multiple objectives. An improved genetic algorithm with fuzzy-logic-based evaluation approach is proposed to find optimal mapping of virtual machines. A cross-layer control approach is used to manage the dynamic migration to cope with dynamic changes of workloads and system conditions.

Chapter 6: This chapter discusses decentralized autonomic systems by utilizing cooperative local autonomic managers and self-organizing management overlays, as exemplified by a decentralized autonomic application management in the In-VIGO grid system.

Chapter 7: This chapter concludes the dissertation and discusses the future work.

CHAPTER 2
RELATED WORK

The research work related to this dissertation spans several areas including job scheduling and workload management in computing clusters, application and resource management in grids, and autonomic management of resources, workloads, and power/thermal in virtualized data centers. The following briefly reviews the work in these areas. More related work will be discussed in detail in the following chapters as follows.

1. Chapter 3 discusses the approaches on resource-usage prediction, resource discovery and allocation, and job rescheduling proposed in related work.

2. Chapter 4 discusses the resource management in virtualized data centers based on rule-based systems, control theory, mathematical models, and reinforcement learning.

3. Chapter 5 presents the related work on workload/application placement on shared resources and dynamic resource allocation and virtual machine migration in virtualized data centers.

4. Chapter 6 presents the related work on decentralized resource management based on agent-based systems and gossip algorithms.

2.1 Workload Management and Job Scheduling in Clusters

In computing clusters, the workload management system typically involves a centralized scheduler which assigns computing resources to computational tasks initiated by end users [20]. By having full control over all the applications and resources, the scheduler takes the responsibility of collecting information of applications and resources, allocating resources, and controlling job execution. The jobs submitted to the system may have various resource requirements such as different number of nodes, different processor types and memory size. They may also have other constraints such as job response times or job deadlines. The workload management software maximizes the use of resources to jobs, giving competing users' requirements and enforcing local policies. Etsion and Tsafrir [21] give a short survey of some common commercial workload management software for computing clusters including Maui[22], LoadLeveler

[23], PBS [24] and LSF [25]. The most commonly used scheduling algorithms include FCFS (first come first served), fair-share, and backfilling. There are a number of research efforts analyzing job scheduling and impact of job scheduling on system performance for large-scale computing systems such as paper [26][27][28][29]. Some of them use user-provided estimation of job runtime to improve overall resource utilization [30][31]. However, these approaches are highly dependent on the accuracy of user estimates of job runtimes, which have been repeatedly demonstrated to be highly inaccurate [32][33][34]. Recently, increasing attention has been paid to fault-aware scheduling in high-performance computing. In [35], Zhang et al. suggest utilizing temporal and spatial correlations among failure events for better scheduling. Oliner et al. [36] present a fault-aware job scheduling algorithm for Blue Gene/L systems by exploiting node failure probabilities. In [37], the proposed scheduling system dynamically adjusting the placement of active jobs (i.e., running jobs) to avoid imminent failures discovered by on-line failure predictors.

2.2 Application and Resource Management in Grids

The traditional workload management in computing clusters is mostly system-centric and does not consider individual application needs. In addition, it does not address the issue of heterogeneity and unpredictability in large-scale computing resources such as grids. Some recent projects explore the effect of different application requirements and apply adaptive scheduling mechanisms based on runtime resource availability. For example, Application-Level Scheduling Project (AppLes) [38] developed an approach that incorporates static and dynamic resource information, application- and user-specific information and performance predictions into job scheduling. Each application is fitted with a customized scheduling agent which performs resource discovery and selection, schedule generation and selection, and application execution. Agents also use the services offered by the NWS (Network Weather Service) [39] to monitor the varying performance of available resources. Their approach requires

a certain amount custom work on each individual application and their performance models are application-specific and not easy to re-apply to new applications. Condor [40] is a high-throughput computing environment that can manage a large collection of distributively owned computing resources. The Condor system simplifies job submission by acting as matchmaker between jobs and resources. Both job requirements and resource properties are described using ClassAd language. The Condor's scheduling system combines dedicated and opportunistic scheduling to allow programs to use unused cycles on idle resources. Opportunistic scheduling assumes that jobs may be interrupted at any time, and is capable of migrating a job to a different host. A major limitation of Condor is that the condor checkpointing code must be linked in with the user's source code which sometimes may not be available. In addition, Condor cannot migrate processes which execute fork() or exec(), or communicate with other processes [41]. GrADS [42] provides a close-loop execution environment, with realtime monitors providing feedback about system performance. Applications are encapsulated as configurable objects, which can be optimized for execution on grid resources. It employs analytical models that are constructed semi-automatically from empirical models (historical data/sample execution data), in order to estimate the performance of a workflow component on a single grid resource. GrADS utilizes Autopilot [43] to assess application's progress using performance contracts between the application demands and resource capabilities. Once the contract is violated, the rescheduler [44] of the GrADS takes corrective actions, such as rescheduling the application via the stop/restart approach or process swapping. Klaus Krauter et al. [45] developed an abstract model and a comprehensive taxonomy for describing resource management architectures, and surveyed different resource management approaches in fifteen representative Grid system.

To automate application management in grid environment, this dissertation proposed to use machine learning approaches to estimate resource demands and

predict application performance using both historical data and online information, which in turn guides the future control actions such as rescheduling. It does not require detailed application information or user-provided estimation. The resource estimation and selection, performance prediction and job rescheduling are performed locally with the facility of global scheduler, which significantly reduces the complexity of the centralized management. Our proposed work aims to enable autonomic capabilities including self-healing and self-optimizing into a general application-centric management framework. The context of interest is virtualized grid systems and data centers, which is discussed in the next section.

2.3 Virtualized Systems

Virtualization is generally a method of making a physical entity act as multiple, independent logical entities. The goal of virtualization is to provide logical view and control of physical infrastructure in order to enable greater optimization, utilization and simplification of management. The following discusses how virtualization benefit the management of grid systems and data centers.

2.3.1 Virtualization in Grid Computing

Virtualization can greatly simplify grid computing by decoupling the architecture and user-perceived behavior of resources from their physical implementations. The capabilities and functions enabled by virtualization help hide the heterogeneity and deal with complexity inherited in grid computing systems. Several projects have studied the integration of virtualization in grid systems. The main idea of the In-VIGO [3] system is to construct virtual grids out of physical resources by adding virtualization layers to the traditional grid middleware, in order to enable on-demand, dynamic virtual resources for application-specific, user-specific grid computing. The Virtual Workspace Service [46] exposes the functionality needed to manage workspaces - abstraction of execution environments implemented through virtual machines (VMs). Santhanam et al. [47] considered integrating VMs into grid systems for sandboxing purposes and

explored different design of VM-enabled sandboxes. Virtual Computing Laboratory (VCL) [48] is an open source implementation of a secure production-level on-demand utility computing and services oriented technology for access to solutions based on virtualized resources. It has a web portal where users may request a virtual machine that VCL configures on-demand. VCL provides an approach to mapping and managing user application and service needs to available local or distributed software images and hardware resources. PlanetLab [49] is a large-scale, distributed platform for new network services. PlanetLab uses virtualization containers, called slide to manage resource allocation and to achieve isolation between a potentially large number of long-lived, independent services. However, it does not provide assurance over the underlying resources bound to each virtual machine and no notifications when a slice's resource allotment changes due to contention or failure.

2.3.2 Virtualization in Data Centers

By introducing a layer of abstraction, virtualization enables multiple virtual machines to run on the same physical server while guaranteeing the necessary security and performance isolation. Virtual machine migration makes it possible to transparently move workloads from one physical server to another with no impact to end users. With fine-grained dynamic resource allocation to virtual machines being enabled, virtualization provides flexible resource management.

Resource Management in virtualized data centers have gained a lot of attention in both academic research [50][51][52][53] and commercial products [12][54]. Given server consolidation, flexible management, and easy deployment provided by virtualization, many research efforts have focused on data center optimization such as maximizing profit [52][51], and minimizing power and cooling costs [6]. Virtual machine migration capability is often used to adapt to dynamic workload changes and improve efficiency of resource usage [55][56][57][58][59]. However, the high resource overhead and performance loss incurred by migration prohibits unlimited usage of this mechanism.

How to minimize the impact of migration as well as satisfy other objectives and constraints needs to be addressed in data center optimization. The following section broadly reviews the ongoing research work on resource management optimization, with focus on data center environments. A more comprehensive survey on the use of optimization techniques in resource management of data centers is presented in Appendix I.

Control theory has also been applied extensively to computer systems for resource management and performance control. A survey of feedback performance control in various computing systems is presented in [60]. To model the dynamics of the controlled system (e.g., application workloads and their hosted VMs), most prior work employs an empirical and "black box" approach to system modeling by varying the inputs in the operating region and observing the corresponding outputs. However, the choice of model structures and model sizes and orders are hard to determine. Linear models are often applied because of their simplicity, but nonlinear behaviors are typically shown in such dynamic systems [61][18]. An adaptive integral controller is developed in [62] for dynamic sizing of a virtual machine based on its consumption such that the relative utilization of the VM can be maintained in spite of the changing demand. In [63], QoS-driven workload management was presented using a nested integral feedback controller, where the inner loop regulates the CPU utilization of a virtual container and the outer loop maintains the application-level response time at its target.

Predictive control and optimal control recently draw much attention in resource and power management of data centers. In [64], a predictive controller was developed to allocate CPU resource to a virtual container proactively by exploiting repeatable patterns in an application's resource demands. The predictive controllers use three different prediction algorithms based on a standard autoregressive (AR) model, a combined ANOVA-AR model, and a multi-pulse (MP) model. Some work such as [52][65][66] formulate the performance and power management in data centers as

dynamic optimization problem which either minimizes associated costs or maximizes total profits. Linear Quadratic Regulator (LQR), Model Predictive Control (MPC), and limited lookahead control (LLC) are used to solve the optimization problem.

In this dissertation, fuzzy-logic-based modeling and prediction approaches are used to estimate dynamic applications' resource needs in order to optimize resource usage of individual virtual machine (see Chapter 4. Compared to the control-based approach, the fuzzy control approach can efficiently describe dynamic nonlinear system behavior without involving heavy calculations or making any assumptions and simplification.

2.4 Optimization

Decision-making problems often appear in system management research including resource provisioning, resource allocation and application placement. These problems are characterized by a large space of potential solutions, with complex tradeoffs between system performance, reliability, manageability and cost. Some research (e.g., [55][56][57][58]) addresses resource and workload management as optimization or constraint satisfaction problems, allowing the use of operations research (OR) solution techniques, such as mathematical programming and meta-heuristics.

Dynamic resource allocation to applications in order to adapt to workload variations has been studied in many research efforts including [67][68][69][70][71][50]. To handle resource competition among multiple applications, utility functions are used in [72][7][50] to represent the value of resources for different applications. The goal is to maximize the global utility by allocating resources among competing applications. Similarly, some work such as [52][6] translates allocated resources into business value such as revenue or penalty for meeting or violating predefined Service Level Agreement (SLA). The optimization problem is to maximize the total profit which is equal to the revenue minus the associated costs and penalties. Zhang et al. [67] and Zhu et al. [68] formulated resource allocation and assignment as combinatorial optimization problems and showed that they are NP-hard problems. Intelligent search such as genetic algorithms, simulated

annealing, tabu search and other problem-specific heuristic algorithms [19] are applied to solve this type of problems.

Ever increasing power density and dissipation in data centers has turned energy management into a key concern. Many research efforts have attempted to control power at different levels. At the finest granularity, CPU voltage and frequency can be manipulated to conserve power. This so called dynamic voltage/frequency scaling (DVFS) technique has been widely used in both embedded and general-purpose systems. In [6], a DVFS policy based on feedback control and queueing theory is presented for minimizing operational costs while meeting the SLAs. In [73][74], performance management and power management are coordinated to achieve specified power and performance objectives. A lot of efforts also have been made to address power efficiency at the level of server clusters or data centers. In [7][75][76], consolidating workloads and turning off unloaded servers are used to achieve energy savings.

However, most work has focused on optimizing only one or at most two management aspects of data centers, by separately managing either the platform layer (e.g., power and thermal management) or the virtualization layer (e.g., VM provisioning and migration, application performance management). The proposed two-level control system (see Chpater 5) utilizes information from both the virtualization and the platform layers, for simultaneously optimizing multiple objectives including making efficient usage of multidimensional resources, avoiding thermal hotspots, and reducing energy consumption.

CHAPTER 3
AUTONOMIC APPLICATION MANAGEMENT IN GRID SYSTEMS

This chapter presents our work on applying autonomic management to grid middleware to achieve fault-tolerance and improve application performance on non-dedicated resources with varying loads. This work is developed and implemented on a testbed for In-VIGO, a grid-computing infrastructure created by the ACIS Laboratory.

3.1 Problem Description

In traditional computing systems, resource management and job scheduling have been extensively studied. The management system components such as workload schedulers, resource manager and workflow engines, have a huge variety of implementations. These systems are typically designed and operated in a relatively static environment over which the manager has a complete control. The manager can implement the mechanisms and policies as needed for efficient use of the resources under his/her control. However, this does not apply to grid systems in which management must deal with the heterogeneity, dynamism, and uncertainty of grid resources.

3.1.1 Challenges of Resource Management in Grids

Grid computing [77] enables users to share resources distributed across administrative domains. Idle and low-priority shared cycles from non-dedicated machines form an important class of grid resources. Two distinguishing features of such resources are, (a) intermittent participation (either voluntary or due to failure), and (b) highly variable load due to their non-dedicated nature. This lack of performance guarantees makes it difficult to exploit such resources to support workloads that require a specified quality of service (QoS).

3.1.2 Workloads of Interest

The workloads of interest in this work are scientific in nature, batch-oriented and characterized by relatively short execution times (ranging from tens of seconds to less

than one hour). Educational usage scenarios of grid-computing systems like PUNCH [2] and In-VIGO [3], offer many examples of large numbers of users running many relatively small jobs of scientific tools over concentrated periods of times (e.g. before homework deadlines). In spite of the short duration of these jobs, they play an important role in shaping the user experiences with a grid system. Long response times for short jobs are less well tolerated than long execution times for tools known to take a long time to run. Unexpected long running times for supposedly quick tasks do discourage further use of a grid system, and generate user discontent and customer losses. Our goal is to enable such workloads to adapt to the highly dynamic and fault-prone grid environments that result from sharing non-dedicated resources.

The following sections describe an approach to autonomic computing that is application-centric and leverages the use of virtualization technology in grid computing. The ideas have general applicability but the context is that of In-VIGO, a grid system that provides application services on-demand using dynamically instantiated virtual machines, networks, data and applications.

3.2 In-VIGO Introduction

In-VIGO [3] (In-Virtual Information Grid Organizations), is a grid-computing infrastructure designed to support computational tools for engineering and science research on grid resources. In-VIGO has been configured and tested to run a variety of applications in material science, bioinformatics, and computer achitecture etc. Its distinctive feature is the extensive use of virtualization technologies to provide secure execution environments as needed by tools and users. A detailed discussion of its design and implementation appears in [3].

3.2.1 Virtualization in In-VIGO

The In-VIGO architecture is built upon components that allow for virtualization of grid resources and user interfaces. A virtual file system facilitates data transfer and access across gird resources. Virtual machines provide isolated and customizable

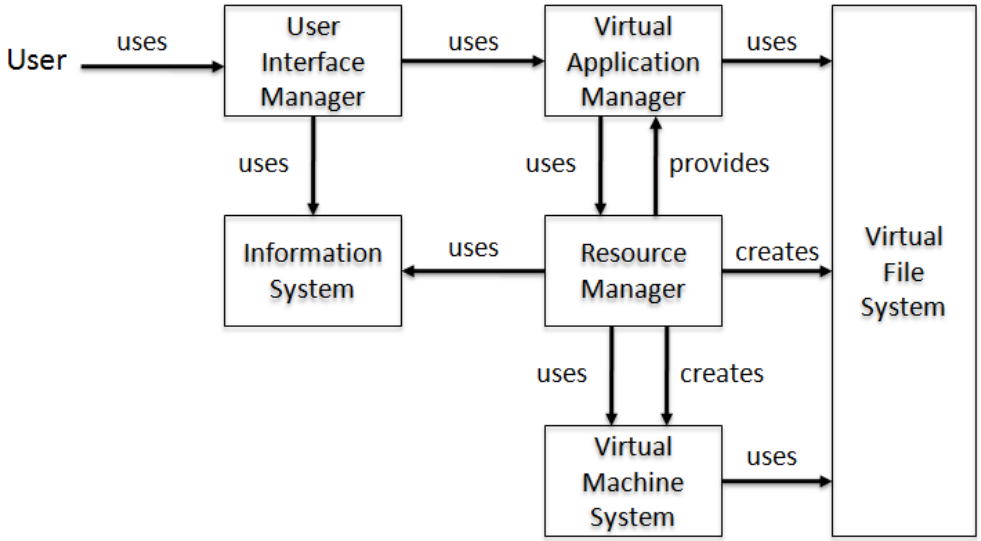

Figure 3-1. High-level block diagram of In-VIGO showing use-relationships and control interactions among its major components

execution environment for applications. Virtual applications (VAPs) allow modification and extension of any computational tool's behavior. By"wrapping" the actual tools with additional software, the In-VIGO VAPs enable customization of an interface to allow a physical application to be seen by different users as multiple different applications with different capabilities.

3.2.2 Architecture of In-VIGO

The major components of In-VIGO are shown in Figure 3-1. Users interact with the middleware via a web-browser-based portal. Typically, a user initiates an application session, to run one or more instances of a tool on grid resources. Each application session is managed by an instance of a Virtual Application Manager (VAM) component dynamically created for the application. The user's actions, for example setting up input parameters, importing data files, executing tools and retrieving results, are directed via the User Interface Manager (UIM) to the VAM. The VAM interacts with the Resource Manager (RM) to launch the necessary tool executions on the available resources. The RM component obtains the status of available resources from the Information System (IS) component, and allocates resources for a tool execution.

The resource management and job scheduling functions are mainly implemented in the VAM and RM components. A VAM manages all actions associated with user requests during an application session and responds to these requests by requesting the RM to provide the necessary resources and launching the tasks. A VAM has session-specific information, e.g. input parameters specified by the user, as well as application-specific knowledge, i.e. the logic needed to respond to user actions, specified in an application configuration file. The RM component that manages job executions on resources is an application-neutral generic interface to heterogeneous resource APIs.

The use of virtualization in In-VIGO, which enables on-demand creation of virtual resources for application-specific computations, has many advantages. However, our experiences show that the existing In-VIGO system still lacks robust and fault-tolerance features due to fluctuations of resource availability and usage. Investigations of dependable computing techniques in the context of grid computing [78][79], have shown that the practical way of handling faulty resources is to make the client, i.e. the submitting endpoint, responsible for the progress and failure handling of applications. In In-VIGO it is the VAM – the application job management component, – that the proposed approach aims to make autonomic, i.e. self-optimizing and self-healing.

3.3 Autonomic Virtual Application Management

The following section describes the proposed two-level control approach to autonomic application management, and the implementation via extensions to VAMs in In-VIGO.

3.3.1 Two-Level Control Architecture

Figure 3-2 provides a high-level view of a two-level control system in In-VIGO. At the application level, a VAM is created for each application session to implement the application-specific logic required to respond to user's requests. Each VAM session is monitored and controlled by its autonomic manager (AM). Information used by AM is

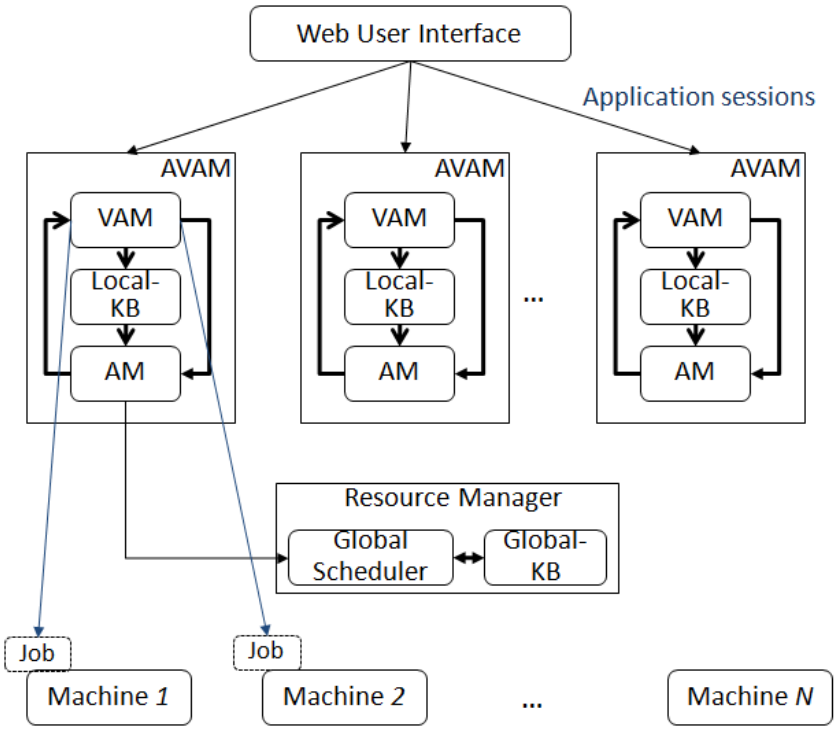

Figure 3-2. High-level view of Autonomic Virtual Application Manager (AVAM) in In-VIGO. This figure shows multiple VAMs that connect to the Resource Manager to submit jobs on multiple machines.

stored in the per-VAM knowledge base (also called Local KB). This includes user- and application-specific information (e.g. expected performance and input parameters) which is provided by the users through UIM, and dynamic resource and job status information monitored by the job and resource sensors.

At the resource level, the resource manager (RM) collects resource usage data such as current CPU load and free memory size, measured by a monitoring daemon on each machine and stores it to the global knowledge base (Global KB in Figure 3-2). Besides resource information, the Global KB also maintains applications' execution history that can be later retrieved by an AM to estimate resource demands for application executions. A global scheduler is a part of the RM, assisting each AM to discover and allocate the most appropriate resources for a given job.

Rather than directly forwarding application tasks to the RM component to determine the resource allocation, an AM manages resource allocation locally for the user's jobs,

by interpreting the current execution-specific information stored in the Local KB and the application execution history in the Global KB. With this two-level control architecture, applications make scheduling decisions themselves, adapting to the availability of resources and optimizing their own performance while the RM maintains the global resource information.

The next section explains in detail how self-optimization and self-healing are achieved by an AM and its interactions with the resource manager.

3.3.2 Autonomic Manager

Each AM consists of two components, the local (per-VAM) resource coordinator and the local job controller. The local resource coordinator uses a learning algorithm to predict the resource usage for each given input-specific run from previous runs' history records extracted from the Global KB. Based on the user's requirements and predicted resource usage, the local resource coordinator allocates the proper resources for the job by cooperating with the global resource scheduler. The allocation of resources by an AM for tasks allows optimal application-specific choice from the pool of available resources.

The local job controller is responsible for controlling the job execution to achieve reliable and optimized performance. At the time of job submission, a job sensor is invoked on the chosen machine by the job controller to monitor the job status. Based on the monitored job status and the predicted resource usage, the controller determines the progress of a given job run and compares it with the user's QoS requirements to decide control actions such as rescheduling the job to other resources. The actual submission and management of a job on the selected resource is accomplished via the RM component, which virtualizes the file space of the application via the GVFS component (see Figure 3-1). Each task gets its own copy of the file system to store its persistent state, i.e. file input/output. This makes In-VIGO jobs idempotent, and as a result there is no need to roll back jobs to their initial state when rescheduling them to alternate resources.

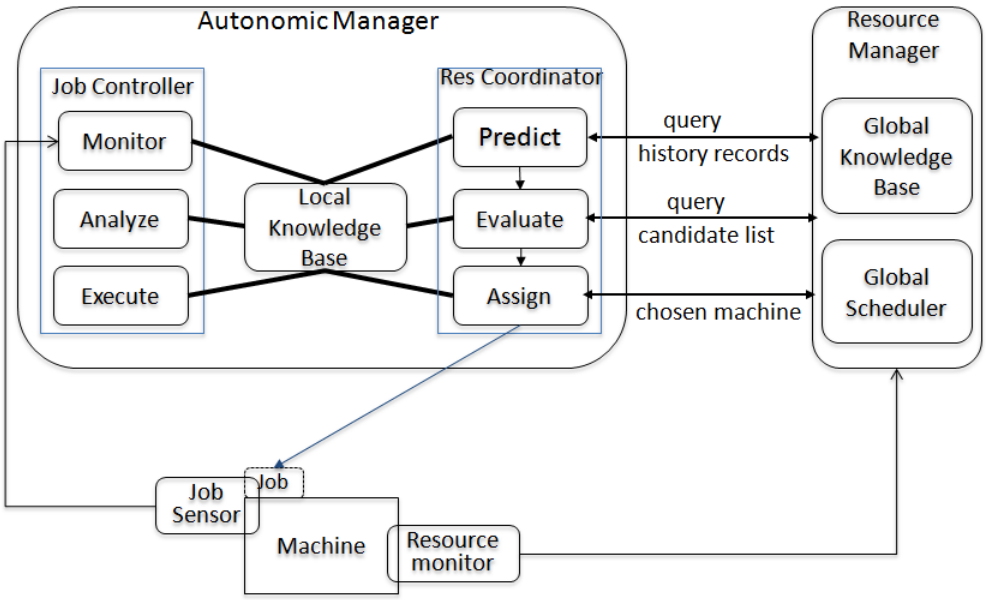

Figure 3-3. The major components of AVAM and their interactions.

Figure 3-3 illustrates the major functions of the local resource coordinator and job controller in an AM and the interactions among these functions and with other components. The local resource coordinator implements three functions named *predict, evaluate*, and *assign*. The functions *analyze, monitor*, and *execute* are implemented in the local job controller. When users execute tools in In-VIGO, they may input various parameters, request different resources and optionally set different performance expectations (e.g. the execution of the given job should finish in 10 minutes). To choose the appropriate resources, it is necessary to know the specific requirements, such as how much memory and CPU are needed for a given run. However, users are usually not able to provide accurate values for these properties. It is the role of the predict function in an AM to estimate the resource utilization of any given job, and hence free users from providing such estimates.

3.3.3 Resource Usage and Performance Prediction

In order to deliver application performance satisfying users' requirements, performance prediction such as job runtime provides critical information for job

scheduling decisions in grid systems. A large body of research work has been done on performance and resource usage predictions on computer systems (e.g., [80][81][82]). The following summarizes two types of approaches for performance prediction.

Model-based prediction: Work in [83][84] proposed a framework for HPC job performance prediction based on application signature and machine profile. An application's signature is a detailed summary of operations carried out by the application independent of any particular machine, for example, patterns of memory usage and communication. A machine profile provides the rates at which a machine can perform basic operations. An application's performance is predicted by mapping its application signature onto a machine profile. This approach requires direct knowledge of internal design of the application and machine architecture, although highly accurate predictions may be achieved with the detailed information and analysis.

Historical data-based prediction: This approach derives application predictions from historical observations of previous application runs. Dinda et al. [85] described and evaluated a system that is used to predict running time of a computation-bound task on a time-shared system, based on the assumption that execution time of such tasks is linear to the load of the hosting machine. It is shown that, among different predictive approaches, the simple linear time-series models are the most appropriate for host-load prediction in terms of predictive power and low overhead.

Another effort using historical data to predict application runtime is based on "similarity" of different job runs. By identifying "similar" jobs with certain job attributes, these approaches apply statistical methods to generate predictions. Local learning techniques have been investigated, which use historical data points to build a local model for approximation. The key is to define a distance metric to measure "nearness" between data points for describing the jobs.

In our case, one very important factor that must be considered in choosing the prediction method is its overhead since the jobs of interest are relatively short lived. In

order to avoid high overhead, the predict function is implemented using a memory-based learning algorithm. In [81] three learning algorithms are evaluated, and the results indicate the nearest-neighbor algorithm is the most efficient one. This algorithm is used in the prediction function of the resource coordinator. The basic idea behind this algorithm is reviewed in the next paragraph (see [81] for a longer discussion and possible improvements).

All the job execution history is stored in the Global KB that can be later queried for estimating resource requirements and job execution progress. The resources (e.g., CPU cycles and memory) consumed by a particular job run often depend on the input parameters supplied to a specific tool. Therefore, the "similarity" of two job runs of a tool is defined by the distance metric of two sets of input values. The distance between a given query-point and a data-point in the input space is computed as follows. For a string-type, we use exact match, and for a value-type, we compute the normalized distance using the standard Euclidean metric. The predict function applies the following steps:

1. Given the application tool's name and input vector V, an SQL query statement is created.

2. The SQL query is forwarded to the Global KB which keeps the records of the tool's previous runs. For every result that satisfies the query, the distance is computed.

3. The nearest neighbor to the given run's input parameters is extracted, and the whole record is selected from the Global KB including resource requirement such as CPU cycles, memory usage etc.

3.3.4 Resource Selection

After the predict function estimates the resource requirements for a particular job run, it queries the global scheduler for resources based on the prediction and user-specific requirements (e.g a Linux RedHat 5.0 system with at least 100 MB of memory). If there are resources satisfying the requirements, the global scheduler puts them in a candidate machine list and also fetches their properties such as CPU speeds,

and dynamic status (such as current load etc.) which are updated periodically by the resource monitor daemons (see Figure 3-3). This list of resources along with their status information is then sent back to the local resource coordinator.

The *evaluate* function determines which machine in the list is more preferable based on the resource usage prediction of the given job. For example, if the job is I/O intensive, a resource that is close to the file server is preferable. For CPU-intensive jobs, our goal is to find the resource on which the given job can run fast. Runtime does not only depend on a machine's CPU speed, but also is strongly related to the machine's load [81][85]. Therefore we use the following equations to estimate a job's execution time on each machine by assuming that the current load of the machine does not change in a relatively short period of time.

$$\begin{aligned} CPUtime &= \frac{CPUcycles}{CPUspeed} \\ Executiontime &= CPUtime(1 + Load) \end{aligned} \tag{3–1}$$

Using the above equations, the evaluate function calculates the predicted job runtimes on each machine in the candidate list, and puts the best ones in a sorted preference list.

3.3.5 Job Control

After choosing the best resource for a job, the assign function is responsible for submitting the job to the chosen resource, and storing this job's information (the executing machine's name and its CPU speed, the job's submission time, and the predicted execution time etc.) in the Local KB. Although the resource manager chooses the machine that looks the "best" at the time the job is submitted, the machine's status such as CPU load may change dramatically during the job's execution. A job sensor is also invoked by the assign function to monitor the job's status on the machine where the job is launched and reports them to the Local KB at a constant interval. The job

sensor is implemented by a script that measures process's CPU time, elapsed time, CPU percentage, and memory usage using the Unix utility *ps*.

Since a VAM may submit multiple jobs on behalf of a user, the job controller keeps track of each job in a hash table. The monitor function checks every job in the table periodically by fetching the information from the Local KB. First, the controller checks whether the job is running. If the job finishes successfully, the controller collects some statistic data about this execution, e.g. the application's input parameters, performance and resource usages, and reports it to the Global Knowledge Base as for historical records. If the job is still running, the controller fetches all the monitored data about the resource's load condition and the job's processes, and then estimates the job's progress based on a general performance model, explained in the next paragraph. If the controller detects any abnormal behavior such as machine failure, job hanging, or intolerable performance, it asks the local resource coordinator to query the global scheduler; and then it tries to allocate another resource on which the job can still finish before the user-specified deadline. The job reallocation also uses the mechanism described above. Then the controller stops the current job and resubmits it to other resources.

For CPU-intensive jobs, how much CPU time is used is largely independent from machine load. Therefore, CPU time is chosen as the criterion to measure how jobs progress, given that the values of CPU cycles and CPU speed are available in the Local KB. The following performance model is used to estimate a job's execution time:

$$Runtime = \frac{CPUtime}{CPUpercentage} \tag{3–2}$$

where CPU percentage represents how much available CPU time currently consumed by the given job, which is updated by the job sensor.

The analyzer periodically calculates the remaining execution time on the current machine using the following equations to predict whether this job's run would meet the

expected deadline, where *Elapsed CPUtime, Elapsed Runtime* are the CPU time and runtime already consumed by the job respectively, and the *Total CPUtime* and *Total Runtime* are the total CPU time estimated by the predict function and the total execution time calculated for this run:

$$\begin{cases} UnfinishedCPUtime = TotalCPUtime - ElapsedCPUtime \\ UnfinishedRuntime = \dfrac{UnfinishedCPUtime}{CPUpercentage} \\ TotalRuntime = ElapsedRuntime + UnfinishedRuntime \end{cases} \tag{3–3}$$

If the analyze function detects that the monitored job cannot finish before the deadline by comparing the predicted total execution time with the deadline set by the user, the job controller asks the resource coordinator to allocate a "better" resource that can satisfy the user's goal or bring about benefits due to rescheduling. So the job controller looks for a resource that satisfies one of the following conditions:

1. $ElapsedRuntime + Overhead + TotalCPUtime \cdot (1 + Load) < Deadline$
2. $Overhead + TotalCPUtime \cdot (1 + Load) < UnfinishedRuntime \cdot k$, where k is rescheduling the benefit ratio.

The first condition means that if the job is rescheduled to the machine, it can still meet the deadline. The second condition means that the re-execution time of the job on the machine plus the rescheduling overhead is shorter than running on the current machine by $(1 - k)$. This condition fits the scenario when, even if rescheduling can not meet the deadline, there are still enough benefits compared with no rescheduling. In order to get a reasonable gain, k is set to 70%.

If the job sensor fails to update the status of the monitored job, the job controller alarms the system that something is wrong, maybe the machine or the network. When the number of consecutive alarms reaches a threshold, the job is considered failed on that resource, and the job controller asks the resource coordinator to allocate another resource and restarts the job on it. Another error condition we want to detect is job

“hanging”. The difficulty of detecting such condition is that the job sensor may still be able to collect job’s status and report it to the Local Knowledge Base, although the job is no longer making any progress. Since we are able to estimate the job execution time with the performance model described above, a time threshold to detect job hanging is set to three times that of predicted execution time.

In order to reduce the overhead caused by querying the large historical data in Global Knowledge Base, historical records are cached in-memory. During a VAM session, the user is likely to submit multiple “similar” jobs, i.e. the same application with similar inputs. Hence with the local records, the autonomic manager can predict these jobs’ resource usages and performance very quickly. Furthermore, it can mark the resources’ quality based on the previous experiences so that it can prefer the “good” resources and avoid the “bad” ones for future jobs.

3.4 Evaluation

The AVAM implementation in In-VIGO is evaluated by answering the following questions:

1. Can it efficiently respond to the dynamic resource information and utilize it to achieve the expected execution performance?
2. Can it prevent and recover from job submission and execution failures?

3.4.1 Experimental Setup

The evaluation experiments were conducted on a testbed of the In-VIGO system. The compute resources consist of two VMware GSX server 2.5-supported single processor virtual machines, hosted on a cluster of 32 Xeon 2.4GHz processors with 1GB memory and 18GB disk storage, and a physical machine with dual 927MHz Pentium III processors, 512MB memory and 23GB disk storage, all running Redhat 7.3. TunProb (Numerical Calculation of the Transmission Probability for One-Dimensional Electron Tunneling), a tool installed in In-VIGO, is used as the workload.

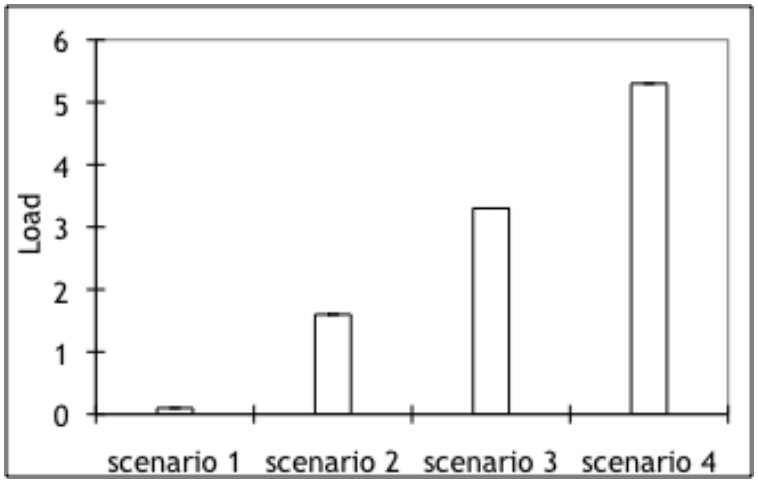

Figure 3-4. Average and standard deviation of load for the unloaded (1), lightly loaded (2), loaded (3), and heavily loaded (4) resource scenarios.

3.4.1.1 Modeling non-dedicated resources in grid environments

In our evaluation, background load on non-dedicated resources in a grid environment is generated by CPU-intensive processes, whose inter-arrival times and runtimes are both modeled as Poisson processes. By changing the ratio of the processes' runtimes to the inter-arrival times, we created four different CPU loads from unloaded to relatively heavy loaded (see Figure 3-4). Light load is generated with processes whose average inter-arrival times (15s) are roughly equal to the runtimes (14s). The medium loaded resource runs processes whose average runtimes (36s) are three times the average inter-arrival time (12s). The heavily loaded resource runs processes whose runtimes (60s) are roughly six times the inter-arrival time (9s). Figure 3-4 shows the averages and standard deviations of the loads for the four scenarios. For the last two scenarios, the standard deviations are relatively large, indicating the generated loads are also highly dynamic.

Grid environments typically have hundreds of non-dedicated resources. The following analysis determines the chances of finding an idle machine in a group of 2, 16, 32 and 256 machines artificially loaded with CPU-intensive jobs. We measure the machines' loads over a period of time in the four scenarios described above,

Table 3-1. The probability of at least one machine being idle or having a light load in four loading scenarios (P1: Probability of at least one machine idle; P2: Probability of at least one machine having a load below 1; P3: Probability of at least one machine having a load below 2).

Arrival rate	Runtime	Avg_load	P1-P2-P3 (2 machines)	P1-P2-P3 (16 machines)	P1-P2-P3 (32 machines)	P1-P2-P3 (256 machines)
0	0	0	1-1-1	1-1-1	1-1-1	1-1-1
1/15s	14s	1.5	0.65-0.96-0.99	1-1-1	1-1-1	1-1-1
1/12s	36s	3.0	0.13-0.48-0.80	0.60-0.97-1	0.82-1-1	1-1-1
1/9s	60s	5.3	0.005-0.05-0.13	0.04-0.23-0.64	0.11-0.36-0.79	0.35-0.97-1

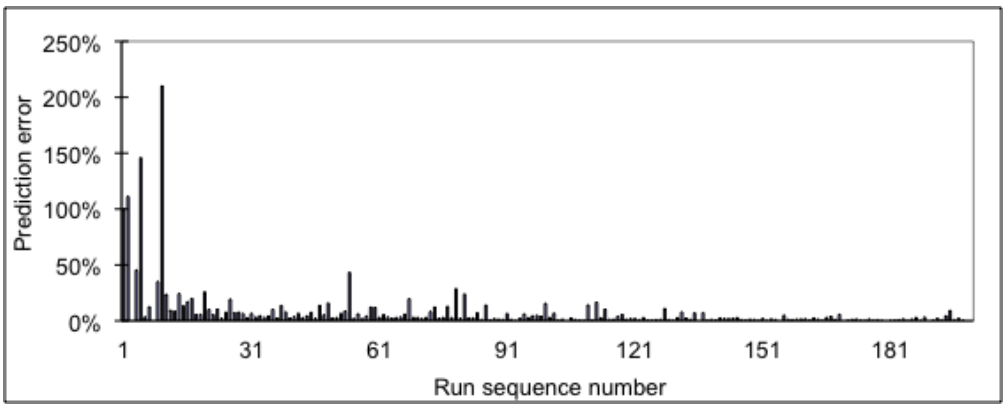

Figure 3-5. The percentage of CPU time prediction error of nearest-neighbor algorithm with a increasing number of TunProb runs.

from unloaded to relatively heavily loaded, and the results reveal that there is a high possibility of at least one machine being idle (see Table 3-1). While this analysis is simplistic, it supports the intuitive assumption that in a real system with large numbers of machines, the probability of the availability of an idle machine at any given time is high, providing an opportunity to avoid performance failures. On this basis, we model the non-dedicated resources in a grid system with two loaded machines and one idle machine to evaluate the mechanisms proposed in this paper.

3.4.1.2 Workloads

TunProb is used as an application benchmark representative of CPU-intensive workloads with short execution times. TunProb requires four input parameters representing the desired one-dimensional tunneling barrier, minimum and maximum

energies and the number of energy steps between them for which the transmission probability must be calculated. TunProb is CPU-intensive, and its resource usage depends on three of the parameters, the minimum and maximum energy and the number of energy steps, which can therefore be used for performance prediction via a nearest-neighbor learning algorithm evaluated in the related work [81]. The algorithm uses distance metric $\sum (x_q - x_i)^2$ (x_q is the query point and x_i is the existing data point), for measuring the distance between two inputs. Figure 3-5 shows its prediction accuracy for TunProb jobs, executed with the minimum energy, maximum energy and number of energy steps randomly selected from 0 to 10, 0 to 1000, and 0 to 1000000 respectively. The nearest-neighbor algorithm learns fast and the percentage of the CPU time prediction error drops below 15% after a hundred runs. The AVAM implementation allows "plugging in" of per-application performance predictors, and the above results indicate that a nearest-neighbor learning algorithm based on the distance metric is suitable for TunProb.

3.4.1.3 Scheduling strategies

Three different strategies were used and compared for scheduling of TunProb jobs on the In-VIGO resources.

- **Round-robin:** The round-robin strategy does not consider the resources' load status and submits the jobs in a round-robin manner to the two loaded machines.
- **Best without rescheduling:** The best candidate without rescheduling strategy uses dynamic resource information to choose the "best" resource, i.e. the more lightly loaded of the two loaded machines, for the job during job submission. It does not however, monitor the job's progress or take any further actions.
- **Best with rescheduling:** The best candidate with rescheduling strategy initially schedules the job on the "best" resource, monitors its execution behavior, predicts its progress and takes appropriate actions, i.e. reschedule the job to the lightly loaded machine if it suffers from a performance fault.

The job QoS requirements managed by Condition 1 and 2 (see Section 3.3.5) are (a) a job runtime deadline no larger than 5 times of its runtime on an unloaded machine,

and (b) a job is rescheduled if it can improve current performance by at least 30%. The alarm threshold/counter for establishing that a job has failed is set at 6. The AVAM is evaluated by comparing the runtime of jobs with the different scheduler strategies in four loaded resource scenarios. We measured the performance improvement due to the AVAM when using the best candidate with rescheduling strategy. Performance improvement due to triggering of Condition 2, measures AVAM's ability to recover from job submission failures.

3.4.2 Experimental Results

3.4.2.1 Performance improvement

In the first set of the experiments, we fix the inputs values of the benchmark application as follows: the energy step is 100000, the minimum energy is 2 and the maximum energy is 200. The average runtime of jobs with these input parameters is about 20s on the unloaded virtual machines and about 40s on the unloaded physical machine. Runtime increases approximately linearly with the CPU load introduced to the machines. The job's deadline is set to 100s. At the job submission time, the two virtual machines with the background load are available resources for the scheduler. After the job submission, the idle physical machine also becomes available as a candidate resource for rescheduling.

For each loading scenario, fifty runs of TunProb were submitted to the testbed continuously. Figure 3-6 shows the average runtime and the standard deviations of executing the benchmark with three different strategies. Figure 3-7 indicates that the percentage of the jobs that meet the deadline (of 100s) for the three strategies. We can observe that when the load on the resources is light, all strategies work well because most of the resources can satisfy the jobs' requirements. When the load becomes heavier and more dynamic, the best candidate strategies perform much better than the Round-robin strategy, while the one with rescheduling substantially outperforms the one without rescheduling. The reasons behind the observations are: first, the best

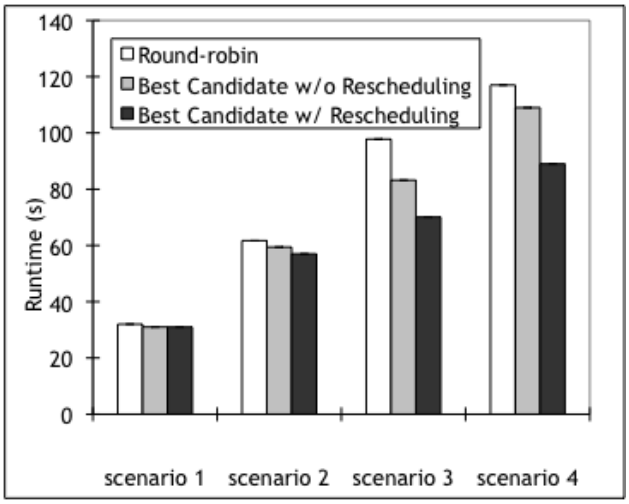

Figure 3-6. The average execution times of TunProb jobs with fixed inputs in the four loading scenarios with three strategies: Round-robin, Best Candidate without Rescheduling and with Rescheduling.

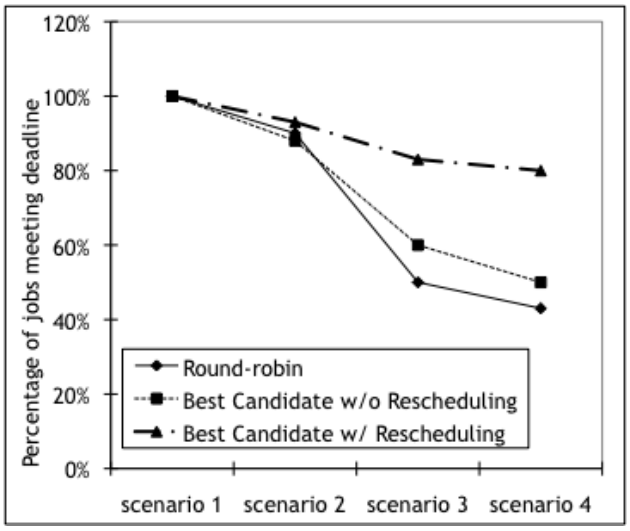

Figure 3-7. The percentage of jobs meeting their deadline (set to 100 seconds) in the four scenarios with three strategies: Round-robin, Best-Candidate without Rescheduling and with Rescheduling.

candidate strategies can always find more appropriate resources for the jobs than the "blind" Round-robin; second, even though the chosen best candidate seems to be the "best" resource for the job at the submission time, it may not be the "best" during the job's execution due to the highly dynamic computing environment, so rescheduling with relatively low overhead can improve the job's execution time and is more likely to meet

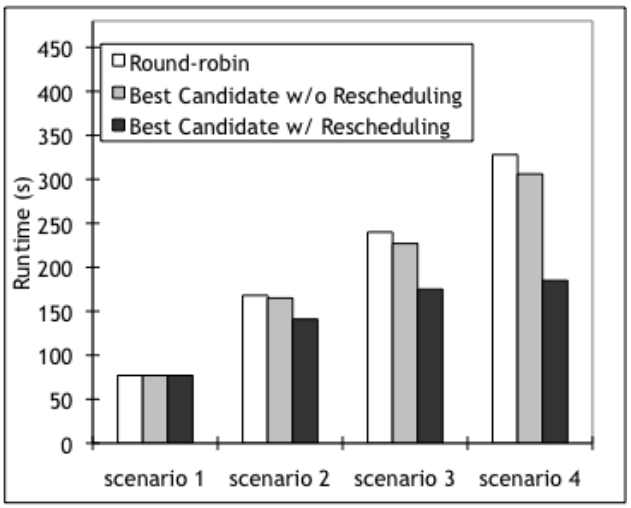

Figure 3-8. The average execution times of TunProb jobs with varying inputs in the four loading scenarios with three strategies: Round-robin, Best Candidate without Rescheduling and with Rescheduling.

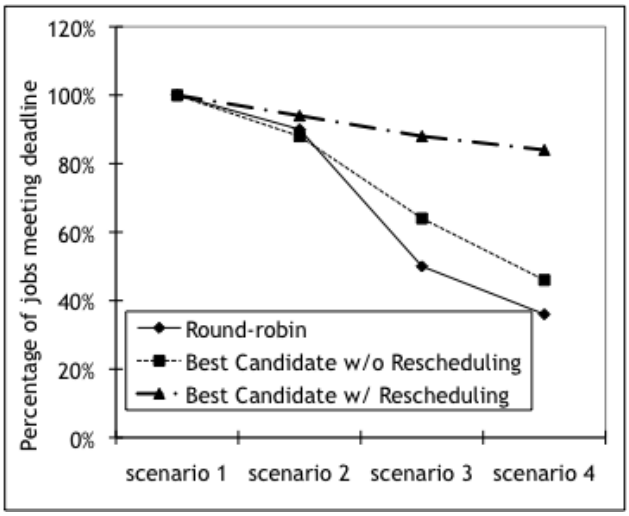

Figure 3-9. The percentage of jobs with varying inputs meeting their deadline (set to four times of execution time on an unloaded virtual machine) in the four loading scenarios with three strategies: Round-robin, Best Candidate without Rescheduling and with Rescheduling.

the expected deadline. In the three loading scenarios with rescheduling, 6%, 12%, and 18% of the jobs had improved performance but did not meet the deadline . The rescheduling of these jobs was triggered by Condition 2, i.e. jobs were rescheduled because they can improve performance by at least 30%, indicating that the AVAM can handle job submission failure using this technique.

Effect of application CPU time prediction errors from the nearest-neighbor learning algorithm on performance is avoided in the above experiments by using fixed input parameters for the benchmark. This allows us to evaluate the rescheduling framework when the application performance modeling only takes into account errors in application runtime prediction for varying load. To take the CPU time prediction errors from the nearest-neighbor learning algorithm into account, the above set of experiments was repeated, while varying the input parameters (energy step, minimum energy and maximum energy) of the application by randomly selecting their values from uniformly distributed ranges of 0 to 1000000, 0 to 10, and 0 to 1000, respectively. The job runtime deadline is set to be four times of the execution time on an unloaded machine. Three virtual machines are used for this experiment, in which two of them are loaded with background processes. Before the jobs' submissions, 100 runs of TunProb with random input parameters in the same above mentioned range are used to warm the predictor. All the jobs are submitted to the two loaded virtual machines, and the third one is used for rescheduling. Figure 3-8 and Figure 3-9 shows that the average runtimes and the percentages of the jobs that meet the deadline for the three strategies. Performance of the three scheduling strategies is similar to the first set of experiments, reflecting the fact that small prediction errors have little effect on the efficiency of rescheduling.

3.4.2.2 Sensitivity to load variances

In the second set of experiments, we want to investigate how efficiently our system responds to the load changes. A TunProb job with the following inputs, energy step of 500000, the minimum energy of 2 and the maximum energy of 200, is submitted to an unloaded machine. The job's average runtime on the unloaded machine is measured to be 80s, so the deadline for job completion is set to 3 minutes. Various amount of background load is introduced into the machine 20 seconds after the job's submission. Figure 3-10 shows the benchmark application execution times with and without rescheduling under different levels of load. The horizontal axis represents

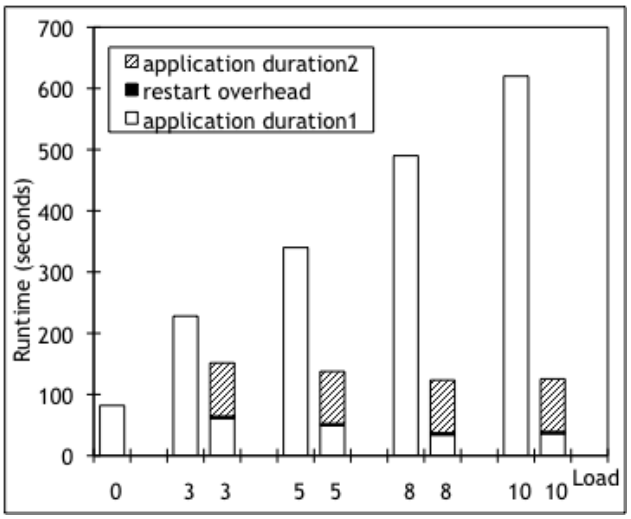

Figure 3-10. Comparison of execution time under changing load with and without rescheduling. Application $Duration1$ is the runtime for the TunProb job on the machine where the job is originally submitted; execution $Duration2$ is the runtime on the machine where the job is rescheduled.

the amount of load we introduced in the machine, while the vertical axis is the job's runtime. The left bar indicates the execution time when the job continues to run on the machine after the load is introduced. The right bar represents the job's execution time managed by the AVAM, which reschedules the job to a better resource because of failure or poor performance. From the figure, we can see that the greater the load introduced the quicker the system is at detecting and reacting to it. The reason for this is that when the load increases to a very high value (i.e. the load is increased to above 5 in our experiments), the job sensor fails to update the job status on time and thus the controller turns the alarm on. After several alarms (we set a threshold value of 6 in the experiments) the AVAM regards the job as failed and reschedules this job to another machine immediately. Otherwise, the manager utilizes the monitored data from the job sensor to estimate whether the job would finish before its deadline. The figure also shows the overhead for rescheduling the job, which averages less than 4 second in our experiments. Compared with the total execution time, this is insignificant for most cases.

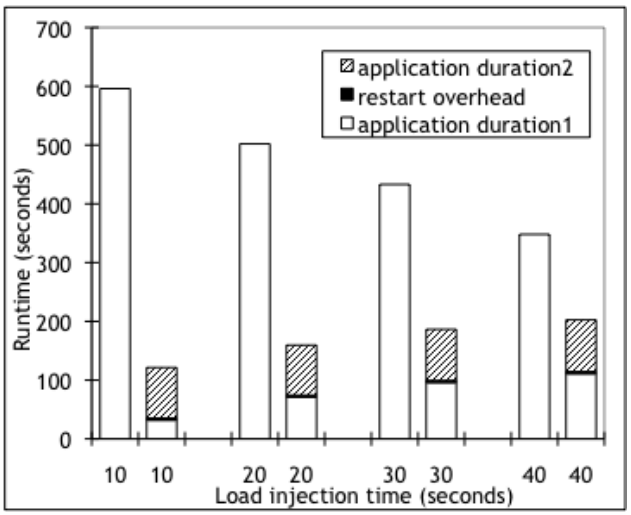

Figure 3-11. Comparison of execution time under different load introduction time with and without rescheduling. Application $Duration1$ is the runtime for the TunProb job on the machine where the job is originally submitted; execution $Duration2$ is the runtime on the machine where the job is rescheduled.

In the third set of experiments, after the job is submitted to an unloaded machine, the same amount of high load is introduced, but at different points during the job's execution. The input parameters for the benchmark are the same as for the second experiment. In Figure 3-11, the horizontal-axis indicates the elapsed time in seconds of the job's execution when the load is introduced, while the vertical axis is the job's runtime. The gain for rescheduling is diminished as the load is introduced later into the job execution. The AVAM recognizes this scenario and avoids unnecessary rescheduling of the application.

3.5 Related Work

The existing work most closely related to our work falls into three categories: resource-usage prediction, resource allocation, and rescheduling.

Topics on resource-usage prediction have been examined by many researchers. Most of these approaches rely on the use of past-execution-time knowledge. Devarakonda et al. [86] uses statistical clustering and state-transition model to characterize process resource usage. The prediction scheme uses the knowledge of the program's resource

usage in its last execution together with its state-transition model to predict the resource usage in its next execution. The approach described in this chapter uses the predictive application modeling developed by [81] because of its efficiency and its consideration of the applications' input parameters. This type of information is available for In-VIGO tools such as the one (TunProb) used as a benchmark. The prediction model employs a local learning algorithm for the prediction of run-specific resource usage on the basis of input parameters supplied to tools. This model works well for our case.

Researchers have proposed a number of systems or approaches for resource discovery and selection in dynamic, heterogeneous computing environments. Some projects [87][88] aimed to support custom-specific systems whose users' specifications can be directly used by a scheduler. In contrast, the system described in this chapter provides applications with a self-learning ability to predict their resource requirements at runtime. The Application-Level Scheduling Project (AppLes) [38] has developed an approach that incorporates static and dynamic resource information, performance predictions into resource scheduling. However, their performance models are application-specific and not easy to reapply to new applications. Our performance prediction approach is based on a generic model that can be applied to any application.

Several projects have implemented rescheduling execution of applications to different resources. These systems either aim to use under-utilized resources, to provide fault resilience, or to reduce the obtrusiveness into workstations. The most related work to ours is described in [89]. Their system continuously monitors applications and evaluates the remaining execution time of the application, and migration decisions are made whenever the applications are not making sufficient progress. The two main differences between their approach and our solution reside in the evaluation model and the rescheduling action. Their evaluation model also depends on application-specific performance models and the overhead caused by migration including reading and

writing checkpoints is much higher than restarting a job in our case. High-overhead migration is not feasible for the relatively short-lived applications considered in this work.

CHAPTER 4
AUTONOMIC RESOURCE MANAGEMENT IN VIRTUALIZED DATA CENTERS

Grid environments, as discussed in Chapter **??**, and data centers face different but yet related challenges in providing performance guarantees to applications. Resource providers in data centers are expected to deliver performance guarantees while optimizing resource utilization, whereas grid middleware cannot expect performance guarantees from individual resources and must relocate applications as needed. This chapter presents our initial work on applying autonomic control in data center environment to optimize application performance and resource usage.

4.1 Problem Description

Data centers have become increasingly important for hosting business-critical applications. A business relationship typically involves data center owners and application providers. A data center provides resources for hosting applications and application providers pay for what they use. It is often desirable for application providers to be able to lease data-center resources under a “pay-as-you-go” model, and for the data-center providers to be able to multiplex shared resources in a way that guarantees the expected performance of applications. To realize this, the data center must provide flexible and manageable execution environments that are customized for each application without compromising its ability to share resources among applications and delivering to them the necessary performance, security and isolation.

4.1.1 Virtualized Data Centers

The traditional data center infrastructure provides very limited options for efficient manageability and improving resource utilization. In traditional data center environment, applications are deployed at different servers to provide necessary security and performance isolation. As more applications are deployed, the number of servers also grows rapidly. Applications hosted in data centers typically have time-varying workload,

with high peak-to-average ratio. Over-provisioning used for meeting peak demand will lead to low resource utilization and high resource wastage.

Virtualization becomes key to providing flexible and manageable execution environments in data centers. It entails the possibility of one physical server hosting multiple independent virtual machines [11][90][91], and the ability of transparently moving applications from one physical server to another through virtual machine migration. Fine-grained virtual machine resource allocation and reallocation are possible in order to meet the performance targets of applications running on virtual machines. The management of virtual machine, e.g. lifecycle management and resource allocation, can be conducted through the interface provided by the virtualization platform. In a virtualized data center, applications are hosted and managed in their dedicated resource containers (implemented as virtual machines) which can be dynamically created and provide strong isolation and security and customizability. Instead of allocating dedicated servers to applications for meeting peak demand, resources are shared among multiple applications to enable efficient resource utilization.

4.1.2 Challenges of Resource Management in Virtualized Data Centers

Applications served by a data center are usually business-critical applications with Quality of Service (QoS) requirements, e.g. e-commerce services. Such applications have time-varying resource demands with typically high peak-to-mean ratios, leading to low resource utilization if over-provisioning is used to meet peak demands. The resource allocation needs to not only guarantee that a virtual machine always has enough resources to meet its applications performance goals, but also prevent over-provisioning in order to reduce cost and allow the concurrent hosting of many applications. Static allocation approaches that consider a fixed set of applications and resources cannot be used because of changing workload mixes, and solutions that only consider runtime behavior of individual applications fail to capture the competition for shared resources by virtualized containers. A key challenge of resource management in virtualized data

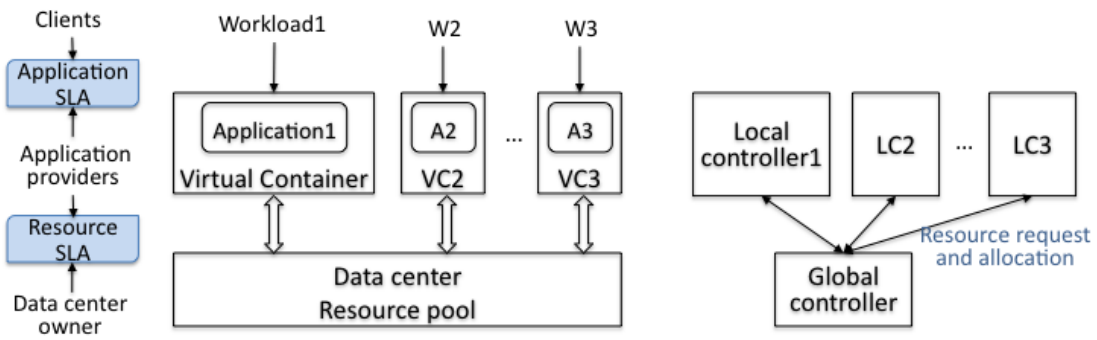

Figure 4-1. A ”Pay-as-you-go” data center model with virtual resource containers to host applications on a shared resource pool, and a two-level controller architecture to allocate physical resources to containers.

centers is the simultaneous on-demand provisioning of shared resources to virtual machines and the management of their capacities to meet service quality targets at the least cost. This work proposes to achieve resource management needed to meet service level agreements (SLAs) by integrating autonomic resource-control functions at two levels - a local controller in each virtual container and a global controller for the data center resource pool.

4.2 Two-Level Resource Control

A two-level autonomic resource management system is developed to enable automatic and adaptive resource provisioning in accordance with Service Level Agreements (SLA) specifying dynamic tradeoffs of service quality and cost. In the system, the resource control functions are integrated at different levels of abstraction: virtual containers and resource pools. A local controller, created per virtual container, is responsible for determining the resources needed by its application and making resource requests accordingly. A global controller manages the virtual containers hosted on the same physical resources. It responds to the local controllers' requests and allocates the shared resources to them in a way that maximizes the total profit (by leasing resources to a large number of containers).

4.2.1 Application Service Level Agreements (SLA) and Resource SLA

A data center, illustrated in Figure 4-1, serves a number of applications. Each delivers a distinct service to its customers using (virtual) resources provided by its dedicated container, which is the virtual machine that hosts the application. The data center allocates the physical resources to each virtual containers based on its application's resource needs.

The application SLA (between an application provider and its customers) states the quality of service providers promised to the clients. To achieve performance isolation and guarantee an application's SLA independently of the loads on other containers, a local resource controller is employed in each virtual container to estimate the resources needed by the application's workload and to make resource requests to the global controller. By doing so, the local controller minimizes leasing costs by avoiding over-provisioning for the application running on the container. Resource SLA (between application providers and the data center owner) specifies both the cost of using resources and the penalty due in case the data center fails to deliver resources needed by the application providers. The underlying assumption is that if the data center does not allocate enough physical resources requested by the local controller resulting in its application's SLA violation, the data center provider will be penalized. The global controller makes allocation decisions among competing requests, trying to avoid violations of Resource SLA.

4.2.2 Benefits of Two-Level Control

This two-level resource control system is preferred over the more obvious centralized approach in which all the control functions are implemented at one centralized location. Since local containers are independent of each other, heterogeneous local controllers' implementations are possible. All of the internal complexities of control functions in virtual containers are compressed by local controllers into straightforward resource requests, which specify the amount of resources needed. The system handles

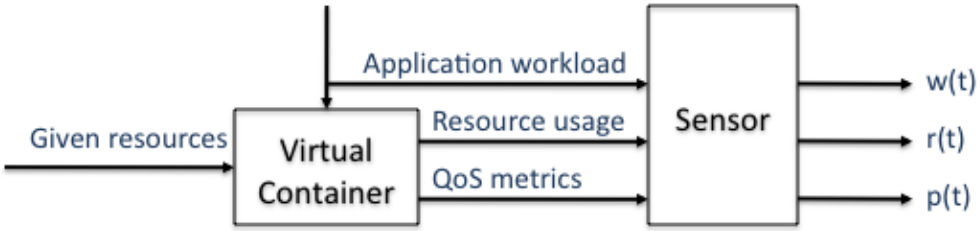

Figure 4-2. Inputs and outputs of a virtual container.

two different types of optimizations independently. The local controller tries to minimize the resource consumed by the virtual container to reduce the resource cost while still satisfying the SLAs of its clients. The global controller seeks to maximize its own profit, which is the revenue received from allocating its resources to virtual containers minus the cost of penalties incurred from resource SLA violations. The following sections explain our approach to the design of the local and global controllers.

4.3 Local Controller

Interaction between the local and global controllers enables a virtual container to augment its resources in response to increased workload, and to reduce its resources when they are no longer needed. The main task of the local controller is to optimize the set of resources needed by an application running in the container. Our approach to the design of such a controller is based on fuzzy logic theory, as discussed next.

4.3.1 Input-Output Model of Virtual Container

To determine the resource needs of an application hosted in a virtual container, the local controller needs to learn the behavior of the virtual container under dynamically changing workloads. Figure 4-2 shows the abstracted inputs and outputs of a virtual container that hosts a running application. The virtual container receives the application workload from its clients, and utilizes the physical resources provided by the data center resource pool to process the workload. The achieved QoS of the application depends on the amount of allocated resources and the incoming workload. The information about the application's workload, its achieved performance and its virtual container's resource

consumption is monitored by the system sensors as Figure 4-2 illustrates. Depending on what information is available from the system, two approaches are proposed for estimating resource needs: (1) fuzzy modeling to characterize the relationship between workload and resource use and (2) fuzzy prediction to determine a mapping from current resource observations to future resource needs.

4.3.2 Basics of Fuzzy Logic

Fuzzy logic [92] is a tool to deal with uncertain, imprecise, or qualitative decision-making problems. Unlike in Boolean logic, where an element x either belongs or does not belong to a set A, in fuzzy logic the membership of x in A has a degree value in a continuous interval between 0 and 1. Fuzzy sets are defined by membership functions that map set elements into the interval [0 1].

One of the most important applications of fuzzy logic is the design of fuzzy rule-based systems. These systems use IF-THEN rules (also called fuzzy rules) whose antecedents and consequents use fuzzy-logic statements to represent the knowledge or control strategies of the system. The collection of fuzzy rules is called a rule base. There are many sources for constructing fuzzy rules, for example, from expert experience or based on an operator's control actions. The approach taken for the design of our system is to learn fuzzy rules using online monitoring information, making it a so-called self-learning fuzzy system.

The process of applying fuzzy rules on the system is called the fuzzy inference (FIS) mechanism. Since fuzzy rules use fuzzy values to describe the system variables, two functions are necessary for translating between numeric values and fuzzy values. The process of translating input values into one or more fuzzy sets is called fuzzification. Defuzzification is the inverse transformation which derives a single numeric value that best represents the inferred fuzzy values of the output variable.

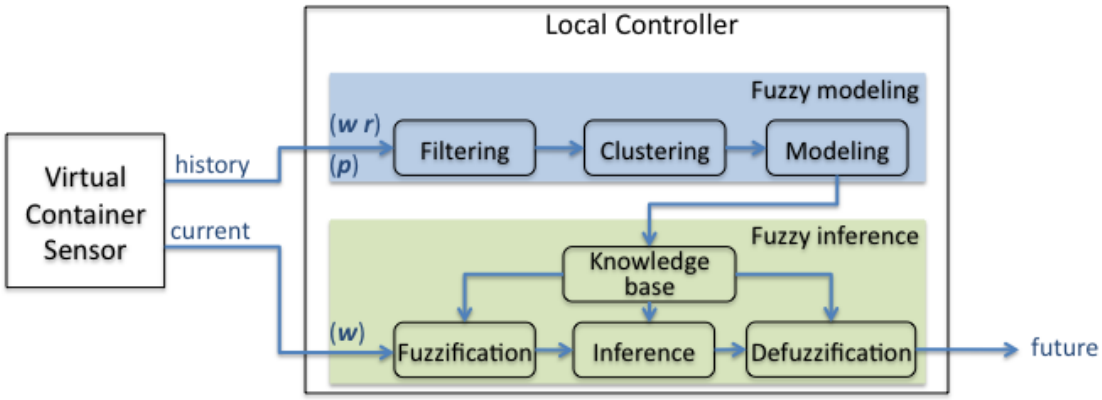

Figure 4-3. Fuzzy modeling and inference functions in a local resource controller.

4.3.3 Fuzzy Modeling

The first approach uses fuzzy logic systems to model the behavior of a virtual container by automatically learning the relationship between workload and the corresponding resource consumption when the desired QoS is achieved. It requires the system to periodically monitor the application workload and their resource usage, which are then used as input-output data pair for generating fuzzy rules. Figure 4-3 illustrates the key functions for fuzzy modeling in the local controller. The data monitored by the sensors are first processed by the filtering and clustering functions. The modeling function constructs fuzzy IF-THEN rules using the produced data clusters and keeps them in the knowledge base. The cluster centers and ranges are used to determine the fuzzy model's parameters. Finally, the fuzzy inference functions process the fuzzy rules kept in the knowledge base to determine the resource needs from the currently monitored workload. The rest of this section explains these functions in detail.

Data Monitoring and Filtering: Monitoring sensors periodically measure the application workload $w(t)$, its performance $p(t)$, and the resource usage $r(t)$ of a virtual container. For a typical data center application, its workload can usually be described by the rate and mixture of its client's requests. For instance, a Web server's workload can be characterized by the HTTP request rate as well as the ratio of static Web-content requests to dynamic ones. The performance metrics are often directly taken from the

application SLA, e.g. the throughput (number of completed transactions per second) and/or average service response time.

The metrics for resource utilization are associated with the different types of consumed physical resources, including CPU percentage, memory size, disk storage, disk I/O rate and network bandwidth. However, an application's virtual resource usage (the values collected inside of the virtual container) does not necessarily represent its physical resource consumption. For example, an application's network I/O consumes not only the physical network bandwidth, but also the physical CPU cycles. In the proposed approach, an application's resource usage is obtained by directly monitoring the physical resource consumption of its virtual container. This is sensible because in the envisioned data center a virtual container is dedicated to an application.

A sequence of input-output data pairs $(w(t), r(t))$ are produced by the sensors at constant time intervals (20 seconds in our experiments), and then filtered based on the corresponding performance measurements $p(t)$. The filtering policy is that a data pair measured at time t is kept or filtered out depending on whether the performance measured at the same time satisfies the application SLA or not, respectively. Performance is satisfactory, only if the resources allocated to the virtual container at time t is sufficient for the given SLA. In this case, the monitored resource utilization represents the actual resource needs, and thus the data pair can be used for modeling. On the contrary, an SLA violation indicates that the allocated resources are not enough to achieve the SLA target. In this case, the resource consumption is capped by the allocated capacity so that the monitored values are less than the desired resource demands and cannot be used in fuzzy modeling.

Data Clustering and Fuzzy Rule Construction: The collected paired data are first clustered to produce a concise representation of the system's behavior and then the data clusters are used to build fuzzy models. Several classic clustering algorithms

can be used, e.g. hierarchical and k-means clustering. In the proposed local controller design, subtractive clustering [93] is chosen for its speed and robustness.

This clustering method assumes that each data point is a potential cluster center and chooses the data center based on the density of surrounding data points. The algorithm selects the data point with the highest density value to be the first cluster center and then removes all data points in the vicinity of the first cluster center in order to determine the next data cluster and its center location. This process continues until all the data are within radius of a cluster center. The variable *radius* represents a cluster center's range of influence in each of the data dimensions, assuming the data fall within a unit hyperbox. Setting small radius values generally result in finding a large number of small clusters. This value is set to 0.5 in the local controller's implementation.

Since each produced two-dimensional cluster exemplifies a characteristic of system input-output behavior it can be used as the basis of a fuzzy rule that describes the relationship between a system's input and output. If n data clusters are formed, n rules can be produced in which the ith rule is expressed as:

IF input w is in cluster i, THEN output u is in cluster i.

Each cluster specifies a fuzzy set with its membership functions determined by the cluster center and range. Using the Gaussian membership function, $\mu_i(x) = e^{-\frac{(x-c_i)^2}{2\sigma_i^2}}$, the center of the membership function c_i equals the center of cluster i and the weight of membership function σ_i equals the radius of that cluster.

The model described by the above fuzzy rule is called zero-order Sugeno-type fuzzy model [17]. The modeling accuracy can be improved significantly by using the first-order Sugeno model, in which the output of each rule is a linear function of the input variables. The rules are rewritten as follows,

IF input w is in cluster i, THEN output $u = aw + b$ (for rule i)

The parameters a and b in the linear equations are estimated by the least-squares method.

Fuzzy Inference: Once the fuzzy model relating workload to resource demand is learned from the selected workload and resource usage measurements, it can be used in a rule-based fuzzy inference module which, given the application's workload, produces the estimated application's resource demand for the virtual container.

The fuzzy inference module consists of four basic functions shown in Figure 4-3. The knowledge base includes a database which contains the membership functions of the fuzzy sets and a rule base where the fuzzy rules are specified. In the fuzzification function, the input $w(t)$ measured from the sensor is mapped to fuzzy values using the membership functions. A decision-making unit, called the fuzzy inference engine, infers from fuzzy inputs to resulting fuzzy outputs according to the rules stored in the knowledge base. The defuzzification function aggregates the outputs and converts them to a single output value. The final output is the weighted average of all outputs with the weight of ith rule being the membership value of the input in cluster i.

In summary, using fuzzy modeling and fuzzy inference shown in Figure 4-3, the local controller estimates the resource needs for the current workload measured by the sensor, and sends requests to the global controller to either ask for more resources if the current allocation is not sufficient to satisfy the application SLA or to withdraw resources when no longer needed.

Adaptive Modeling: The discussion so far has only considered offline model learning from collected data. As workload or system condition changes, the model describing the system's behavior needs to capture the changes accordingly. The adaptive modeling is employed by the local controller in which the model is repeatedly updated based on online monitored information.

The clustering function takes new data into consideration as soon as they arrive (after the filtering) and keeps updating, so that up-to-date clusters are always provided for the modeling. Whenever the data clusters are updated, the parameters of the membership functions are changed accordingly in the database. If a new cluster is

added, a corresponding rule is then added into the rule base; and similarly, if a cluster no longer exists, the rule associated with it is removed from the rule base.

In the case when the allocated resources are insufficient for the workload, the monitored data becomes disqualified and is filtered out because of the performance degradation. The shortage of qualified data will hurt the model's learning speed and quality. To avoid this situation, whenever the filter function detects that the percentage of qualified data is less than 50% during a time window *T* (is set to 5 minutes in the prototype), the controller asks for an additional predefined percentage (10% is used in the prototype) of current resource allocation from the global controller to improve the application's performance back to the desired level.

4.3.4 Fuzzy Prediction

The fuzzy-modeling based approach described above automatically builds a mapping from the application workload to the corresponding resource needs for the desired QoS. This approach is applicable only when the application workload can be characterized and monitored. However, data centers typically host a variety of applications, therefore, it is unclear what should be a set of standard metrics for application performance due to the diversity of applications coexisting in a data center. In some cases it is hard to describe an application workload using a few metrics like request rate. The second proposed approach – fuzzy-prediction – only requires information about the resource usage (e.g., CPU utilization), which is easy to obtain by monitoring system-level metrics. The basic idea is to determine future resource needs on the basis of observations of past resource usage.

Fuzzy Rule Construction: The fuzzy prediction system, illustrated in Figure 4-4, has some components that are similar to those used in the fuzzy modeling approach. The fuzzy rules are generated from monitored data and stored in the rule base. The fuzzy inference system processes the learned fuzzy rules to forecast future resource use based on the current system observations. Let $r(t)(t = 1, 2, 3, \ldots,)$ be resource

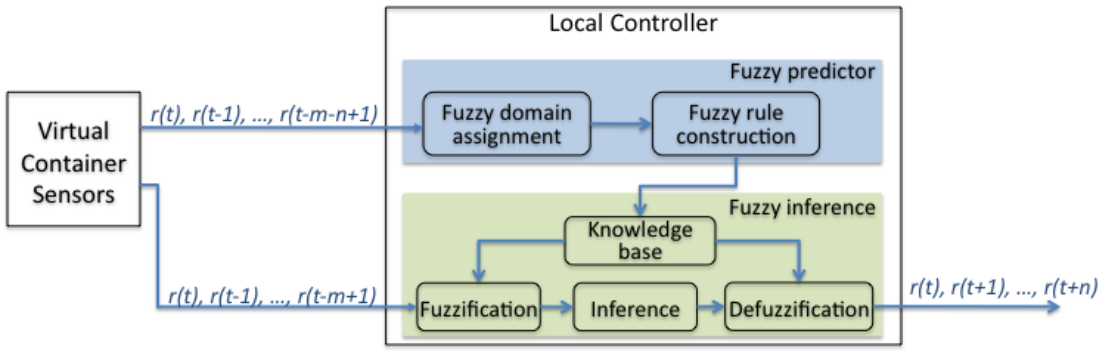

Figure 4-4. Fuzzy prediction functions in a local controller.

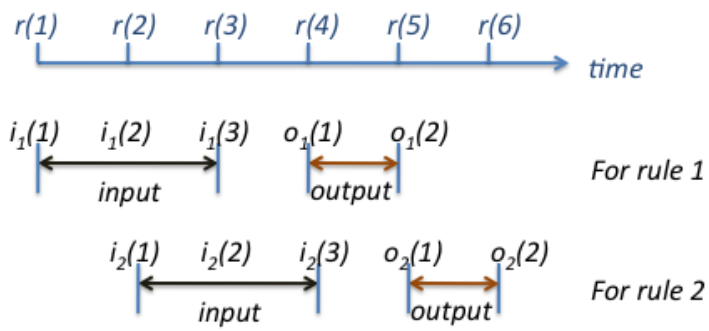

Figure 4-5. Three-input two-output data pair examples.

usage measurement at sampling time. The problem can be formulated as: at time t, given the latest m measurements $r(t), r(t-1), \cdots, r(t-m+1)$, determine the resource use at future time $r(t+1), r(t+2), \ldots, r(t+n)$ (m and n are the number of inputs and outputs for a fuzzy rule, respectively).

A fuzzy rule is generated from an input-output data pair, whose components are subsequences of the successive resource usage measurements, the input subsequence preceding the output subsequences. Figure 4-5 shows an example of three-input two-output ($m = 3$ and $n = 2$ in this case) fuzzy rules. To translate input-output pairs into fuzzy rules, the first step is to divide the input and output spaces into fuzzy domains. Assuming that the input and output ranges are normalized to [0, 1], each space is divided into $2N + 1$ domains, denoted by $R_1, R_2, \ldots, R_{2N+1}$, each assigned a fuzzy membership function. Figure 4-6 gives an example of membership functions where the

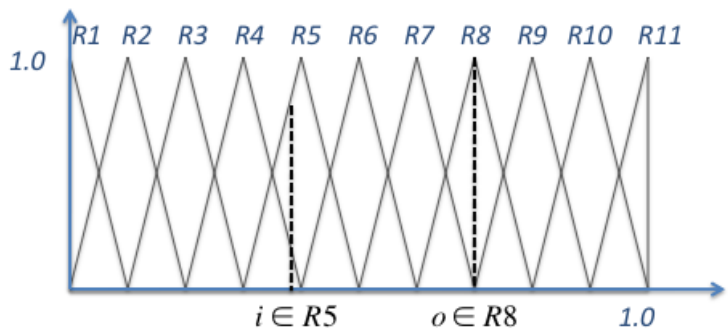

Figure 4-6. An example of dividing the input/output space into 11 fuzzy domains and the corresponding membership functions.

input/output space is divided into 11 domains ($N = 5$ is used in our prototype) with triangular membership functions.

The next step is to assign a given data point to the fuzzy domain with the highest membership degree using the membership functions described above. For example, for an input-output data pair i and o, i is assigned to domain R_5 and o is considered to be R_8 in Figure 4-6. Finally, a fuzzy rule is constructed from a pair of input-output data as follows,

IF i_1 is R_{i1} and i_2 is R_{i2}, . . . , and i_m is R_{im} , THEN o_1 is R_{o1}, . . . , o_n is R_{on}. (for rule i)

Therefore, every sequence of $m+n$ consecutive resource usage measurements can be used to generate a fuzzy rule which maps the input space ($i_1, i_2, \ldots, i_m$) representing the recent system state to the output space ($o_1, o_2, \ldots, o_n$) representing the future state.

Fuzzy Rule Update: A fuzzy rule is generated at every sampling time and each rule is represented as a point in the $(m + n)$-dimensional rule space. If every rule is stored in the rule base the memory requirements will be excessive, and it is probable that there would be conflicting rules which have the same IF part but a different THEN part. The first problem is solved by partitioning input-output spaces into a finite number of domains as described above so that at most one rule (i.e., a point) is stored in the rule base for each domain. The number of rules increases as new input-output data are collected, but

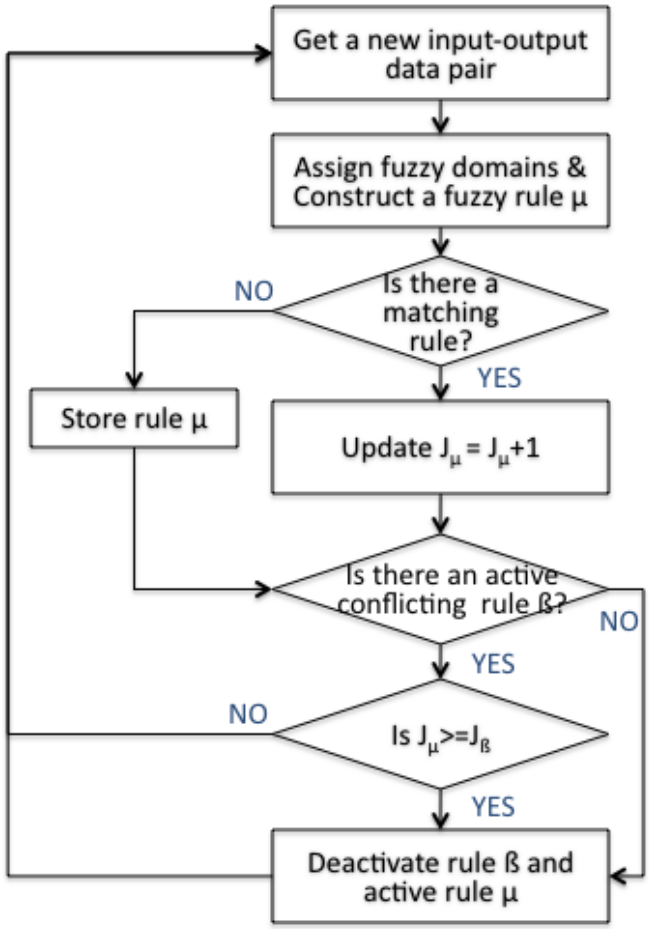

Figure 4-7. The fuzzy-rule updating procedure.

it never exceeds the maximum number of domains partitioned in the $(m+n)$-dimensional rule space.

To overcome the second problem, when updating the rule base a reliability index is computed for each rule as J_i = the number of occurrences of rule i. Whenever a rule is generated, the system scans all the rules stored in the rule base. If there is a matching rule (i.e., a rule in the same domain), the value of J is increased by 1. Otherwise, the new rule is added to the rule base and J is initialized to 1. Figure 4-7 illustrates the procedure for updating rules. If there exist conflicting rules, the one takes effect is determined by the value of the reliability index. The rule with the highest reliability index is activated, indicating that the active fuzzy rule appears more frequently than other

conflicting rules. If two conflicting rules have the same value of reliability index, the one that appeared most recently is activated.

Fuzzy Inference: Given the latest resource usage measurements as inputs, as Figure 4-4 shows, the fuzzy inference engine processes the active fuzzy rules kept in the rule base to determine the outputs which consist of the future resource usage. Initially, there is no rule in the rule base. After the first $m + n$ measurements are obtained, the first rule is generated and stored in the rule base. Afterwards, at each sampling point, a new rule is constructed and the rule base is updated following the updating procedure shown in Figure 4-7. This updating procedure makes the proposed fuzzy prediction capable of self-learning the resource usage behavior of the managed virtual container.

Compared with the fuzzy-modeling approach, both methods essentially learn from historical data to build fuzzy rules and can adaptively update the rules when new data are available so that it can reflect system changes very quickly. No prior knowledge or mathematical models of the system are required and they both are a one-pass build-up procedure that does not need iterative time-consuming training. The difference between the two approaches is that the fuzzy modeling approach maps workload to resource consumption, while the fuzzy prediction maps the observations of recent resource usage to the future resource needs.

4.4 Global Controller

Each individual local controller tries to minimize the resource cost by only requesting the resources necessary for meeting the application SLA. The global controller receives requests for physical resources from local controllers and allocates the resources among them as required. It seeks to make allocations that maximize the data center's profit, which is, based on the proposed profit model in a pay-as-you-go data center, the revenue received by allocating the resources minus the penalties due to resource SLA violations.

The local controllers periodically send resource requests to the global controller which makes allocation decision based on the received requests and currently available resources in the data center. To simplify the problem, we consider a single resource type and a single allocation period. Suppose that K virtual containers are concurrently active in the data center. Let req_k denote the resources requested from virtual container k, and alc_k be the amount of resources granted to it by the global controller. The data center receives revenue of rev_k for every allocated resource unit over an allocation period. But if the global controller cannot satisfy the request req_k, the data center pays a penalty of pnl_k per unit of unmet resource demand, according to the resource SLA. Each resource allocation decision made by the global controller is expressed as a resource allocation vector $(alc_1, alc_2, \ldots, alc_K)$, and the total profit obtained by the data center for a time period is,

$$profit(alc_1, alc_2, ..., alc_K) = \sum_{k=1}^{K} [rev_k alc_k - pnl_k(req_k - alc_k)] \quad (4\text{–}1)$$
$$s.t. 0 \leq alc_k \leq req_k,$$
$$\sum_{k=1}^{K} alc_k \leq A$$

where A is the total available resource capacity in the data center. The profit equation can be rewritten as follows,

$$profit(alc_1, alc_2, ..., alc_K) = \sum_{k=1}^{K} (rev_k + pnl_k) alc_k - \sum_{k=1}^{K} pnl_k req_k \quad (4\text{–}2)$$

$(rev_k + pnl_k)$ is considered as the profit rate for virtual container k. Assuming that the global controller can allocate any resource fraction to the virtual containers, a greedy algorithm that allocates resources in the order of decreasing profit rates is an optimal allocation (this is similar to the case of a fractional knapsack problem [94]).

To optimize profit over multiple time periods, the allocation decision has to be repeated. Equation 4–3 defines a cumulative profit which is the discounted sum under a discounting factor γ over a time horizon T. The factor models the fact that future profit is worth less than current profit because of the uncertainty in the future. For T allocation periods,

$$\max \sum_{t=1}^{T} \gamma^t profit_t = \max \sum_{t=1}^{T} \sum_{k=1}^{k} \gamma^t [rev_k alc_{tk} - pnl_k(req_k - alc_{tk}) \tag{4–3}$$

Based on the above profit model, a greedy strategy that maximizes the total profit for every period is still optimal because the allocation decision for current period does not affect the future periods.

4.5 Experimental Evaluation

This section summarizes the experimental evaluation of the proposed two-level control system for dynamic resource allocation in a data center environment with time-varying workloads. Section 4.5.2 and 4.5.3 discuss the experiments that evaluate the ability of the local controller to track the resource needs of changing workloads. Section 4.5.4 considers the maximal profit approach (Equation 4–1) discussed in Section when the global controller must allocate limited resources among several competing virtual containers.

4.5.1 Experimental Setup

Data Center Testbed: The testbed is deployed on a 16-CPU IBM x336 based cluster that provides virtual containers for applications. VMware ESX Server 3.0.1 is installed in each cluster node equipped with dual hyperthreaded Intel Xeon 3.2GHz CPUs and 4GB memory. Virtual machines are created on the servers and used as virtual containers to host applications. The workload-generating clients are placed on VMware-Server-1.0.0-based virtual machines, hosted on another cluster of 32 dual-2.4GHz hyperthreaded Intel Xeon nodes. Web-based workloads are generated by the clients and issued to the applications across a Gigabit Ethernet network.

Application and Workloads: The Java Pet Store [95] was chosen to represent a typical e-business application. It implements an online store that allows users to browse and make orders, and managers to manage orders, suppliers and inventory. Synthetic HTTP workloads that mimic the key aspects of real-world workloads are created with various client sessions issued by httperf [96]. Each individual session consists of a sequence of requests generated by repeating and mixing the following customer actions: go to the storefront, sign in, browse products, add some products to shopping cart, and checkout. Two key parameters are adjusted to vary a session's workload on the application: the user think-time (the time between two consecutive requests) can be changed to generate different request rates; the ratio of dynamic requests (e.g., sign in, check out and search product) to static requests (e.g., browse static Web pages and view images) can be varied in order to change the workload characteristics. A Perl program was developed to create different workloads and drive httperf to issue the requests.

Traces collected from '98 World Cup sites are also used in the experiments to represent real-world workloads. The logs provided by an Internet repository [97] consist of about 1.3 million requests made to the '98 World Cup Web site between April 30, 1998 and July 26, 1998. A real-time log replayer's [98] is used to generate workloads according to the trace.

Global/Local Controller Prototype: The virtual containers are monitored and controlled through the Web-service-based management interface provided by VMware ESX Server. It allows the allocation of a server's physical resources among its hosted virtual machines (e.g. setting the minimum, maximum and proportional resource shares of a virtual machine), and also provides the real-time monitoring of a virtual machine's resource utilization.

The proposed two-level controllers are implemented in Java, running along with the virtual containers. Every virtual machine has a local controller to manage the virtual

container it provides, and every ESX server node has a global controller to manage the shared resources for the virtual containers hosted on it. The sensors, also developed in Java, monitor the workload (request rate and mixture), the application throughput (reply rate), and the resource consumption (CPU usage). The monitoring period is set to 20 seconds. Because the concerned workloads are mostly CPU intensive, the experiments focus on the utilization and allocation of CPU resources.

4.5.2 Fuzzy-Modeling Approach

The following explains the experiments for validating whether the local controller can accurately estimate resource needs using the fuzzy-modeling and fuzzy-prediction approaches under dynamically changing workloads.

4.5.2.1 Static Web requests

In the first experiment, the workload generator issues a new session to the Pet Store every 10 seconds, up to a total of 15 sessions. These sessions only contain requests for static Web content with a user think-time ranging from 0.1 to 1 second, and each session lasts around 1 minute. After a group of 15 sessions are completed, another group is generated similarly but with a decreasing average think-time (and hence an increasing request rate). This setup emulates the presence of burst in real-world workloads. The entire experiment lasts for 4000 seconds.

Because the workloads used in this experiment consist of only static Web-content requests, the CPU usage is highly correlated with the request rate, which is then used as the only metric to characterize the workload. In this case, the input and output to fuzzy modeling are the request rate and CPU usage measurements. The first 50 pairs of data points collected from the sensors are used to initialize the learning of the fuzzy model. Afterwards, the model is continuously updated every 200 seconds (during which 10 new data points become available from the sensors). Figure 4-8 illustrates the model learned at the beginning and the end of the experiment, which shows an approximate

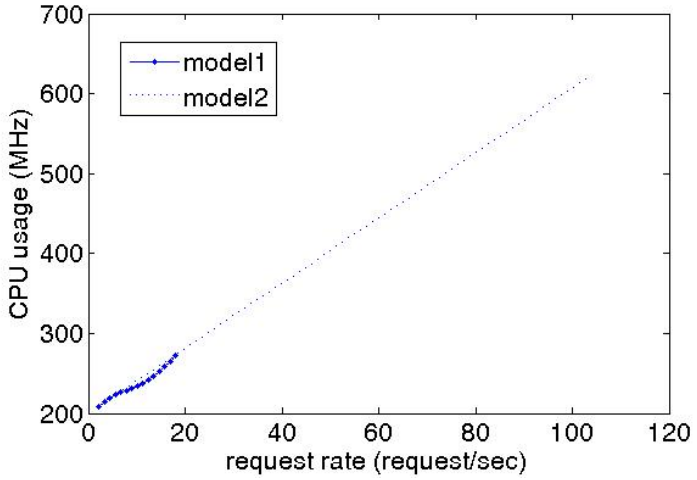

Figure 4-8. Fuzzy model learned from the workload with static Web requests at the beginning (model1) and the end (model2) of the experiment.

linear relationship between the request rate and CPU usage in the range of 0 to 100 requests/second.

The local controller continuously estimates the CPU demand based on the current workload and the latest learned fuzzy model, and dynamically adjusts the CPU requests to the global controller. Because the available resources are sufficient for the only virtual container used in this experiment, the global controller always allocates the exact amount of CPU requested by the local controller. To prove the accuracy of the fuzzy modeling, the same experiment is repeated on the virtual container which is statically allocated with a large amount of CPU (3.2GHz) in order to obtain the ideal throughput for the same workload.

The throughput from these two experiment runs are compared in Figure 4-9, indicating that the actual throughput obtained by dynamic allocating resources using the fuzzy-modeling approach is almost identical to the ideal throughput. Compared to the static allocation of CPU with the peak value (overprovision based on the highest load), the dynamic resource allocation approach saves about 55% of CPU cycles otherwise needed for this experiment. This confirms that the online fuzzy modeling can accurately

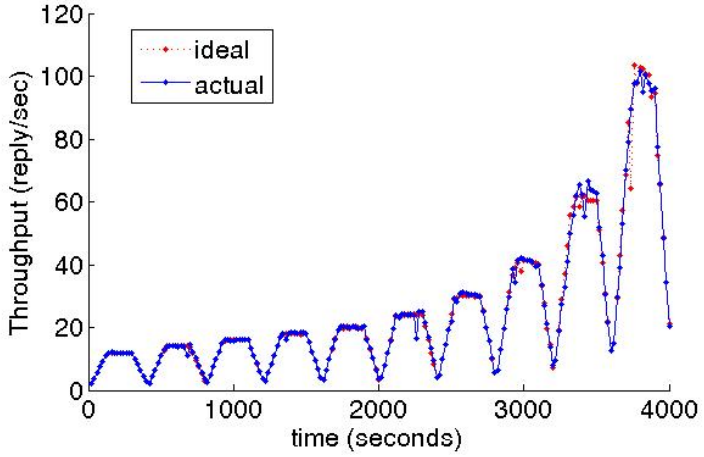

Figure 4-9. Comparison of the throughput achieved by using local controller and the ideal throughput for the workload with static Web requests.

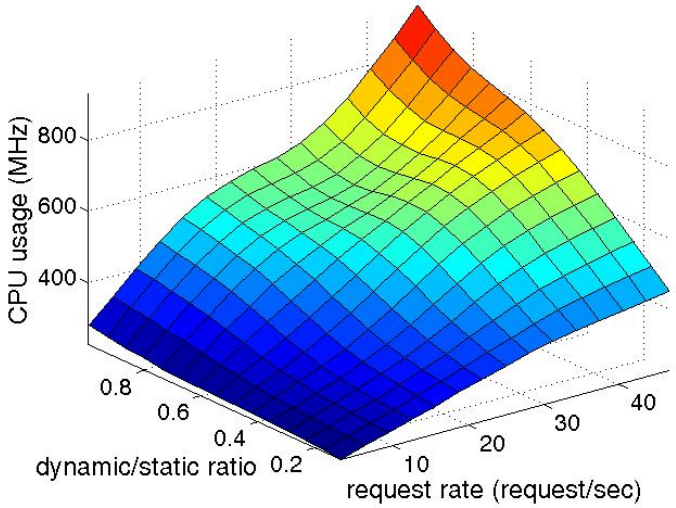

Figure 4-10. The surface of the 3D fuzzy model learned from the workload with dynamic Web requests.

learn the relationship between the workload and resource demand, and effectively guide the resource allocation for the virtual container.

4.5.2.2 Dynamic Web requests

In the second experiment, the workloads are generated similarly to the previous one, except that dynamic Web requests are also considered. Every group of sessions differs not only in the request rate but also the proportion of dynamic requests in the

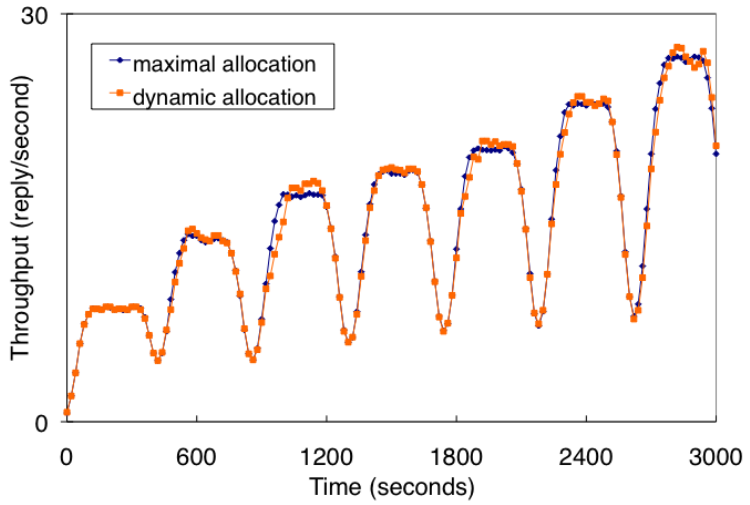

Figure 4-11. Comparison of the throughput achieved by dynamic allcation using fuzzy modeling and the ideal throughput with maximal allocation for the workload with dynamic Web requests.

workload: the ratio of dynamic to static requests grows from 0 to 1 across the groups. Servicing dynamic Web content requires processing by the application server and database, and thus typically consumes more resources than servicing static Web content. If the fuzzy modeling still uses only request rate as the input, the resulting model cannot effectively represent the actual relationship between workload and resource demand. The experiment results (observed but not shown here) also confirm that the throughput achieved by using such a model is much worse than the ideal throughput for the workload.

In contrast, using both the request rate and dynamic/static request ratio to characterize the workload, a 3D fuzzy model can be constructed to describe the relationship between workload and resource demand more accurately. Figure 4-10 shows the surface of the model learned at the end of the experiment. One of the advantages of fuzzy modeling demonstrated by the above experiments is that fuzzy models can effectively learn simple as well as non-linear and complex relationships between inputs and outputs.

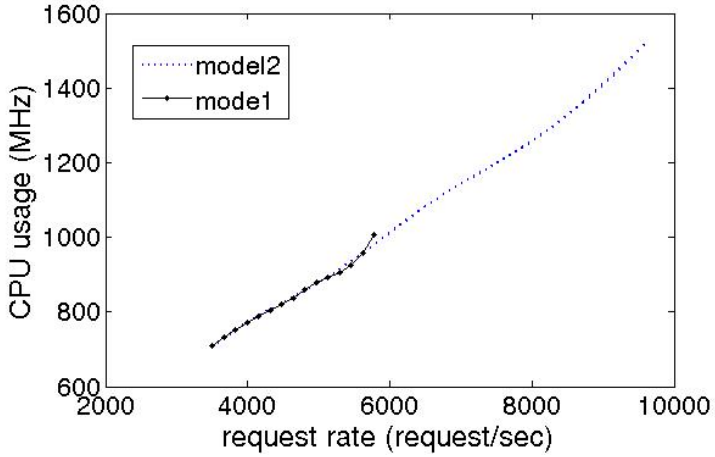

Figure 4-12. The surface of the 3D fuzzy model learned from the workload with dynamic Web requests.

Figure 4-11 compares the application's throughput to the ideal throughput obtainable for the workload. The graph shows that the throughput achieved is again very close to its ideal level (the difference is under 6%). It is also noticeable that when the workload is high the difference becomes relatively larger. This is because of the delay between the change of workload and resource allocation, which is largely due to the granularity of the online monitoring and control. When the workload is heavy, this delay causes the application's throughput to fluctuate a little around the ideal one. However, the overall error is still very low. About 33% of CPU cycles are saved by this dynamic allocation, compared to a fixed allocation where overprovision is based on the highest load.

4.5.2.3 Trace-based workload

In the third experiment, the '98 World Cup Web site trace collected on May 31 from 5am to 5pm (local time in Paris) is used to generate workload, and it is played at 12X speedup to enhance its intensity. All the documents requested by the trace are created by the log replayer tool based on the sizes recorded in the trace. Because only static Web pages are requested in the trace replaying, the workload is characterized by the request rate.

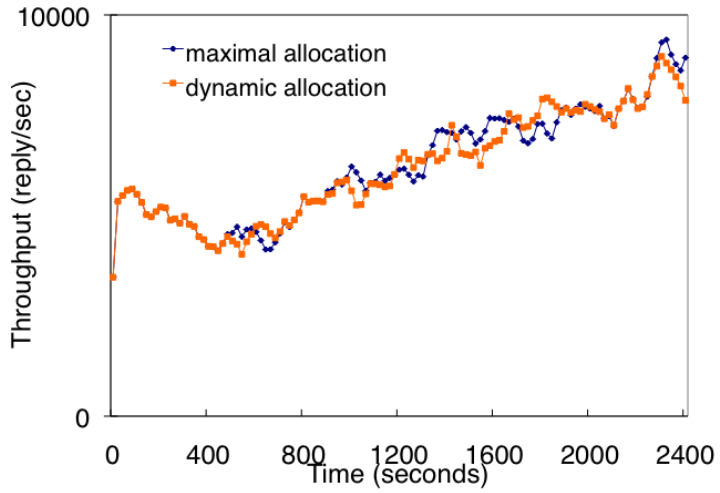

Figure 4-13. Comparison of the throughput achieved by using local controller and the ideal throughput for the trace-based workload.

During the experiment, the first 30 measurements of workload and CPU consumption are used to initialize the fuzzy model. After that, the model is updated every 200 seconds. Figure 4-13 shows that the application's throughput achieved by using the local controller is close to the ideal throughput obtainable for the workload (the difference is within 5%), indicating the effectiveness of the fuzzy modeling approach under real-world workloads. The dynamic allocation uses less than 75% of the CPU cycles used by a static approach that allocates maximum CPU fraction based on the highest workload.

4.5.3 Fuzzy-Prediction Approach

Similar to the previous experiment, three days of web traces from '98 World Cup Web site are chosen to generate workload. The traces are played at 24X speedup to reduce experiment duration. During the experiment, only the CPU utilization of the virtual container is measured and fed to local controller every 20 seconds. The first 50 measurements are used to initialize the fuzzy rules. After that, the rule base is updated whenever the new CPU usage measurement is available (every 20 seconds). Every one minute, the local controller estimates the CPU needs for the next minute based on the fuzzy rules learned from the pervious observations and then sends requests to

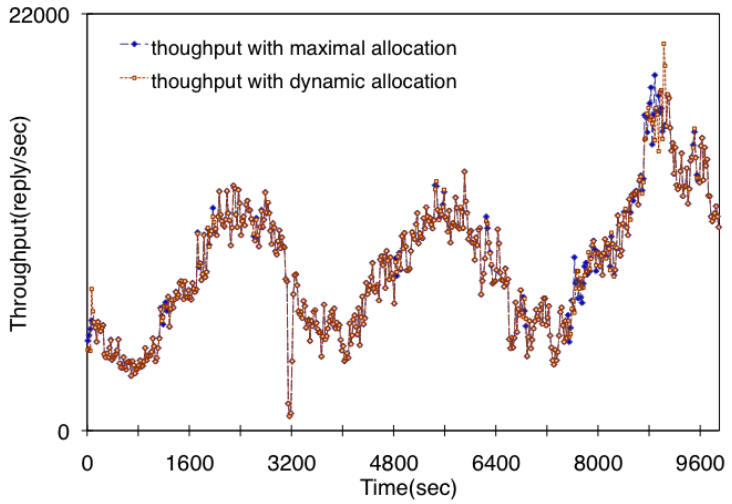

Figure 4-14. Comparison of the throughput achieved by dynamic allocation using fuzzy prediction approach and the ideal throughput with maximal allocation.

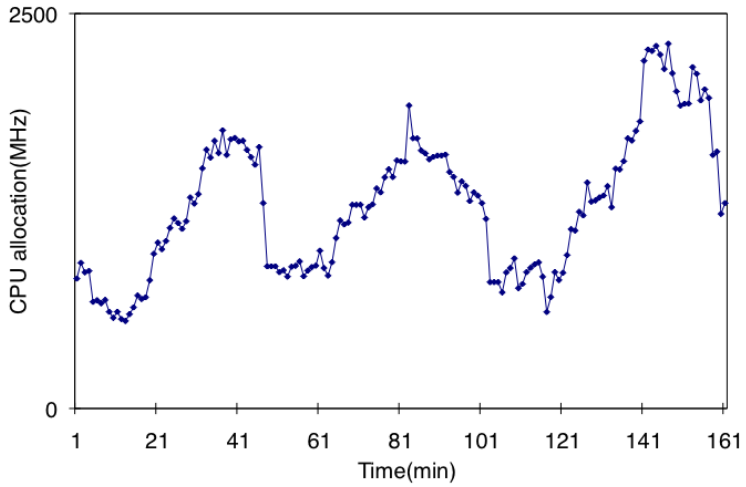

Figure 4-15. The CPU allocated to the virtual container through the interaction of local and global controller.

the global controller. Then the global controller adjusts the CPU allocation according to the requests from the local controller every minute. Figure 4-14 shows the resulting throughput for the trace-workload by using fuzzy prediction and the throughput achieved by using maximal allocation (5.4GHz). The results are very close and the differences between them are less than 1% on average. Figure 4-15 plots the CPU allocated to the

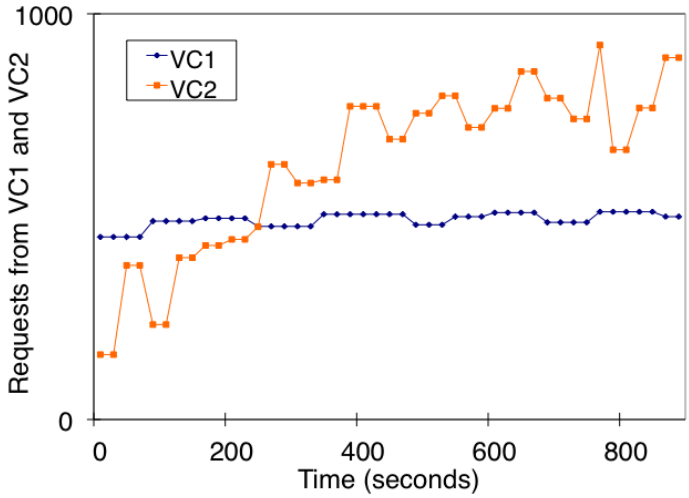

Figure 4-16. CPU requests from two virtual containers (VC1, VC2) that share limited resources.

virtual container during the experiment and about 44% resources can be saved using the dynamic allocation.

Comparing the fuzzy-modeling and fuzzy-prediction, both approaches can produce accurate short-term resource prediction for local controllers. Fuzzy modeling applies clustering techniques to provide concise data presentation, resulting smaller rule base (less than ten fuzzy rules during the experiments) than fuzzy-prediction approach (about several tens of fuzzy rules in the experiments).

4.5.4 Global Controller Validation

The last set of experiments investigates the global controller's allocation of limited resources among multiple competing virtual containers. Two virtual containers (VC1, VC2) running on the same server node compete for the available CPU cycles (1GHz). VC1 serves a fixed workload, which has a constant request rate of 30 requests/sec; while VC2 receives an increasing workload with a request rate rising from 10 up to 60 requests/sec. The workloads used in this experiment only consider static Web requests.

Both local controllers of these two containers employ the fuzzy modeling approach to dynamically estimate their CPU demands for the workloads, and the amounts of resources requested during the experiment are plotted in Figure 4-16. The local

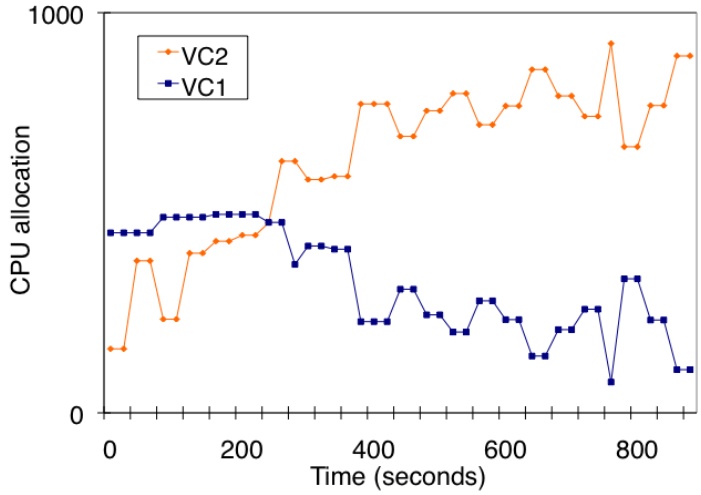

Figure 4-17. CPU allocation that favors VC2.

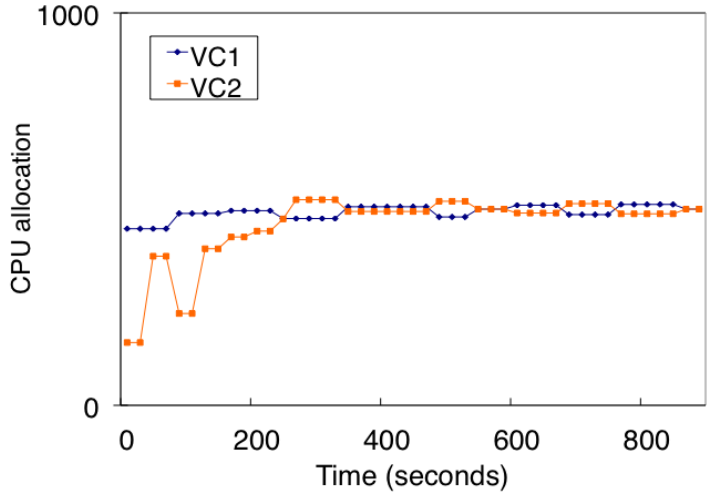

Figure 4-18. CPU allocation that favors VC1.

controller of VC1 requests around 500MHz of CPU throughout the entire experiment; while VC2 increases its CPU request from about 200MHz to more than 800MHz as its workload grows.

When the CPU needed by VC2 goes beyond 500MHz, the global controller responds to the resource shortage by reducing the allocation to VC1 or VC2. The allocation policy of the global controller is to maximize its profit by employing the greedy algorithm discussed in Section 4.5. Two simple scenarios are considered in

the experiments. In the first case, the profit rate of VC2 is higher than VC1; therefore, the global controller decides to satisfy the resource requests from VC2 by reducing the allocation for VC1 whenever a CPU shortage happens. Figure 4-17 shows the actual CPU allocations for the two containers throughout the experiment. The second case considers the opposite situation where VC1 has a higher profit rate and thus is favored in the resource allocation. In this case, VC2 suffers from the resource shortage when the global controller cannot allocate enough resources for both containers (Figure 4-18).

4.6 Related Work

To the best of our knowledge there is no prior work using a fuzzy modeling approach to data center resource management. The following briefly summarizes other work with some common elements with the approach described in this chapter.

Rule-based systems: This approach uses a set of event-condition-action rules (defined by system experts) that are triggered when some precondition is satisfied (e.g., when some metrics exceed a predefined threshold). For example, the HP-UX Workload Manager [99] allows the relative CPU utilization of a resource partition to be controlled within a user-specified range, and the approach of Rolia et al. [100] observes resource utilization (consumption) by an application workload and uses some fixed threshold to decide whether current allocation is sufficient or not for the workload. With the growing complexity of systems, even experts are finding it difficult to define thresholds and corrective actions for all possible system states.

Control theory: Approaches based on control theory have been applied to resource management to achieve performance guarantees. Most of the work assumes a linear relationship between the QoS parameters and the control parameters, and involves a training phase with a given workload to perform system identification. Typically, control parameters must be specified or configured offline and on a per-workload basis. Abdelzaher et al. [101] investigated this approach for QoS adaptation in Web servers. In [62][64], a nonlinear relation between response time and CPU allocation

to a Web server is studied, and a bimodal model is used to switch between underload and overload operating regions. To deal with time-varying workloads, more recent work applies adaptive control theory, in which models are automatically adapted to changes using online system identification.

Model-based: Previous research efforts [69][102][64][103] have been trying to model computer systems from different perspectives. Bennani et al. [71] predicts the response time and throughput for both online and batch workloads using multiclass open queueing networks. Liu et al. [104] uses AR models to map CPU entitlement percentage to the mean response time with a fixed workload. Chandra et al. [69] uses a time-domain queueing model to relate the resource requirements to its workload. Some of these approaches make simplifying assumptions such as using a single queue to model the whole system, which may fail to capture complexities of the relationship between application workload and resource usage. Some models are validated only using simulations.

Reinforcement learning (RL): Tesauro [105] proposed to use reinforcement learning for autonomic resource allocation. Compared with our fuzzy-logic-based approaches, both can automatically learn from past experiences without an explicit performance model. However, RL usually uses lookup table to store the information it obtained from training data. The size of table increases exponentially with the number of state variables, causing the scalability issue. Fuzzy-logic based system keeps its knowledge more efficiently in the form of fuzzy rules and fuzzy membership functions. RL may also have a long training time due to the absence of domain knowledge or good heuristics, while the construction of fuzzy rule base in our approach does not require time-consuming training. In [106], the authors proposed to use a hybrid RL method combining RL and queuing models, in which RL trains offline on data collected while a queuing model policy controls the system to avoid performance degradation in live online training.

Fuzzy control: Diao et al. [16] proposed a profit-oriented feedback control system for maximizing SLA profits in Web server systems. The control system applies fuzzy control to automate the admission control decisions in a way that balances the loss of revenue due to rejected work against the penalties incurred if admitted work has excessive response time.

The proposed resource management system in this chapter differs from the prior work in the following aspects:

- The resource control functions are separated between resource provider and application provider, which makes the design of data center resource management more flexible and robust. Each local controller tries to maximize its own profits by requesting ”just enough” resources for satisfying application SLAs as well as reducing unnecessary resource cost. The global controller takes into account the tradeoff between revenue obtained from satisfied resource requests and cost from violations of resource SLAs.

- Fuzzy-logic-based approaches provide a generic approach to representing the relationship between system variables. It can be easily applied to any type of applications hosted in virtual containers. This approach makes no underlying assumption of the workload characteristics, and can learn any type of relationship very fast. Especially, the fuzzy system can efficiently model the nonlinear system with dynamically changing operating conditions.

- The resource management process is automatic without any human intervention. The fuzzy rules are automatically learned from online monitoring data and the knowledge base is updated continuously as new data arrives, enabling the system to capture transient or unexpected workload changes.

4.7 Conclusions

This work presents a flexible two-level resource management system that is able to provide high quality of service with much lower resource allocation costs than worst-case provisioning. At application level, in order to make local controller accurately estimate the resource demands for different workloads, two fuzzy logic based methods - fuzzy modeling and fuzzy prediction - are proposed to guide resource allocation based on online measurements. Both approaches have the adaptive learning ability requiring no domain knowledge about the system. Specifically, the fuzzy modeling approach

characterizes the relationship between the workloads and the corresponding resource requirements, while the fuzzy prediction builds a mapping from recent resource usage to future resource needs. ”Adaptive” means the knowledge can be easily updated when new information is available to adapt to the system changes and reflect the most recent system conditions. The global controller at the resource-pool level tries to find the optimal resource allocation based on the proposed profit model, towards maximizing the total profit of the data center.

Our approach, in conjunction with virtualization techniques, can provide application isolation and performance guarantees in the presence of changing workloads by dynamically allocating resources at fine time granularity, which results in high utilization and low cost as well. The proposed resource management system is implemented on a virtualized data center testbed and evaluated using applications that are representative of e-business and Web-content delivery scenarios. Both synthetic and real-world Web workloads are used to evaluate the effectiveness of the approach. The experimental results show that the system can significantly reduce resource cost while still guaranteeing application QoS in various scenarios.

CHAPTER 5
MULTI-OBJECTIVE OPTIMIZATION IN VIRTUAL MACHINE MANAGEMENT

Chapter 4 introduced our initial work on resource management in virtualized data centers. It addresses dynamic optimization of resource allocation among competing applications that are running in their virtual machines and sharing physical resources. A simplified profit model is used by the global controller, which only considers the revenue and cost of allocating resources to different applications. There remain some questions to be answered.

1. What happens if we consider other optimization objectives such as reducing operational energy cost, avoiding thermal hotspots? How can an optimal solution be found with multiple possibly conflicting goals?

2. How can virtual machines (VMs) be placed on physical servers and how does the placement affect the optimization of data center resource management? Considering a large number of virtual machines and hosts, how can the best placement be found in an efficient way?

3. How can virtual machine migrations be used to adapt to dynamically changing environments in data centers without incurring high overhead?

In this chapter these questions are addressed as follows. Section 5.2.1 and 5.2.4 discuss the multiple objectives considered in managing virtual machine placement and how to use a fuzzy-logic-based approach to combine different objectives. Section 5.2.3 and 5.2.5 present an improved genetic algorithm to efficiently and globally search the space of virtual machine placement solutions with fuzzy multi-objective evaluation. In section 5.3 dynamic virtual machine migration to adapt to changes of system conditions and workloads is addressed using a cross-layer approach.

5.1 Problem Description

5.1.1 Multi-Objective Optimization

A great amount of work has been devoted to techniques to optimize resource management of a data center. Earlier work mostly focuses on improving resource usage while maintaining application performance guarantees [107][50][108]. Currently,

power consumption [7][8][109] and thermal dissipation [110] are significant contributors to data center operational costs. To reduce these costs, the use of virtualization to consolidate workloads and turning off unloaded servers has been proposed to achieve greater energy savings [6][75][52][111]. Work in [112][9] proposed a temperature-aware workload placement approach to minimize peak temperature. Most of the extant work focuses on only one or at most two specific aspects of management, such as minimizing power consumption, balancing thermal distribution, or maximizing resource usage. However, these may be conflicting objectives when considered all together. For example, tightly packing virtual machines onto a small number of servers and turning off other servers is an effective way to reduce energy costs. However, concentrating workload on a subset of the system resources can cause heat imbalances and create hotspots, which may impact cooling costs and degrade server life and performance. An effective VM placement strategy should consider tradeoffs among all these objectives.

5.1.2 Virtual Machine (VM) Placement and Migration

Two types of virtual machine placement are considered in our work. (1) Initial (or static) VM placement, to decide how to place a number of virtual machines at once in an unloaded data center, considering both VM requirements such as CPU, memory and IO bandwidth, and physical host capacities and platform requirements such as power and temperature. (2) Dynamic VM placement, to dynamically reassign VMs to hosts due to the changes of system conditions or VM requirements caused by dynamic workloads. For both tasks, the global controller tries to simultaneously optimize multiple potentially conflicting objectives, including the elimination of thermal hotspots, the minimization of total power consumption, and achieving desired application performance. However, these two types of tasks have different characteristics and should be solved using different strategies. The VM initial placement has long-term effects because extensive change in VM placement is impractical due to the resource overhead incurred and time consumed by multiple VM migrations. Also, the initial placement typically occurs much

less frequently than dynamic placement, e.g when the data center starts its operation, after reset/idle states, special usage regimes and when dynamic provisioning leads to unsatisfactory states. In these and other initial VM provisioning scenarios, the global controller can take a long time (relative to dynamic placement) to determine the VM placement, which is an NP-hard optimization problem. A sophisticated algorithm, such as the one proposed in our work and [9][55] should be used to search the solution space globally to achieve better performance. For dynamic placement, to quickly adapt to changes in the system or workloads, the controller is required to make decisions at runtime so as to, for example, reduce performance losses or mitigate thermal anomalies. Another important consideration for dynamic VM placement is the resource overhead (including power consumption) and performance loss incurred from migrating VMs. A complete new VM placement without considering current placement is impractical. Being one of the optimization methods with the least complexity and overhead, an online local search heuristic is a possible alternative for VM dynamic placement and the one proposed in our work.

The following two sections detail our work on initial VM placement and dynamic VM migration using multi-objective optimization approach.

5.2 Initial Virtual Machine Placement

In the scenario considered in this work each user requests the use of one or more applications with an expected quality of service that requires a certain amount of resources and the data center responds to the request by deploying a virtual machine dedicated to the applications and allocating required resources to it. Two types of resource mapping are involved – the mapping from application workload to resource requirements and the mapping from virtual resources to physical resources. Based on the work described in Chapter 4, a two-level control architecture (see Figure 5-1) naturally supports these two mappings through local controllers at the virtual-machine level and a global controller at the resource-pool level. A local controller implemented

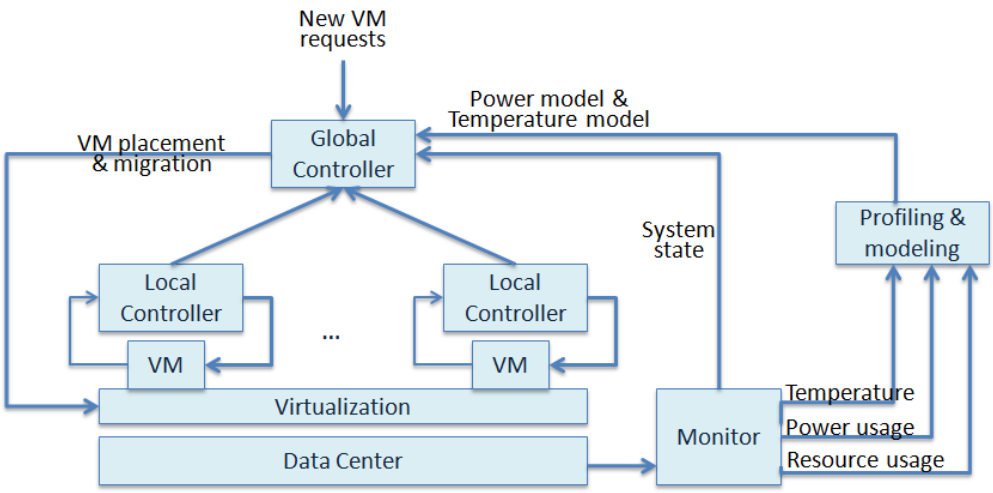

Figure 5-1. Two-level control architecture and information flow for autonomic resource management in a virtualized data center.

in every virtual machine is responsible for determining the amount of resources needed by an application and asking for more or less resources to guarantee application performance at minimum cost. A global controller determines virtual machine placement and resource allocation. The work described in Chapter 4 focused on the design of local controllers that use fuzzy logic-based modeling approaches to adaptively model the relationship between application workloads and their resource demands. This chapter concentrates on the design of the global controller at the resource level.

5.2.1 Multi-Objective VM Placement Decision

As illustrated in Figure 5-1, the global controller receives resource requests from users which are expressed as VMs with specific resource needs. The size of a VM is represented as a d-dimensional vector in which each dimension corresponds to one type of the requested resources (e.g., CPU, memory and storage). Resources on physical servers are allocated as “slices” along multiple dimensions according to the resource demands of VM requests (see an example in Figure 5-2). Each VM is assigned to a slice of a server and the resources consumed by the VM are bounded by the size of this slice. The monitoring system of the data center measures system information including resource usage, power consumption and temperature of each server and collects them

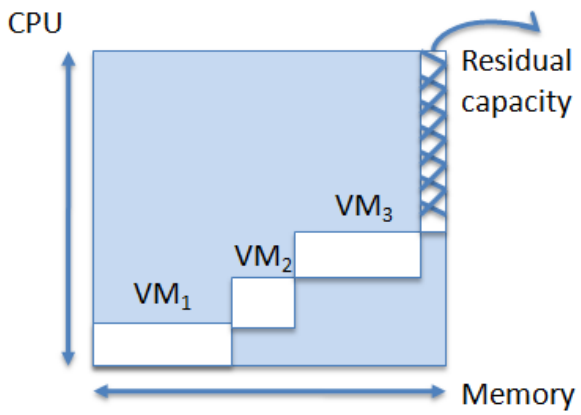

Figure 5-2. An example of resources allocated to three VMs placed into a single physical machine.

into a centralized profiling repository. The profiling and modeling components utilize the system measurements to create models of power and temperature, which are in turn used by the global controller to optimize its placement decisions, which considers the following factors.

Resource Wastage: The residual resources available on each host may vary largely with different VM placement solutions. In anticipation of future requests, the resources left on each server should be balanced along different dimensions. Otherwise, unbalanced residual resources may prevent any further VM placement, thus wasting computing resources. As Figure 5-2 illustrates, the outside shaded rectangle represents the total CPU and memory capacity of a physical server. The host's resource capacity is reduced along each dimension by placing three VMs and allocating resources to them. The three small rectangles indicate the amount of resources allocated to each VM. The crosshatched area in the figure denotes the residual resources available for future allocation. In the example of Figure 5-2 the host has a lot of unused CPU capacity but little memory available causing the host to not be able to accept any new VM because of memory scarcity.

To balance the resource usage along different dimensions, the following notation is used to calculate the potential cost of wasted resources. R_i represents the normalized

residual resource (i.e., the percentage of residual resource to the total capacity) along dimension i. Using subscript k to identify the dimension that has the smallest normalized residual capacity, the wasted residual resource on a server is calculated as the sum of differences between the smallest normalized residual resource and the others $W = \sum_{i \neq k}[R_i - R_k]$. Therefore, the bigger the differences of residual resources are among different dimensions, the more resources are wasted.

Operational Power: Power consumption tends to vary significantly with the actual computing activity. Extensive research work has been done to estimate power consumption using performance counters or system activity measurements. Based on the results from profiling the power consumption of an IBM BladeCenter (see detailed experimental data in Section 5.4.2), a commonly used linear power model [75][113] is used in our work to estimate the power consumption. In order to save energy, servers are turned off when they are unloaded (alternatively, low power states [114] other than "power-off" could be considered within the framework of our approach.). The total operational power C consumed by the servers is calculated as

$$C = \sum_j [P_j], p_j = \begin{cases} p_1 + p_2 U_j^{CPU} & \text{if } U_j^{CPU} > 0 \\ 0 & \text{otherwise} \end{cases}$$

, where P_j and U_j^{CPU} denote the power consumption and CPU utilization of jth server.

Thermal dissipation: Thermal performance is one of the critical metrics in data center management. Sharp spikes in server utilization may result in disruptive downtimes due to generated hotspots. According to the well-known duality between heat transfer and RC circuit electrical phenomena [110], a thermal RC circuit can be used to model the steady state temperature of a processor $T = PR + T_{amb}$ where P denotes the power consumption, R denotes the thermal resistance, and T_{amb} is the ambient temperature. Using the linear relationship between power consumption and CPU activity, the temperature is related to the CPU load of the host according to the

Table 5-1. Symbols used in VM placement problem formulation.

Symbols	Explanation
M	Number of physical servers
N	Number of VM requests
[c_j^{CPU} c_j^{mem}] (j = [1 ... M])	Capacity vector of the jth server
[r_j^{CPU} r_j^{mem}] (i = [1 ... N])	Resource requirements of the ith virtual machine
$a_{ij} \in [0, 1]$	Allocation matrix in which $a_{ij} = 1$ if vm_i is allocated to the jth server
W_j	Resource wastage of the jth server
P_j	Power consumed by the jth server
T_j	Temperature of the jth server
U_{CPU}	CPU utilization
U_{mem}	Memory utilization

equation $T = (p_1 + p_2 U^{CPU})R + T_{amb}$. This linear relationship between CPU temperature and CPU activity is confirmed by our profiling study of CPU temperature conducted on an IBM BladeCenter (see data in Section 5.4.2). Furthermore, research work has shown that the cooling cost increases if the temperature is unbalanced across a data center [112][9]. Thermal management in data centers aims at mitigating individual hotspots, keeping temperature within a safe operating range and balancing temperature across data center servers.

Each of the above-discussed factors represents an optimization objective during VM placement. Based on the observed system states, the global controller utilizes the power and thermal models to estimate the future system state and select the best placement based on the optimization criteria. Many research efforts have focused on power and thermal management leading to proposals of different power and thermal models. One advantage of our proposed global controller is that it can incorporate any type of models, depending on the investigated systems. The models used in our work were inferred from measurements on the IBM BladeCenter system mentioned in Section 5.4.

The proposed multi-objective VM placement optimization problem is formulated as follows (Table 5-1 lists the symbols used in this rest of the chapter:

Goal: $\min \sum_{j=1}^{M} W_j$ and $\min \sum_{j=1}^{M} P_j$ and $\min \max(T_j)$

Constraints: $\sum_{i=1}^{N} r_i^{CPU} a_{ij} < c_j^{CPU}, \sum_{i=1}^{N} r_i^{mem} a_{ij} < c_j^{mem} j = [1, ..., M], \sum_{j=1}^{M} a_{ij} = 1$

The first two objectives are to minimize the total resource wastage and power consumption by all the servers. The third objective function is to minimize the peak temperature among the servers. The first constraint constrains the allocated resources from each server to not exceed its capacity. The second constraint ensures that each virtual machine is allocated to one and only one of the servers[2] . Consider N virtual machines to be placed on M available servers – there are a total of N^M possible placement solutions, making a full enumeration impractical to find the best solutions. The following shows how to apply an improved grouping genetic algorithm to efficiently search the solution space.

5.2.2 Basics of Genetic Algorithms

Combinatorial optimization is concerned with methods for optimization over discrete choices (typically a large number). Many combinatorial problems are classified as NP-hard, such as the well-known task scheduling, packing, and VLSI placement problems. Heuristic techniques for solving optimization problems in an approximate way have been used widely. Among them, genetic algorithms (GAs) [115] have shown success in tackling various classes of combinatorial problems.

A genetic algorithm is a stochastic and population-based search technique. It operates on a set of encoded strings called chromosomes, each representing a solution in the search space and being assigned a fitness value that reflects the solution's goodness with respect to the optimization objectives. A population of candidate solutions evolves to better solutions iteratively. In each iteration, referred

[2] Current virtualization techniques already allow the creation of VMs that run across multiple servers; for simplicity this case is not considered but the proposed approach can be used by either redefining individual physical resources as collections of servers or considering multi-server VMs as multiple one-server VMs that must be co-scheduled (an additional constraint in the problem formulation).

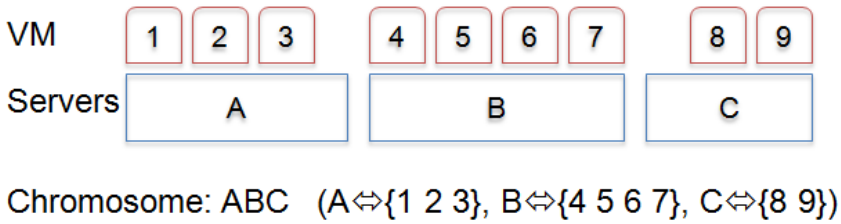

Figure 5-3. An example of VM placement and its corresponding chromosome.

to as a generation, a new set of strings is created by genetic operations (crossover and mutation) on the current solution pool to form the new generation. In crossover operations, two individuals (i.e. solutions) are selected as parents and portions of their strings are used to produce new strings representing new solutions. Mutation is another common genetic operator that is applied to a single chromosome in which some of the strings are randomly selected and changed.

5.2.3 Grouping Genetic Algorithm

The grouping problem is to group a set of items into a collection of mutually disjoint subsets. Falkenauer [116] pointed out that a classic genetic algorithm (GA) performs poorly on grouping problems such as bin-packing and introduced the grouping GA (GGA), which is a GA heavily modified to suit the structure of grouping problems. A special encoding scheme is used in GGA in order to make the relevant structure of grouping correspond to genes in chromosomes. In addition, special genetic operators for crossover and mutation are used to suit the structure of chromosomes. To further improve the performance, a new operator, called ranking-crossover, is propsoed and used in this work, as explained in the following.

Encoding: In grouping problems, the objective function is defined over groups rather than isolated objects. Therefore, the encoding schema in GGA is group oriented. Figure 5-3 illustrates an example of virtual machine placement. Nine virtual machines are partitioned into three groups (one per physical machine) and the corresponding

chromosome features three genes, each of them encoding a group of virtual machines allocated to one of three servers.

Crossover: The aim of the GGA crossover operation is to produce offspring out of two parents in such a way that the children inherit as much as possible of useful information from both parents. The GGA crossover randomly selects a portion of the first parent (i.e., some of the groups) and injects it into the second one. Some VMs could appear twice in the solution, so the groups (physical machines) containing them in the second parent are eliminated. Some of the VMs could be missing as a result of this elimination step – GGA uses a local heuristic, such as first-fit (which this work tried), to reinsert the missing VMs.

However, this crossover operator does not perform very efficiently in our case because the inheritance is performed completely blindly with random selection and insertion, and it is unlikely to obtain good results from a relatively small number of trials. A ranking-crossover is proposed to enable new generated solutions to inherit the good features from their parents more efficiently. The first step is to evaluate each individual group (a physical server with its hosted VMs) based on three types of efficiencies (discussed below) which correspond to the three aforementioned optimization objectives.

Resource usage efficiency ($E_{resource}$): It reflects how well the resources of different types are utilized. The goal is to fully utilize the resources in all dimensions. In the case of resources with CPU and memory, the efficiency is defined as the product of CPU usage and memory usage, i.e., $E_{resource} = U_{CPU}U_{mem}$.

Power consumption efficiency (E_{power}): It reflects how much useful work is produced under certain power consumption. $E_{power} = \frac{workload}{power} = \frac{U_{CPU}}{p_1+p_2U_{CPU}}(p_1+p_2)$ (the factor p_1+p_2 is used to make the efficiency value fall into $[0 \sim 1]$ range). The power consumption efficiency monotonically increases with CPU usage, and reaches the highest point when CPU usage is 100%.

Thermal efficiency ($E_{thermal}$): A logistic function is used to calculate the thermal efficiency as $E_{thermal} = 1/(1 + e^{(T-Ts)})$. The efficiency value decreases rapidly when the CPU temperature goes beyond the safe range (T_s is set to 70°C in the experiments discussed in Section 5.4).

The values of all these three efficiencies are in the $[0 \sim 1]$ range. The group evaluation function uses the average value of the three efficiencies to evaluate groups in each solution. The next step is to compose a new solution by selecting the groups from the parents in a deceasing order of evaluation values. When a group is selected, its VMs that appear in a previously chosen group are eliminated. In this way, the new generated solutions inherit the "good" structured groups and are able to evolve to better solutions quickly.

Mutation: GGA's mutation is also group oriented. A few groups (physical machines) are randomly selected and eliminated. The virtual machines in those groups are inserted back in a random order using a first-fit heuristic algorithm. However, the evaluation (see details in Section 5.4) showed that this operation is not very useful for our case because of the blind deletion and insertion.

5.2.4 Fuzzy Multi-Objective Evaluation

The proposed virtual machine placement attempts to minimize several (possibly) conflicting objectives. In order to use GGA to solve multi-objective problems, the fitness function used for selecting new generations of candidate solutions should reflect all objectives. The potential solutions obtained through GGA are evaluated using the following proposed fuzzy-logic-based system.

Consider our virtual machine placement problem where the goal is to minimize resource wastage, power consumption and maximum temperature. Three linguistic variables - *resource wastage* (w), *power* (p) and *temperature* (t) - are introduced and one linguistic value is defined for each variable, namely, fuzzy sets *small resource wastage* (sw), *low power consumption* (lp) and *low temperature* (lt). Membership

functions of these fuzzy sets are decreasing functions of the variable values, since the smaller the value, the higher is the degree of satisfaction. The search algorithm seeks to find the solutions that are nearest to each individual goal. Hence, the evaluation of a solution can be expressed by the following fuzzy rule:

IF solution x has small resource wastage (sw), AND low power consumption (lp) AND low temperature (lt), THEN x is a good solution.

The most desirable solution is the one with the highest membership in the fuzzy sets sw, lp, lt. Using the ordered weighted-averaging fuzzy operator proposed by Yager [117], the above fuzzy rule evaluates to the following, $\mu(x) = \beta min(\mu_w(x)\mu_p(x)\mu_t(x)) + (1-\beta)avg(\mu_w(x)\mu_p(x)\mu_t(x))$ (β is set to 0.5 in the evaluation experiments described in Section 5.4), in which $\mu_w(x)$, $\mu_p(x)$, and $\mu_t(x)$ represent the membership degree of solution x in the fuzzy sets defined by sw, lp, and lt, respectively. $\mu(x)$ is the membership value for solution x in the fuzzy set of good solutions. The solution with the highest $\mu(x)$ is the one that best meets all the goals and is reported as the best solution.

The membership functions for the fuzzy set sw, lp, lt are linear decreasing functions. The following calculation is proposed to determine the lower and upper bounds of the membership functions. The total CPU and memory requirements of all VM requests are represented by R_{CPU} and R_{mem}, and the CPU and memory capacity of each physical server is c_{CPU} and c_{mem}. The ideal minimum number of servers needed to host all the VMs is $m_{min} = \max(R_{CPU}/c_{CPU}, R_{mem}/c_{mem})$ (by assuming that VMs can be partitioned and each server is fully utilized by the VMs running on it), therefore the lower bound of power consumption is $P_{lower} = m_{min}(p_1 + p_2)$. The maximum number of servers for hosting VMs is $m_{max} = \min(M, N)$, so the upper bound of power consumption is $P_{upper} = m_{max}p_1 + R_{CPU}p_2$. To determine the lower and upper bounds of resource wastage, we use r_i^{CPU} and r_i^{mem} to represent the percentage of CPU and memory requirements of the ith VM. The lower bound of total resource wastage is

```
I. Initial_GGA
  S=Population size
  Ng=Number of generations.
  Ro=Crossover rate
  Rm=Mutation rate
  P ← Initialization(S)        /* Generate initial population */

II. Iteration_GGA(S, Ng, Ro, Rm, P)
For i = 1 to Ng
  For j = 1 to S*Ro
    (x,y) = Select(P)                    /* Select parents x and y randomly*/
    offspring[j] = RankingCrossover(x, y)
  Endfor

  For j = 1 to S*Rm
    y  = Select (P)                     /* Randomly select a parent */
    offspring[j + S*Ro] = Mutation(y)
  EndFor

P = EvaluateSelect(P, offspring)      /* Evaluate and select the best S solutions */
EndFor
```

Figure 5-4. Improved grouping genetic algorithm with fuzzy multi-objective evaluation for initial virtual machine placement.

$W_{lower} = |\sum r_i^{CPU} - \sum r_i^{mem}|$ and the upper bound is $W_{upper} = \sum |r_i^{CPU} - r_i^{mem}|$. Based on the thermal model discussed in Section 5.2.1, the lower and upper bounds of CPU temperature are $T_{lower} = p_1 R + T_{amb}$ and $T_{upper} = (p_1 + p_2)R + T_{amb}$.

5.2.5 GGA with Fuzzy Multi-Objective Evaluation

Figure 5-4 shows the major procedures of the proposed GGA algorithm with fuzzy multi-objective evaluation. The algorithm consists of two parts, randomly generating a number of solutions to form an initial population and reproducing new generations of solutions from the existing solution pool. For our problem, the initial population is produced as follows.

- S permutations of VM request orderings are randomly generated.

- For each VM request sequence, the first-fit algorithm is used to allocate the VMs to the physical servers. In this way, S different placement solutions are generated.

During each generation of GGA a set of offspring are produced by the ranking-crossover operator discussed above. This operator ensures that the offspring inherit their parents' important properties. The crossover and mutation rate are the percentage of new solutions generated from existing solution pool for each generation using crossover and mutation respectively. The generation selection is based the evaluation of each solution using the proposed fuzzy-logic based multi-objective evaluation. All three objectives are transformed to their corresponding fuzzy sets represented by their membership functions. By evaluating the fuzzy rule, the membership value of each placement solution is regarded as its fitness value. A number S of the best placement solutions are chosen from the solution pool comprising both the parents and their offspring for new generation. Therefore, the average fitness of the population and the fitness of the best individual solution increase in each generation.

5.3 Dynamic Virtual Machine Migration

This section describes our work on the dynamic VM placement problem. The main decisions required to solve this problem are when, which and where to move VMs. In most prior studies, the trigger for dynamic VM migration only depends on either the states of VMs (or the performance of their hosted applications) or the resource usage of their hosts, without considering the other important information from the platform layer such as power usage efficiencies and temperature distribution. Incorporating the information from both the virtualization layer and the platform layer, we identified three conditions for dynamic VM placement including thermal emergency, resource contention and low power efficiency, which will be explained in detail in this section.

The decisions of which and where to migrate VMs are based on the proposed multi-objective optimization approach with the migration cost being taken into the consideration. In addition, a reliable and robust detection and selection approach using

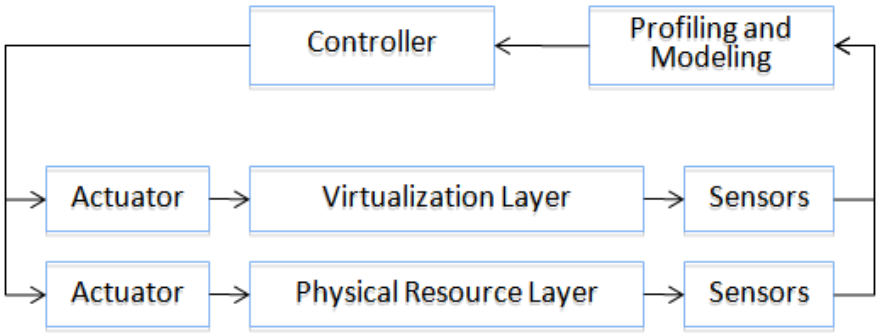

Figure 5-5. Cross-layer control architecture and information flow for dynamic VM management in a virtualized data center.

sliding-window and trend analysis is incorporated into the decision-making process for dynamic VM placement to achieve stable system state and avoid wasting resources and time for unnecessary control actions.

5.3.1 Cross-layer Profiling, Modeling and Controlling

Figure 5-5 shows the proposed cross-layer approach for managing virtual machines and their hosts in a virtualized datacenter. Both the platform layer and virtualization layer have multiple sensors for monitoring resource usage, power consumption and temperature of server nodes, as well as resource utilization of individual VMs hosted on the those nodes. (It is also possible to incorporate application-level performance information retrieved from inside of VMs. However, it is unclear what should be a set of standard metrics for application performance due to the diversity of applications coexisting in a data center. In order to provide a general framework for all types of data center environments, application performance data is not used in our system implementation.) Sensor data are collected from across multiple nodes into a centralized profiling repository which can be accessed by a global controller for managing both the physical-resource layer and virtualization layer in a data center. The profiling and modeling components shown in the figure utilize the monitored system data to create models of power and temperature, which are later used by the global controller to optimize its decisions.

5.3.2 Conditions for Dynamic VM Placement

As explained in the introduction, dynamic VM placement is needed in the following three conditions:

Thermal Emergency: The temperature of each server should be maintained below a threshold, because overheating of components will cause thermal cycling, and eventually hardware failures. In addition, high server temperature will increase cooling costs because additional cooling energy is required to eliminate hotspots. If localized hotspots are detected, virtual machines with high intensive workloads are moved to relieve the situation and reduce cooling costs.

Resource Contention: Performance degradation may result from multiple VMs competing for resources. Moving the VMs out of the troubled host can enable them to obtain the necessary resources to maintain their performance as well as alleviate the resource contention.

Low Energy Efficiency: When a server becomes idle or its utilization is very low, it still consumes a large portion of energy compared to when it is busy – this is wasted energy as little or no useful work is done. In such situation, large energy savings can result from moving all the VMs out of the idle host and turning off the host.

Figure 5-6 shows the control flow of the controller. After retrieving monitoring data from the profiling repository, the initial thread (called monitor thread) created by the controller checks the above three conditions in turn. The reason for sequential checkup is because the control decisions for different conditions may conflict with each other. The checks take place sequentially according to their priorities as specified by the system administrators and their order can be easily changed when necessary. Whenever a condition is met on a monitored server, the controller initiates a new thread (called action thread) to determine the actions and execute them on the chosen VMs and servers. The reason for this is to avoid blocking the monitor thread from checking other servers due to time-consuming actions such as VM migrations. However, conflicts and inconsistency

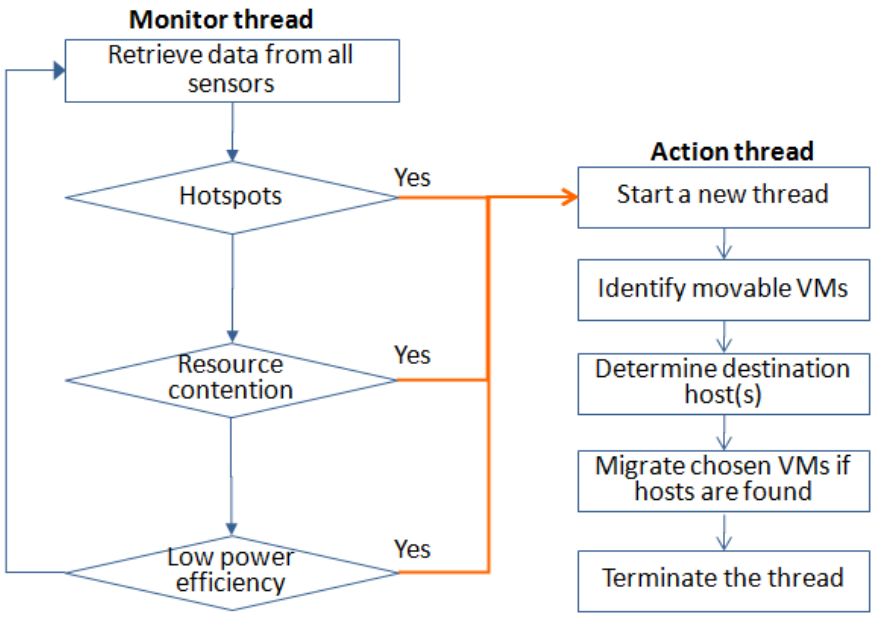

Figure 5-6. Control flow of the controller.

issues may arise when multiple action threads are trying to determine destination hosts for their VMs and migrate them at the same time. For example, the same server may be chosen as destination host by multiple action threads and become overloaded if multiple VMs are migrated to it. In addition, a destination host experiences high resource usage during VM migration, causing the controller incorrectly trigger VM migration. Therefore, the process of destination selection is considered as a critical section and a lock is applied to synchronize it among multiple action threads so that there is at most one thread in selecting host process at any given time. Once a server is selected as destination host by an action thread, it is temporarily marked until migration is finished so that other action threads will not select it as destination and the monitor thread discard the monitored data from that server during the migration to avoid triggering migrations.

5.3.3 Controller Functionality

There are basically three functions implemented in the controller: condition detection, VM selection and destination selection. The following will explain them in detail.

Condition Detection: This function determines when a control action such as migrating VMs and turning on/off physical servers needs to be performed. As shown in Figure 5-6, the controller periodically checks the three conditions mentioned in the previous section using the profiling data generated from the sensors, and if a condition is met, the controller starts a new thread to further investigate the situation. This is an event-detection problem and simple detection method such as threshold-based detection can be used. For example, when the current temperature monitored on a server exceeds a predefined threshold, the server is identified as a hotspot and the controller is activated in order to possibly trigger a VM-migration action. However, in a typical data center setting, the hosted application workloads change dynamically over time. This causes the system conditions, such as resource utilization, power consumption and CPU temperature, to fluctuate over time. Without recognizing this transient nature of variation, the single-threshold detection may trigger many unnecessary actions and even worse, cause system oscillations (e.g., continuous VM migrations and machine on/off switches).

To make condition detection both reliable and timely, a two-level detection approach is proposed as follows. For the first-level detection, the basic idea is that to avoid false detection over transient changes, i.e., persistent observations of threshold violations over a period of time are required to trigger the detection. A sliding-window detection is applied, in which the analysis of the time-varying data such as CPU temperature and resource utilization is performed over the values covered by a window of finite length. The monitoring data are sampled at a constant interval, and the data window keeps sliding over time. In each window, an alarm is triggered if the percentage of data outside

of a predefined threshold $T_{level-1}$ is larger than a value V. The first-level detection may be incorrect for data showing an increasing or decreasing trend. The reason is that although most of the absolute data values in the window are outside of the safe range, the near-future data is likely to fall back into the safe range because of the trend. The second level detection uses trend analysis over the data in the sliding window to predict the near-future data values. If the predicted value violates a pre-specified threshold $T_{level-2}$, a second alarm is generated. When there is no prior knowledge about those time-varying data, a linear model is used for trend estimation and the model parameters are determined using the linear least-squares approach. Only if both alarms from the two-level detection are triggered, the controller starts a thread to further investigate the situation on the servers which generate alarms and determines the control actions. Details on how to select window sizes and thresholds under different conditions are discussed later in this section.

Virtual Machine Selection: The most important control action considered in this paper is VM migration, a convenient way of moving hosted workload around a data center in order to mitigate a thermal emergency, alleviate resource contention or turn off hosts to save energy. The VM selection function determines which virtual machines should be moved considering the benefits and costs of the migrations. The following discusses the strategies for selecting VMs under three conditions, namely:

Thermal Emergency: The objective of migrating VMs under this condition is to bring the temperature of the overheated server into the safe range as soon as possible. Previous research work has shown that CPU temperature is highly related to CPU activity [118][112][9]. Therefore, those VMs with high CPU utilization should be the first to move in order to lower the CPU temperature efficiently. At the same time, VM migration incurs processing and IO overhead, which in turn increase power consumption and server temperature. The migration process for a VM mainly involves copying the VM's memory file and image to the destination and also keeping track of which pages

are being modified if live migration is applied. Migrating VMs with smaller memory file or footprint can reduce the migration time and overhead. Considering the above two aspects, the VMs on the flagged server are sorted in decreasing order of the ratio of their CPU utilization to memory size (USR). Choosing to migrate VMs with the largest values of USR maximizes the CPU load reduction (and the implied lowering of temperature) per data bytes moved.

Resource Contention: The goal of VM migration under this condition is to make VMs and their hosted application obtain sufficient resources to meet performance guarantees. The VM selection algorithm needs to identify which VMs are competing for the insufficient resources since moving an idle VM does not improve the situation. On the identified server, each VM's utilization of the resource under pressure is retrieved and the average utilization of all VMs is calculated. Only the VMs with utilization higher than the average are considered to be competing VMs and others are filtered out. Also considering the migration cost, the competing VMs are further ordered in increasing order of their memory size. The VMs with smaller size are chosen for migration.

Low energy efficiency: When the controller identified an idle server or a server with low resource utilization, all the VMs running on the server need to be moved out in order to bring the server down.

Destination Selection: To determine a new destination host for the selected VM, the host selection function has the following considerations.

Temperature: After VM migration, the destination host's CPU temperature may rise due to the added VM's workload - the selection function must make sure the temperature is still in the safe range after the migrated VM is deployed. Furthermore, recent research work has shown that the cooling cost increases if the temperature is unbalanced across a data center [112][9]. Placing a VM in the coolest server can help balance the temperature distribution. Utilizing the temperature model inferred from

monitored system data, the selection function is able to predict the CPU temperature of the destination host after VM migration.

Power: Although moving a VM and its workload does not change the total power consumption under a homogenous data center setting, the selection of destination may affect future power usage. For example, moving a VM to an idle server will keep it from shutting down to save energy. On the contrary, tightly placing VMs on a small number of servers creates more opportunities for other servers to be turned off. From the point view of saving power, it is preferable to place the VM in the most occupied server that still has enough resources. For a system consisting of heterogeneous servers, placing a VM in different places may result in different power consumption. The selection algorithm can use power models learned from the modeling process to predict future power consumption of candidate destination hosts.

Performance: To avoid generating new resource contention and guarantee each VM's performance, preference is given to the hosts that have the largest amount of free resources that are sufficient to host the migrating VMs.

Every one of the above considerations tries to optimize the destination selection from different perspectives. However, the selection results from optimizing different objectives may conflict with each other. One of the most common methods for multi-objective optimization is to convert multiple objectives into a single-objective function using a weighted sum. It is hard to identify the appropriate weights for different objectives because the values of the objectives have different units and ranges. To solve this problem, three utility functions named temperature efficiency, power efficiency and performance efficiency, are defined over three objectives, respectively.

- Temperature efficiency: The temperature efficiency decreases monotonically with increasing temperature so that high temperature indicates low efficiency and vice versa. The objective is to keep CPU temperature in the safe range, so this utility function is designed to decrease slowly when the temperature is far below the safe threshold and drops rapidly when approaching or going beyond the threshold. Many nonlinear functions such as exponential functions and polynomial

functions can serve this purpose. For simplicity, we use a polynomial function in the form of $EFF(T) = 1 - T_m m \geq 2$, in which $Eff(T)$ represents the efficiency value for temperature T, and m is the degree of the utility function. In order to make the efficiency value fall into $[0 \sim 1]$ interval, T is then normalized using the function $\hat{T} = \frac{T - T_{low}}{T_{high} - T_{low}}$, in which $[T_{low}, T_{high}]$ is the safe range of CPU temperature T. Choosing any degree m in a small range (e.g., $m \leq 5$) does not result in large variations in efficiency values so that it does not affect the selection results significantly.

- Power efficiency: This efficiency reflects how much useful work is produced by the consumed power on an active server. By using the linear power model with CPU utilization $P = p_1 + p_2 CPU$ (parameter p_1 and p_2 are obtained from the modeling process), the power efficiency is defined as $Eff(P) = \frac{Workload}{Power} = \frac{CPU\%}{p_1 + p_2 CPU\%(p_1 + p_2)}$. In this utility function, the workload is represented by the CPU utilization (CPU%) and the factor $(p_1 + p_2)$ is used to make the utility value fall into $[0 \sim 1]$ range. The power efficiency increases monotonically with increasing CPU usage, and reaches the highest point when CPU usage is 100%.

- Performance efficiency: It reflects the extent of use of resources of different types. To prevent resource contention, the efficiency decreases rapidly when the usage of one or more of the resources approaches and exceeds the maximum allowed for guaranteeing the workload performance. The setting of maximal resource usage is shown in Table 5-2. Similar to temperature efficiency, the resource utilization used in the utility function of different types is normalized into $[0 \sim 1]$ range. The utility value of consolidation is set to the minimum value of efficiency among different resource types.

With a little knowledge about the system, the utility functions not only help smoothly express the degree of preference (desirable, tolerable, undesirable etc) under different values for each objective, but also normalize the ranges of all the objectives into the same interval [0, 1]. Therefore, it is easy to combine them into a single objective function consisting of the sum of three utility functions.

Given the resource requirements of the selected VM, the destination selection algorithm iterates over all the available servers and retrieves their current resource utilization information, and predicts the future state including temperature, power consumption and resource utilization on each server after the selected VM is deployed using the power and temperature models learned from the modeling functions. The algorithm then calculates the combined utility values for each candidate host and the

server that returns the highest value is chosen as the destination host for the migrating VM.

5.3.4 Stabilization Considerations and Complexity Analysis

Data centers typically host a variety of applications having dynamically changing workloads, and therefore the resource requirements of the VMs and also the resource utilization of their hosts may change dramatically over time. Because of such variations, a seemingly suitable VM placement or migration decisions can quickly become inappropriate with respect to performance, temperature or power on destination hosts, which in turn would generate more unnecessary and costly migration actions. Even worse, the system state [1] may oscillate due to continuous VM migration that keep triggering new migrations and/or server turn on/off actions. The decisions on when and where to move VMs, and when to turn on/off servers should not only be based on the current conditions, but also on the desirability of such a decision over a certain time into the future. To ensure the performance and stability of the controller, the following approaches for parameter selection are applied in the three functions of the controller.

When to move: As discussed in the beginning of above section, a sliding window detection approach is used to reduce false alarms on transient changes. A small window size produces fast and aggressive detection, while large values cause slow detection but at the same time can filter out more transient threshold violations and avoid unprofitable migrations, therefore the window size selection has an impact on the performance of detections. The trend analysis also avoids the situation when the resource demands of the moved VMs fall back to normal ranges as before soon after the migration, making

[1] We refer the state of a host at time t as its resource usage, temperature and power consumption at t, and the state of a data center is the collection of the states of all hosts. The state of a VM is represents by its resource demands. A stable state means that the percentage of monitored data (representing the state) inside of a predefined range is larger than a value V over a specified period of time T.

the control actions unnecessary and wasteful of resources and power. One of the necessary conditions to ensure profitable migrations is that the rate at which state changes of a host or a VM occur is slower than the rate of VM migration is performed. In another word, the stable interval (i.e., the period of a detected host or a moved VM staying in stable state) must be longer than the migration time. If a VM or host is in a stable state for a period of time T, it is predicted that to stay in that state for another T with high confidence so that the future stable interval T_{stable} is estimated as T. The necessary condition is that $T_{stable} > T_{migration}$. The detection algorithm prescribes stable states of the VMs and their hosts must be observed over a sliding window before an alarm is triggered, therefore, the window size is regarded as the stable interval. By making the window size larger than a typical VM migration time prevents unnecessary migration to a certain degree. In our implementation, the migration time is measured at about 30 seconds for a typical size VM with 1024M memory, the sliding window T_{window} is set to 2 minutes (which is considered large enough to produce profitable migration while not delaying the detection too much in our implementation) so that a VM is predicted to stay in its state after the migration for at least a period of time $T_{window} - T_{migration}$ and the migration is predicted to bring some benefit to the system. The similar analysis can be applied to server turning on/off actions. The window size for detecting low power efficiency is set to 6 minutes, which is four times of the average period for turning on/off a blade node (measured from our IBM BladeCenter).

The thresholds for condition detections used in our prototype implementation are listed in Table 5-3. The reason for setting level-1 thresholds lower than level-2 thresholds is that the detection algorithm can catch the ascending trend faster and avoid triggering the descending trend. Through profiling variety of workloads on our IBM BladeCenter testbed, (for experiment purpose) we observed that the CPU temperature of a blade node varies between about $20 \sim 55^{\circ}$ depending on the intensity of the workloads. The level-1 and level-2 thresholds for thermal emergency are set to 48° and 50°. For resource

contention, we set 90% as high utilization thresholds for CPU, IO and network (the reason for setting 90% instead of 100% is to give a small buffer for transient utilization changes).

Where to move: Similar to the above discussion, we want to make sure that the destination hosts are in a stable state to avoid generating new hotspots or resource contention, which in turn triggers new migrations and possibly cause system oscillation. To incorporate stability and trend analysis into destination selection, the algorithm uses not only the most recent monitoring data of the servers, but also historical data for calculating the predicted combined utility values. Let $[d(t-W), \ldots, d(t-1), d(t)]$ represent the time-varying monitoring data over a period of W, and $\mu(t)$ and $\sigma(t)$ be their time-varying mean and standard deviation. The trend $tr(t)$ is calculated using least-squares as discussed in Section B. At time t, the predicted data is calculated as $\hat{d(t)} = \mu(t) + \sigma(t) + tr(t)W$. With the second term, the selection algorithm prefers not to select a host with highly varying system state.

During the move: During the VM migration, the original and destination hosts may experience temporary high CPU, disk and network IO usage. To avoid triggering further VM migration on those hosts, the controller will ignore the monitoring data collected from them until the migration finishes.

To evaluate the complexity of the controller, suppose there are M physical machines and each machine hosts N VMs. The controller periodically checks every server in the data center, which takes $O(M)$ time. Once the detection is triggered on a particular host, the VM search algorithm sorts all VMs according to the criteria discussed in the above section running on that host, and the sorting algorithm has $O(NlogN)$ complexity. The controller iterates over every VM in the sorted order and stops if a destination host if found for the VM being checked. Since the host searching algorithm's complexity is $O(M)$, the worst case for selecting hosts for VMs is $O(M * N)$, in which every VM on the host is checked. The combined complexity of the algorithms used in the controller is

$O(M * N) + O(NlogN)$. The maximum number of VMs can be hosted on a single host is limited by the available physical resources such as CPU and memory, therefore, the combined complexity is linear on the number of total physical hosts.

5.4 Experimental Evaluation

This section summarizes the experimental evaluation of the proposed control system for virtual machine placement and migration in a data center environment. Section 5.4.1 and Section 5.4.2 discuss the virtualized data center setup for the experiments and the modeling data obtained from profiling the testbed, respectively. In Section 5.4.3, the proposed multi-objective GGA placement algorithm is evaluated using a set of simulation experiments and compared with traditional bin-packing offline algorithms and also single-objective GGA approaches to show its performance, scalability and robustness over a wide range of environments. The simulation uses the modeling parameters obtained from profiling an IBM BladeCenter, so that the results can accurately capture the real system behavior. Section 5.4.4 discusses the evaluation of the proposed approach to dynamic VM placement and migration on an experimental testbed build up by an IBM BladeCenter.

5.4.1 Experimental Setup

We built our testbed on an IBM BladeCenter with nine HS21 blades connected through 2 gigabit switch modules for hosting virtual machines, and an IBM Totalstorage disk system providing storage for VM disk images over NFS. Each blade node has two Xeon (Dual-Core Woodcrest) 2.33GHz processors and 8GB RAM, one of them is reserved for running the global controller, and eight of them are used for hosting Linux virtual machines injected with different types of workload of variable intensities.

5.4.2 Profiling and Modeling

In order to obtain the power and thermal models required by the global controller, we used IBM's advanced management module [42] to measure power consumption and CPU temperature of Blade servers. The IBM BladeCenter has 14 HS21 blades each

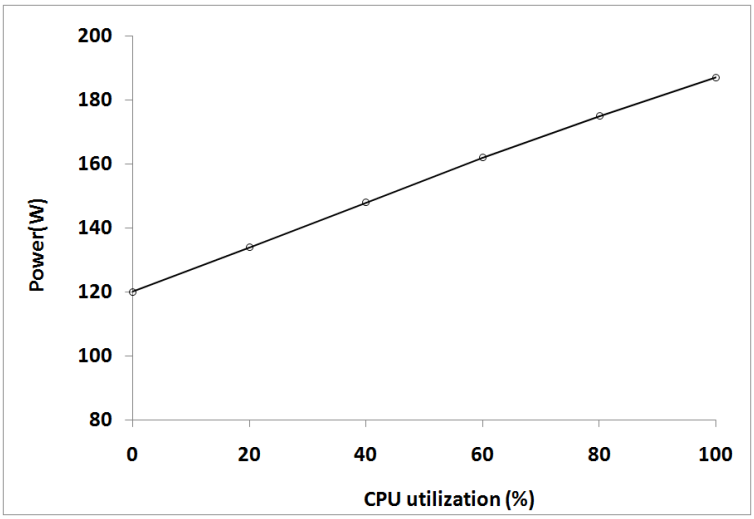

Figure 5-7. Power consumption with varying CPU utilization.

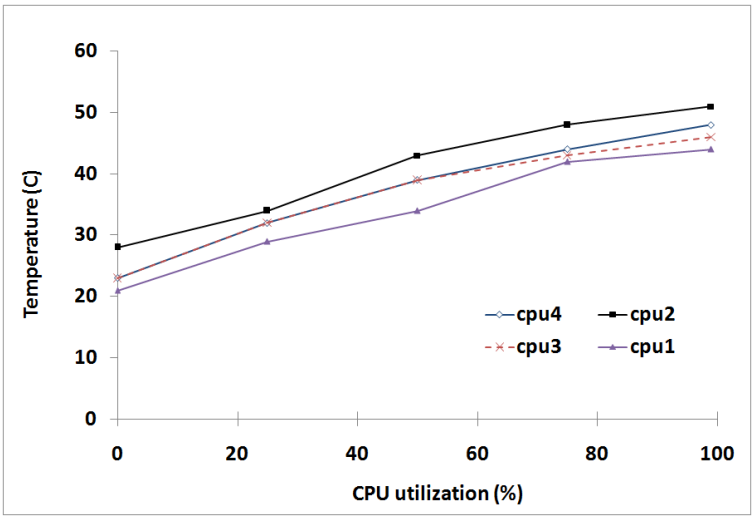

Figure 5-8. CPU temperature with varying CPU utilization.

with two Xeon (Dual-Core) 2.33GHz processors with 4MB L2 cache, 8GB RAM, and 73GB SAS disk.

Figure 5-7 shows the power consumed by the blade server and Figure 5-8 plots the CPU temperature of four cores [2] with respect to CPU utilizations. It is clear that the

[2] The temperature measurements used for modeling capture the average temperature of the four cores on a server.

power consumption is linearly related to the CPU utilization and the CPU temperature is also an approximately linear function of CPU utilization without considering the effect of heat generated by other servers nearby.

5.4.3 Evaluation of Multi-Objective Initial VM Placement

We use three sets of experiments to evaluate the proposed multi-objective VM placement approach with respect to performance, scalability and robustness. The virtual-machine CPU requests (measured in GHz) are uniformly distributed over the set 0.25 0.5 1 1.5 2 2.5 3 4 and memory requests (measured in GB) are uniformly distributed over the set 0.25 0.5 1 1.5 2 2.5 3 4 to simulate different sizes of VM requests. The number of the available servers and the VM requests are configured during the setup phase to simulate different sizes of the problem. Table 5-4 lists the parameter setup for the three sets of experiments. For each setting, we generated random inputs and ran the experiments 20 times and computed the average results (except for Figure 5-10) discussed below.

Performance: In the first set of experiments, we compared the proposed multi-objective GGA approach with six competing algorithms including four well-know offline bin-packing heuristics and two single-objective GGA (SGGA) approaches.

FFD-CPU and FFD-MEM: First-fit-decreasing (FFD) places items in a decreasing order of size, and at each step, the next item is placed to the first available bin. FFD-CPU represents the FFD solution sorted by virtual-machine CPU requirements and FFD-MEM is the FFD solution sorted by memory requirements.

BFD-CPU and BFD-MEM: Best-fit-decreasing (BFD) places a virtual machine in the fullest server that still has enough capacity. BFD-CPU and BFD-MEM represent the BFD solutions sorted by CPU requirements and memory requirements, respectively.

SGGA-P and SGGA-T: Both algorithms use GGA to search the solution space and the fitness value is evaluated with respect to power for SSGA-P, or to temperature for SSGA-T.

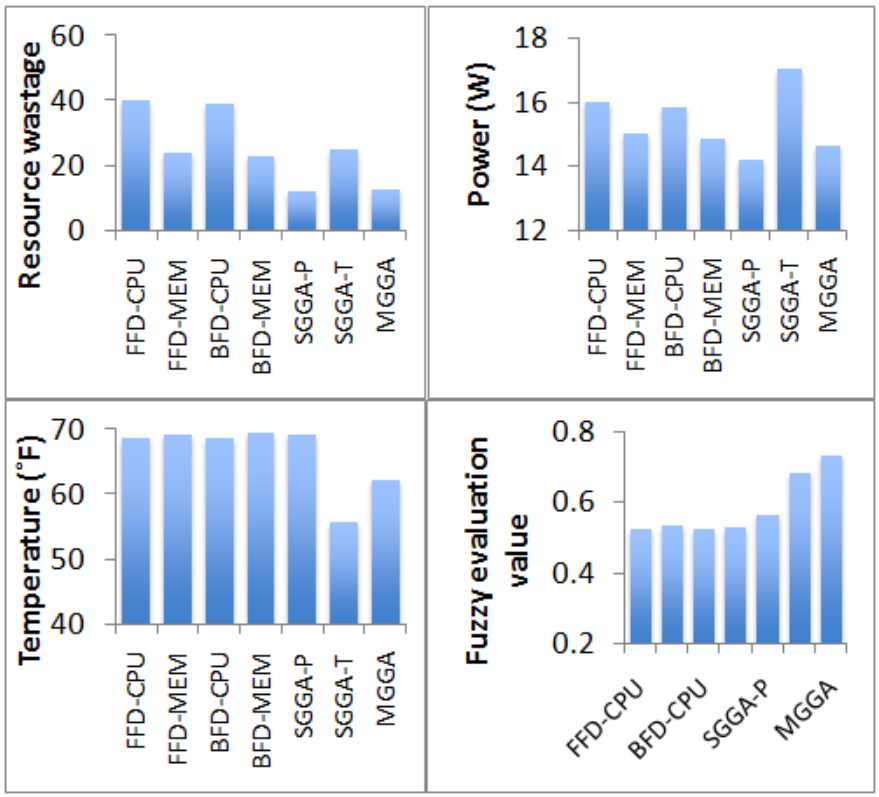

Figure 5-9. Performance comparisons of seven placement algorithms. Comparison of proposed multi-objective optimization with six other competing solutions with respect to total resource wastage, power consumption, maximal temperature and fuzzy evaluation values.

MGGA: Different from SGGA, the fitness value for multi-objective GGA (MGGA) is evaluated considering all the three objectives including minimizing resource wastage, power consumption and maximum temperature.

Figure 5-9 compares the total resource wastage, power consumption, and maximum temperature as well as the fitness value for each of the algorithms under consideration. The key observations concerning this figure are as follows:

FFD, BFD and SGGA-P yield the highest temperature because they all tend to consolidate VMs into a smaller number of servers, resulting in higher resource utilization and higher temperature of the servers. Among them, SGGA-P produces the lowest power consumption because the improved GGA algorithm is able to search the solution space more efficiently and globally so that it can find the placement solutions with a

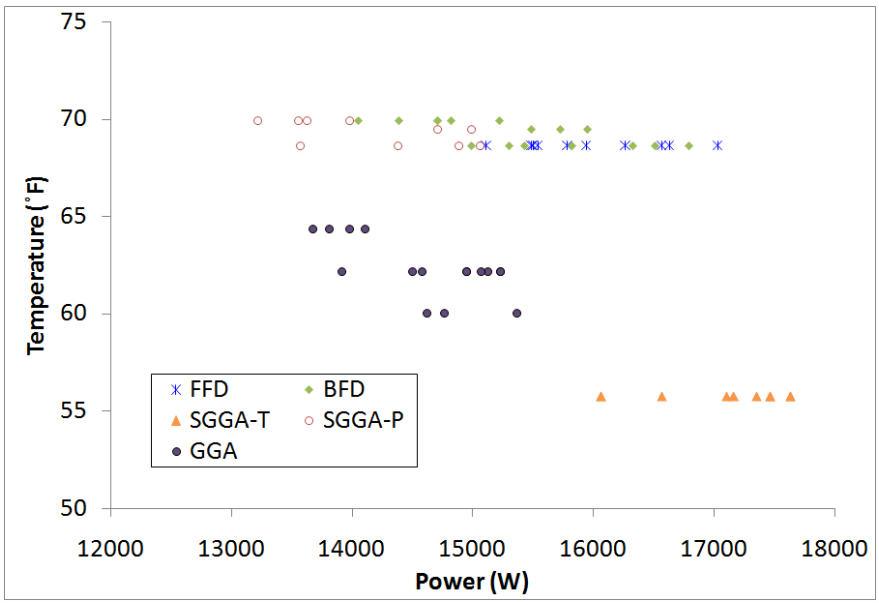

Figure 5-10. Solution points obtained from seven placement algorithms for twenty randomly generated 128-machine 250-VM inputs.

smaller number of used servers compared with FFD and BFD. The resource wastage of SGGA-P is also low because the placement tries to fully utilize the resources in all dimensions. On the contrary, SGGA-T yields the lowest temperature because the algorithm tends to evenly distribute VM requests to all of the available servers, therefore the resource utilization of each server is low as well as the CPU temperature. At the same time, SSGA-T generates the highest power consumption because no servers can be turned off to save energy. MGGA produces relatively low values for power consumption, peak temperature, and resource wastage because it takes all objectives into consideration and strives to find solutions that optimize every objective and achieve good balance among conflicting goals. The best performance of MGGA compared to other competing algorithms is also confirmed by the highest fitness value shown in Figure 5-9.

Figure 5-10 illustrates the solution points obtained by seven placement algorithms for twenty different 128-machine 250-VM inputs using a two-dimensional graph, in which x-axis represents the power consumption and y-axis is for the CPU temperature (The

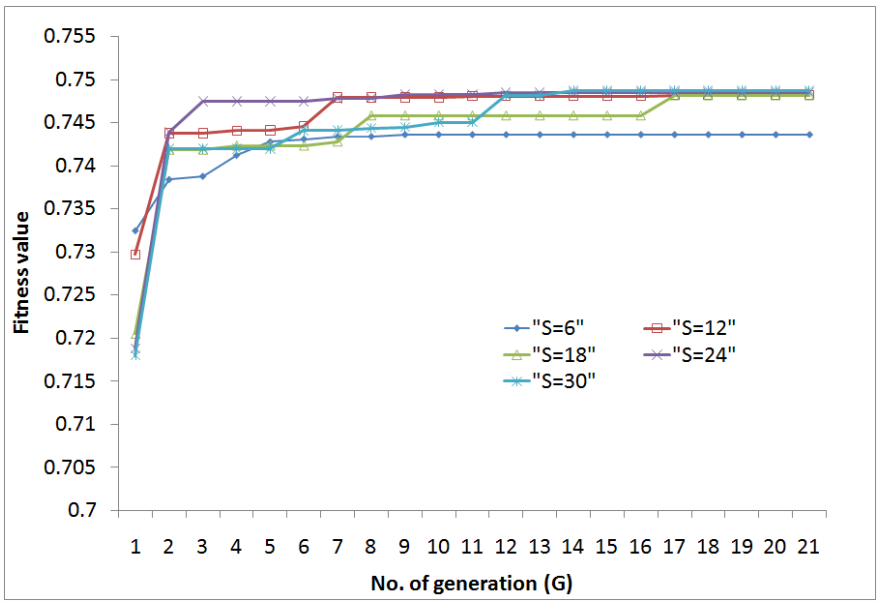

Figure 5-11. Fitness value of MGGA for different values of S and G.

dimension for resource wastage is omitted to make the figure clear.). Each point in the figure represents the solution obtained by one of the algorithms for every input. The points obtained by SGGA reside at the two ends of the figure. They either have the highest peak temperature and lowest power consumption (by SGGA-P), or lowest peak temperature and highest power consumption (by SGGA-T). The points of MGGA are located in the middle and achieve better balance between the two conflicting goals. The points obtained by FFD and BFD have the same peak temperature as SGGA-P but higher power consumption than SGGA-P, showing that they are not Pareto optimal because the solutions of SGGA-P are dominant to theirs in every objective.

Robustness: The initial solution size (S) and the number of generations (G) are two of the fundamental parameters for the GGA algorithm. The intuition is that the performance of the algorithm improves with larger values of S and G because there are more existing candidate solutions to explore and more generations of solutions being produced. The previous set of experiments uses small values for both parameters, while this set of experiments explores the sensitivity of these results to the various parameter values. Figure 5-11 shows the fitness values for different values of S and

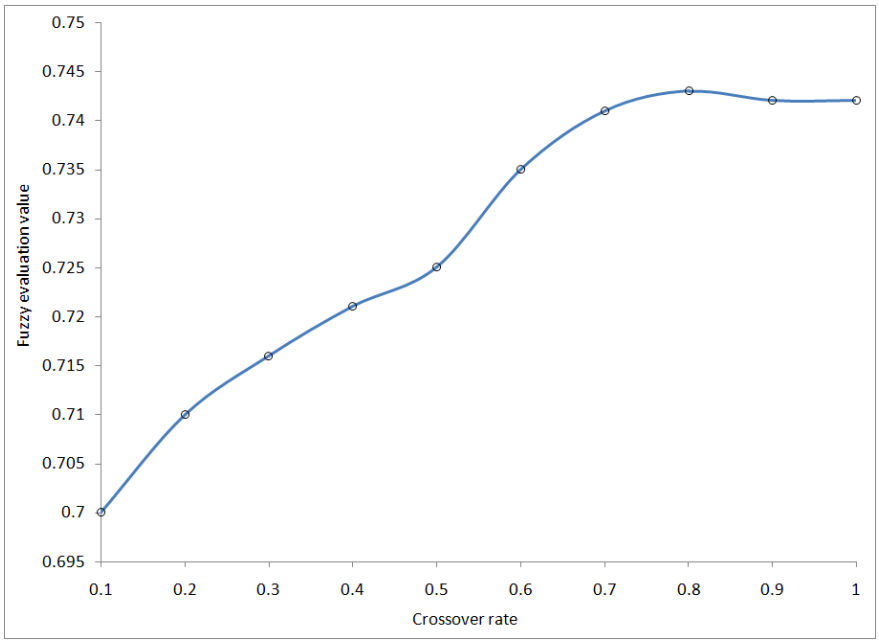

Figure 5-12. Fitness value of MGGA for different values of crossover rate.

G. When the initial solution size is very small (less than 5), the performance of GGA does not improve much even with a large number of generations because there are very few solution points available for GGA to evolve with. When the value of S exceeds 12, the marginal benefit of increasing initial solution size is rapidly decreasing, indicating that GGA has enough points to produce better solutions. Another observation is that in most cases, the performance stops improving after G goes beyond about 8, showing that the proposed ranking-crossover GGA algorithm can quickly improve the solutions and reach the optimal or sub-optimal points. These observations validate robustness of the proposed GGA approach in the sense that the performance obtained by the small "representative" set parameter values is very close to larger parameter values.

As mentioned in Section 5.2.3, crossover and mutation rate are another two important parameters of the GGA algorithm. A set of experiments was conducted to investigate the performance with respect to these two parameters. Figure 5-12 plots the fitness values of GGA with varying values of crossover rate. The performance stops improving after the crossover rate goes beyond 0.8. The experiments (the results are not shown

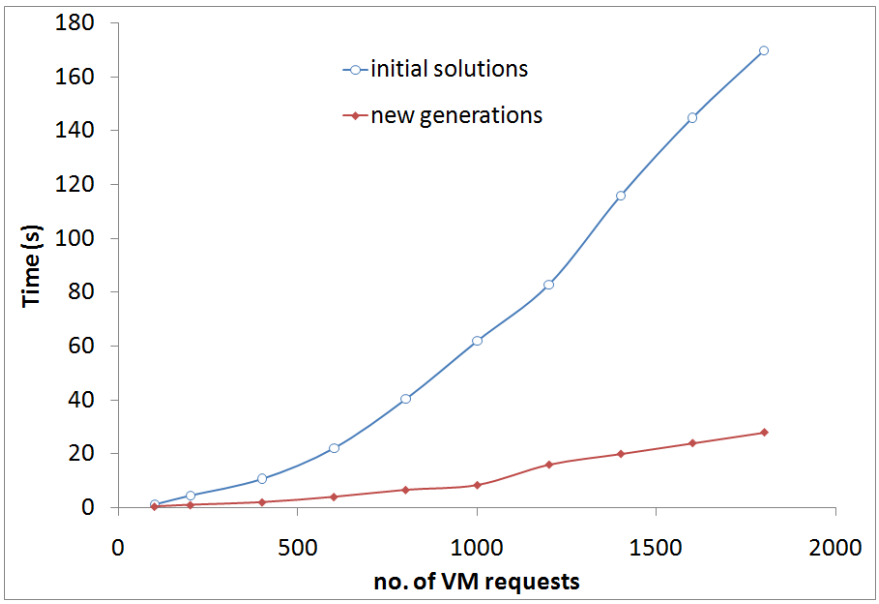

Figure 5-13. Fitness value of MGGA for different values of S and G.

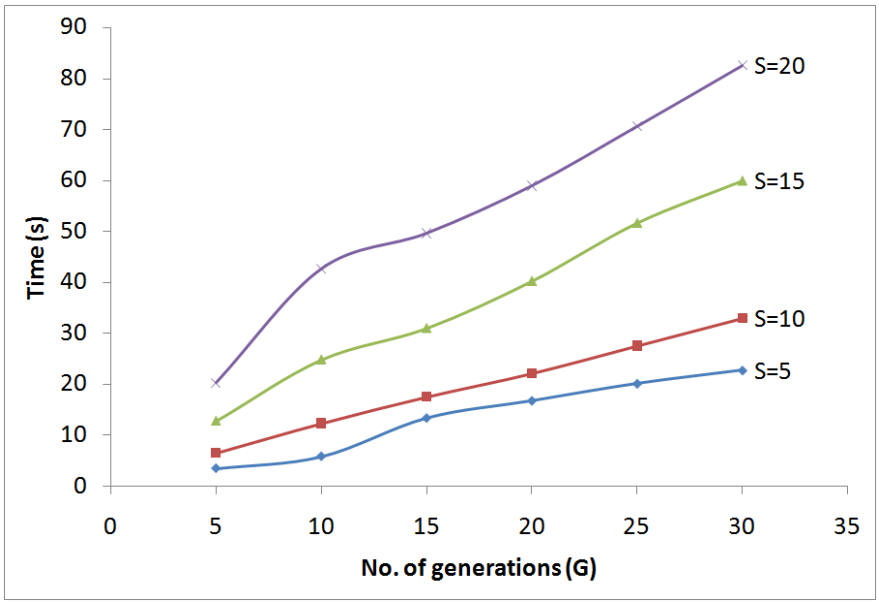

Figure 5-14. Fitness value of MGGA for different values of crossover rate.

in the paper due to the space limitation) also show that the mutation operator does not help improve the performance because the random deletion and re-insertion cannot steer the existing solutions better. Therefore, the rate of crossover and mutation are set to 0.8 and 0 for all other experiments.

Scalability: The last set of experiments is used to study whether the proposed GGA algorithm is scalable for large size of data centers and VM requests. In the experiments, the number of physical machines (M) is varied from 50 to 1000 and the number of VM requests (N) from 100 to 2000. The execution time is measured on a 2.00GHz Pentium M machine. Figure 5-13 plots the time of generating the initial population of solutions and successive generations with increasing problem sizes. The algorithm takes less than 3 minutes to solve the difficult 1000-machine, 2000-VM placement problem. The time to generate new solutions for a 128-machine 250-VM problem for different S and G is shown in Figure 5-14. It is clear that the execution time is approximately linear with respect to the values of G and S.

Analyzing the complexity of the GGA algorithm, it consists of two main parts. For the initial solution generation, the algorithm performs first-fit on a random permutation of VM requests. The complexity is $O(SNlogN)$ for generating a number of S solutions. In the successive solution generations, the most costly function is the placement evaluation. The algorithm evaluates all the VMs placed on each physical servers in every candidate solution, therefore the complexity is $O(NSG)$ for a number of N virtual machines, S solutions and G generations. Combining these two parts, the complexity of GGA algorithm is $O(SNlogN) + O(NSG)$, which yields a polynomial execution time.

5.4.4 Evaluation of Dynamic VM Migration

5.4.4.1 Prototype implementation

The Bladecenter runs VMware Server as the virtualization platform. The VMs are configured to have one CPU and 2 GB disk size. Their memory sizes are randomly selected from the range of 256M to 1024M, for generating different migration overhead. All the monitoring sensors are implemented in Perl and deployed as daemons running on each host. The resource utilization (including CPU, disk and network IO) information is collected from /proc and /sys system files provided by Linux; CPU temperature is monitored using the lm_sensors tool; and power usage is obtained through the IBM

Advanced Management Module. The sensors periodically produce data (every 10 seconds in our implementation) and store them to a profiling repository on the shared storage over NFS, which can be accessed by the global controller anytime.

The controller also runs as a daemon on the reserved blade node and all the functions described in Section III are implemented with Perl scripts. Periodically, it pulls out the monitoring data from the profiling repository, checks whether the conditions for VM migrations or others are met, determines control actions and sends them to the actuators implemented on the hosts.

5.4.4.2 Workload generation

To emulate a typical data center setting, a mix of different types of workloads is generated by deploying and running multiple benchmarks on the virtual machines. *SysBench* is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system running a database under intensive load. It has six test modes, allowing several system parameters to be varied, including file I/O performance, scheduler performance, memory allocation and transfer speed, POSIX threads, implementation performance, and database server performance (OLTP benchmark). Three of the test modes are used in the experiments, namely “CPU mode” to generate integer computation, “file I/O mode” for intensive disk IO activity, and “OLTP mode” to represent data base application. *Lookbusy* is another application for generating different types of synthetic loads on a Linux system. In CPU mode, the parameter settings allow the system to either maintain a constant CPU usage at the level specified, or periodically vary within a range by following a curve function (cosine function). The CPU usage level, curve range and period can be specified by the users. This benchmark is used to generate dynamically changing CPU workloads. *Linpack* is a benchmark for performing numerical linear algebra and can be used to measure of a system’s floating point computing power. It represents an HPC application in the data center testbed.

A Workload generator implemented in *Perl* is used to inject different workloads into each of the virtual machines hosted on the testbed. For every VM, the generator randomly selects a benchmark and selects its parameters randomly from their allowed ranges. Therefore, each testbed node is assigned a mix of varying workloads with different intensities.

5.4.4.3 Competing approaches

To evaluate the proposed approach and compare with some existing virtualization management approach, another two controllers are also implemented. The first one ignores stability issues and exposes the benefits of the proposed approach using operation windows and prediction to avoid control actions that lead to unnecessary migration and unstable configuration. This first controller triggers a VM migration whenever the monitored temperature, or resource utilization of a host exceeds a certain threshold. The host selection in the controller uses the most recent monitored data collected from the sensors instead of using a data history to incorporate its variation and trend. The second controller uses single-objective optimization for destination host selection and exposes the benefits of the proposed approach due to its use of multi-objective optimization over. This controller has three selection policies, 1) *coolest*: select the host having the lowest temperature as the destination for VM migration. 2) *idlest*: select the host with the lowest CPU load as the destination. 3) *fullest*: select the host with the highest CPU load while still having enough CPU capacity for the new VM.

To test the response to the three data center conditions aforementioned in Section 5.3, besides the random workloads on blade nodes, the workload generator intentionally created a series of events corresponding to hotspots, resource contention, and low energy efficiency conditions on randomly chosen servers (by adding or removing certain types of workloads on the VMs hosted on the servers), over a 2-hour run. Each event can be either transient (by running the workloads for less than 1 minute) or stable (the workloads last at least 10 minutes). All the following evaluation results are

obtained though a series of experiment runs, in each there are random workloads plus a sequence of events generated on the servers. The proposed multi-objective optimization with stabilization consideration in VM and destination host selection (MOS) approach is compared with a multi-objective optimization without stabilization consideration (MONS), single-objective optimization (SO) with coolest (SOc), fullest (SOf), and idlest (SOi) policy, and no control policy for each experiment run with the same generated workloads. Since application downtime is not a concern in our experiments, we chose offline migration (also called suspend-copy-resume migration) because it is quicker and consumes less resource than live migration. It is easy to incorporate live migration in our testbed if necessary.

5.4.4.4 Evaluation results and analysis

In this section, the overall benefits of proposed approach compared to no-control scenario are summarized first, followed by the more detailed results and analysis of how the system benefits individually separately from multi-objective optimization and stabilization.

Figure 5-15 shows the real-time monitoring data (CPU utilization, disk utilization, and CPU temperature) for all the blade nodes during an experiment run, which experienced four events, a transient CPU contention, a stable IO contention, a transient IO contention, and a stable CPU contention (also a hotspot event). The data is updated every 10 seconds From Fig. 4(a), it is seen that without any control actions, server 1 experienced very high disk IO usage (around $90\% \sim 100\%$) from about time period 60 till the end of the experiment. Server 0 began to have a stable high CPU load starting at time period 120, which also resulted in CPU temperature violation starting from around time period 150. Although Server 7 also experienced high IO usage, it is caused by only one VM and is not identified as an IO contention event by the controller. Fig. 4(b) shows that the controller reacted to the stable IO and CPU contention (also hotspot) events by migrating selected VMs to the best destination hosts can be found using multi-objective

Figure 5-15. Monitoring data (CPU utilization, disk utilization, and CPU temperature) from the blade nodes during an experiment run, which experienced four events, a transient CPU contention, a stable IO contention, a transient IO contention, and a stable CPU contention (also create a hotspot event).

optimization approach. The IO usage on server 1 and CPU usage on server 2 dropped back to safe range (around $50 \sim 60\%$) after VM migrations. The temporal high IO usage around time period 70 (on server 5), 140 (on server 6) and 150 (on server 5) are caused by VM migration. The temperature on server 0 was also kept in the safe range.

Figure 5-16 shows the time period of thermal and resource usage violation experienced by all the blade nodes during the experiment run with and without control. The results are normalized to the MOS approach. As seen in Figure 5-16, the temperature violation period is reduced by 80% and resource usage violation period is reduced by about 70% by dynamic VM migration using MOS approach. Figure 5-17 shows the performance of a VM which was running Sysbench oltp mode and hosted on

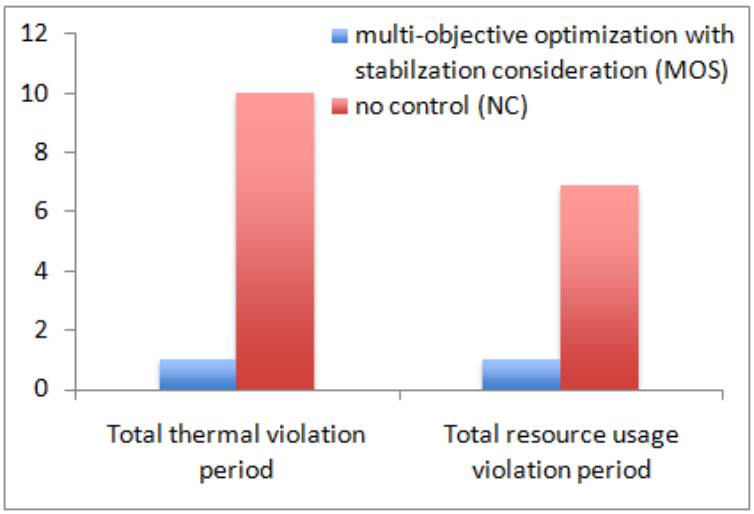

Figure 5-16. Comparison of total thermal and resource usage violation period between MOS approach and no control.

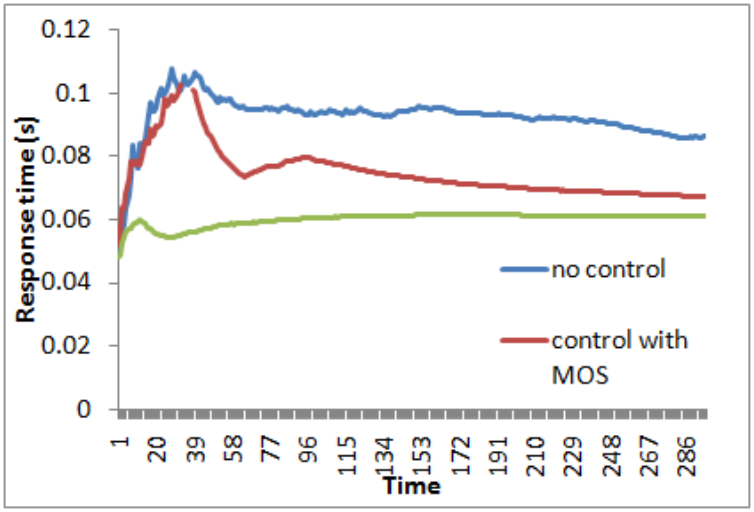

Figure 5-17. Comparison of the performance of a VM running Sysbench oltp mode on server 1 under no control, control using MOS and ideal case.

server 1 which experienced IO contention. The ideal performance is obtained by running the VM exclusively on a complete idle server node. It is seen that the response time increases dramatically due to the IO contention. The controller detected this situation and migrated the VM to another host using MOS. After migration, the performance of the VM improved quickly and gradually got close to the ideal performance. The other VMs running Sysbench on server 1 have the similar performance, which are not shown in the paper. Table 5-5 lists the timing and resource overhead for dynamic VM migration. The

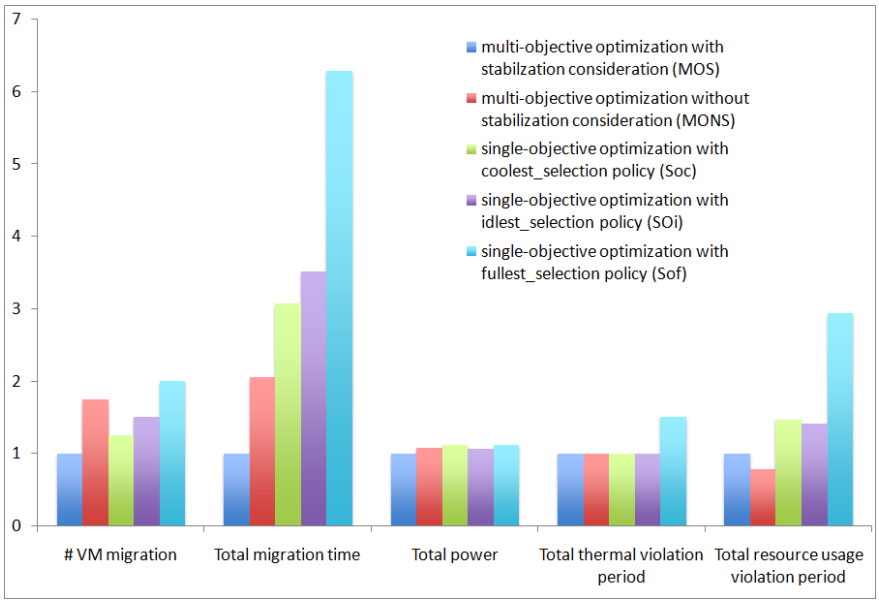

Figure 5-18. Comparison of proposed multi-objective optimization with stabilization consideration with four other competing solutions including multi-objective optimization without stablization consideration, single-objectivc optimization using coolest, idlest, and fullest host selection policy.

VM and destination host selection algorithm executed by the controller takes less than 1 second, and consumed very little resources (less than 1% CPU). The VM suspension on source host and VM start on destination host take a few seconds and utilize about 20% CPU. The main overhead for a VM migration takes place at copying VM files. It takes about 20 seconds to copy a VM with 256M memory and 40 seconds for a VM with 512M memory. During the file copying, both source host and destination host experience very high disk and network activity.

Figure 5-18 compares the performance of the proposed controller using multi-objective optimization with stabilization consideration (MOS) approach with the controller without stabilization consideration (MONS) and the controller optimizing single-objective in host selection (including coolest, fullest, and idlest) under five metrics including the total number of performed VM migrations, the total time used for VM migrations, the total consumed power, the total thermal violation period, and total resource usage violation period during the experiment. All the results are normalized to the MOS

approach. As seen in Figure 5-18, MOS triggered and performed the fewest number of VM migrations and consumed the least migration time. Since high CPU, disk and network overhead are incurred during VM migrations, lower number of VM migrations and less migration times indicates less resources are wasted for VM movements. The total thermal violation period records the time that the CPU temperature of any blade node exceeds the safe operating range. All the approaches except fullest_selection policy achieve very low thermal violation period. The reason is that fullest_selection policy tries to concentrate the workloads on fewer servers, resulting in high CPU load on certain hosts as well as their CPU temperature. The total resource usage violation period is the time that resource usage of any blade node is beyond the allowable resource usage for guaranteeing application performance. From the results, MOS and MONS approaches perform the best under this metric. The idlest and coolest selection policies have relatively higher violation period and the main reason for this is that both approaches try to optimize a single objective and does not consider the effects of other factors. For example, a server that hosts the VMs with high IO activity may have very low CPU utilization and CPU temperature as well. With coolest or idlest selection policy, a controller may migrate a VM with high IO workloads to such a server, resulting in IO usage violation and causing further VM migrations due to IO contention. The fullest_selection policy gives the worst performance because moving VMs and their workloads to a host already having high CPU load increases the chance of that server violating CPU usage and incurs more migrations.

5.5 Related Work

The problem investigated in this dissertation - mapping of VMs to physical servers - is related to a variety of research topics including workload placement on shared resources, dynamic resource allocation and the classical bin-packing problem.

The classical bin-packing problem is to determine how to put the items in the least number of fixed-space bins. This NP-hard problem has been extensively studied

(see [119] for a recent survey). One related problem, the application placement is theoretically studied in [120] in which the goal is to maximize the total number of applications that can be hosted on a shared hosting platform. The authors prove that offline-APP is NP-hard by reducing from Multidimensional Knapsack problem. First-fit, best-fit and worst-fit approximation algorithms that place applications in nondecreasing order of their requirements are discussed. Cardosa et al. [121] has investigated the problem of power-efficient VM allocation in virtualized enterprise computing environments. They leverage min, max and shares parameters, which are supported by the most modern VM managers. The objective function to be optimized includes the power consumption and utility gained from execution of a VM, which is assumed to be known a priori. The authors provide several heuristics for the defined model and experimental results. Verma et al. [122] studied CPU usage correlation between applications over long term to determine VM placement on a shared data center in order to save power as well as reduce performance violations due to VM consolidation.

More recently, thermal and energy management have received much attention, especially in large-scale data center environments. Some research investigates the placing of applications on energy/thermal-efficient locations [123]. A temperature-aware workload placement is presented in [112][118][9]. Sharma et al. [124] addresses the similar problem and proposes to measure cooling efficiency for guiding the workload placement. There is also some work lately on conserving power usage. Resource allocation is combined with energy management by turning off servers with no load, or low load after unloading the servers [125][126]. Some work [6][75][109][127] also considers using dynamic voltage/frequency scaling (DVFS) to further improve energy efficiency.

Using dynamic virtual machine migration capability on a virtualized platform to improve efficiency for resource usage and power consumption has drawn a increasing

attention recently. From the commercial side, VMware's Distributed Resource Scheduler (DRS) uses VM migration to dynamically balance computing capacity across a collection of physical resources. VMware DRS only responds to CPU and memory overload cases, and does not consider other platform factors such as temperature and power consumption. VMware Distributed Power Management (DPM) utilizes VM migration to consolidate workload on fewer physical machines and power off the unused hosts to save energy. There are also many works published in recent research literature. Many of them consider the dynamic VM migration as an optimization problem, for example, with an objective being to maximize total resource demands or minimize power consumption [55][56][57][58]. Some of them also consider migration cost and performance guarantees [128][55][59]. Khanna et al. [129] proposed an online algorithm for reconfiguring VM placement which is triggered by the violation of resource utilization thresholds. The goal is to maximize the gains associated with migration. Wood et al. [59] tackles a similar problem, in which migration is initiated to avoid violation of application SLAs or resource utilization thresholds. The migration algorithm moves workloads from the most overloaded servers to the least-loaded ones, while minimizing data transferred during migration. Bobroff et al. [128] describes a dynamic remapping VM to PM algorithm, in order to statistically satisfy SLA targets under dynamic workloads. pMapper [57] addresses the power and migration-cost-aware application placement problem in heterogeneous server clusters. Heuristic algorithms adapted from first-fit-decreasing are proposed to produce local optimal solutions. Entropy [58] makes use of the Constraint Programming (CP) paradigm in order to find the optimal placement for VMs and then tries to construct a VM reconfiguration plan with the least migration cost. Similarly, Mistral [55] addresses the dynamic VM placement by first finding the minimum number of nodes to host VMs using a modified bin-packing algorithm and then constructing a search graph from the current configuration with best-first search technique.

Table 5-6 lists the recent work on dynamic VM migration and compares them in relation to migration objectives, migration trigger (when to migrate), VM and destination selection (which and where to move). The trigger for dynamic VM migration in the above-mentioned work only depends on the states of VMs, the performance of their hosted applications, or the resource usage of their hosts, without considering information such as power usage and temperature distribution. The migration objectives only consider one or two aspects of data centers such as power consumption and migration cost. The approaches of finding which VMs to be moved and where to move them are either based on global search or local search. Although global search may produce better results, it becomes impractical for a large data center due to the large search space. In addition, the use of global search without considering existing VM placement causes a high number of migrations and high waste of resources. Utilizing the information from both the virtualization layer and the platform layer, we identified three conditions for dynamic VM migration, which are thermal emergency, resource contention and low power efficiency. The selection of VMs for migration in our work considers both the efficiencies of improving system conditions and migration cost under different conditions. We also proposed to use a multi-objective optimization approach for destination selection. In addition, a robust detection and selection approach using a sliding-window and trend analysis is incorporated into the decision-making process of the controller. This approach leads to stable system states and prevents the waste of resources and time for unnecessary control actions.

5.6 Conclusions

Enhanced virtualization technology enables the ability of workload consolidation and migration, as well as fine-grained control of resources assigned to a single virtual machine. At the same time, it brings the need for more intelligent VM management that can adapt to changing workload demands and satisfy various management goals as well.

In this work, the problem of VM placement and migration is formulated as a multi-objective combinatorial optimization problem aiming to simultaneously optimize possibly conflicting objectives including making efficient usage of multidimensional resources, avoiding hot spots, and reducing energy consumption. For initial VM placement in which a number of virtual machines are placed at once in an unloaded data center, modified genetic algorithm is proposed and developed to effectively deal with the potential large solution space for large-scale data centers. Fuzzy multi-objective evaluation is applied in the algorithm to combine the conflicting goals. We also investigated dynamic VM placement approaches to cope with workloads that over time change their resource requirements. In such cases, the initial mapping needs to be modified for better allocation of existing and new virtual machines. However, the high cost incurred by VM migration prohibits unlimited usage of this mechanism. The controller needs to minimize the impact of migration as well as satisfy other objectives and constraints, when modifying an existing placement and allocating new virtual machines. A cross-layer control system is proposed to manage the dynamic mapping of VMs to physical resources. The controller unifies the information from different layers to determine control actions such as when, which and where VMs need to be moved and when to turn on/off physical hosts. Three conditions including thermal emergency, resource contention and low energy efficiency are identified to need dynamic VM migration. Different strategies for selection of which VMs should be moved are used under different conditions, to improve the efficiency of the movement and reduce migration overhead. The selection of destination host for migrated VMs is posted as a multi-objective optimization problem and three utility functions, each representing an optimization objective, are used to combine them into single objective. Stabilization issues for condition detection and destination host selection are also considered in the design of the controller.

The profiling data obtained from measurements of power consumption and CPU temperature in an IBM BladeCenter are used for building the models of power consumption and CPU temperature of blade servers, which are then applied in the proposed placement and migration algorithm. For testing initial VM placement over a wide range of data center setting and VM requests, the simulation-based experiments studied the proposed approach with respect to its performance, scalability and robustness and showed the superior performance of the proposed approach compared with well-known bin-packing algorithms and single-objective approaches. For dynamic VM placement, an experimental testbed is implemented on an IBM BladeCenter. A mix of different types of workloads is generated to emulate a typical data center setting. The performance of the proposed controller is compared with another two controller, one without stabilization consideration and the other using single-objective approach. The results show that the proposed approach significantly reduces unnecessary VM migration and its associated resource overhead, avoids unstable host selection, and also improve the application performance, and efficiencies of resource and power usage of data centers.

Table 5-2. The list of utility functions.

Name	Utility functions	Symbols
Temperature efficiency	$Eff_i(T) = 1 - (\frac{T_i - T_{low}}{T_{high} - T_{low}})^m$	T_i: The temperature of server i T_{low}: The temperature of an idle server i (15C) T_{high}: The temperature of an overloaded server i (55C) m: degree of utility function (set to 3 in the implementation)
	$Eff_i(C) = min(Eff_i(CPU), Eff_i(IO), Eff_i(Net))$	CPU_i: The CPU of server i CPU_{low}: The CPU usage of an idle server (0%) CPU_{high}: The CPU usage of an overloaded server (100%)
Performance efficiency	$Eff_i(CPU) = 1 - (\frac{CPU_i - CPU_{low}}{CPU_{high} - CPU_{low}})^m$	IO_i: The disk utilization of server i IO_{low}: The IO usage of an idle server (0%) CPU_{high}: The IO usage of an IO overloaded server (100%)
	$Eff_i(IO) = 1 - (\frac{IO_i - IO_{low}}{IO_{high} - IO_{low}})^m$	IO_i: The network IO utilization of server i Net_{low}: The lowest network IO usage (0) Net_{high}: The highest network IO usage (20M bytes/sec)
	$Eff_i(Net) = 1 - (\frac{Net_i - Net_{low}}{Net_{high} - Net_{low}})^m$	m: degree of utility function (set to 3 in the implementation)
Power efficiency	$Eff_i(P) = \frac{CPU_i}{p_1 + p_2 CPU_i}(p_1 + p_2)$	CPU_i: The CPU usage of server i $p_1 and P_2$: constants for linear power function m: degree of utility function (set to 3 in the implementation)
multi-objective utility	$Eff_i = Eff_i(T) + Eff_i(C) + Eff_i(P)$	

Table 5-3. Thresholds and window sizes for condition detection.

	Level-1 Threshold $TH_{level-1}$	Level-2 Threshold $TH_{level-2}$
Thermal Emergency	46°C	50°C
Resource Contention	CPU: 85% IO: 85% Network: 85%	CPU: 90% IO: 90% Network: 90%
Low Energy Efficiency	CPU: 10%	CPU: 10%
Percentage P	80%	
Window size for thermal & resource contention	120 seconds	
Window size for low energy efficiency	360 seconds	

Table 5-4. Parameter setup for three sets of experiments.

Experiment set	Problem sizes	GGA Parameters
I (performance)	$M = 128, N = 250$	$S = 12, G = 8$
II (robustness)	$M = 128, N = 250$	$S = [2 \sim 100], G = [5 \sim 20]$
III (scalability)	$M = [50 \sim 1000], N = [100 \sim 2000]$	$S = [5 \sim 20], G = [5 \sim 20]$

Table 5-5. The timing and resource overhead for dynamic VM migration.

Events	Time (second)	CPU overhead	Disk overhead (%)	Network overhead (Mb)
VM and host selection	≤ 1	≤ 1	0	0
VM suspension	$3 \sim 6$	$10 \sim 20$	0	0
VM file copy	$20 \sim 30$ (256M) $40 \sim 50$ (512M)	Source: $20 \sim 30$ Destination: $20 \sim 30$	Source: 0 Destination: $60 \sim 100$	Source: $110 \sim 120$ Destination: $120 \sim 150$
VM start	$1 \sim 2$	$10 \sim 25$	0	0

Table 5-6. Related Work on Dynamic Virtual Machine Placement.

Paper	Migration Objectives	Migration Trigger	VM Selection and Destination Selection
Entropy [58]	Minimize the number of active nodes while maintaining performance.	VM-state-based (if the state of a VM changes : active, inactive)	Phase I: find the min. number of nodes to host all the VMs using dynamic programming. Phase II: construct a migration plan to achieve the minimum number of hosts with the least migration cost.
pMapper [57]	Minimize the power and migration costs.	Server-Threshold-based (if CPU utilization of a server is higher than a threshold)	Local search 1. Choose the smallest sized VM from the servers that trigger VM migration 2. Sort the VMs to be migrated from all servers in ascending order of their size (CPU utilization) 3. Select the most energy-efficient server as destination host.
PADD [56]	Minimize energy consumption while satisfying SLAs	Server-Threshold-based (if free CPU capacity of a server is lower than a threshold)	Local search 1. VM selection: maximum demand, average demand, and minimum standard deviation of demands 2. Destination selection: not specified
Dynamic Placement [128]	Provide statistical SLA guarantee while minimizing the number of hosts	Invoke replacement periodically	Previous placement is not taking into account 1. All VMs are sorted in descending order of predicted demands (AR mode) 2. First-fit packing heuristic to place VMs to servers
Mistral [55]	Optimize total utility including application utility, power costs, and transient adaption costs	VM-Threshold-based (When a VM workload deviates from a specified workload band)	Global search in Phase I (previous placement is not taking into account) Phase I: find the min. number of nodes to host all the VMs using a modified bin-packing algorithm. Phase II: construct a search graph from current configuration and use a best-first search to reduce search space
vManage [130]	Guarantee power budget when migrating VMs	VM-Threshold-based (When a VM SLA is violated)	Local Search 1. Choose the VMs that have SLA violations 2. Select a host that satisfy both the VM's demand and power budget for a period of time
Black-box and Gray-box [59]	Eliminate hotspots (resource contention) in order to maintain VM performance	Black-box: Server-Threshold-based (if CPU/network utilization exceed a threshold) Grey-box: VM-Threshold-based (if app. SLA is violated)	Local search 1. Sort servers in decreasing order of their load. Within each server, sort the VMs in decreasing order of their load/size (memory size) 2. Choose the VMs in sorted order from the most loaded server, and try to move it to the least loaded server.

CHAPTER 6
COOPERATIVE AUTONOMIC MANAGEMENT IN DYNAMIC DISTRIBUTED SYSTEMS

The work described in last three chapters applied centralized management at global control level. However, the centralized global manager introduces a single point of failure and can become a bottleneck in handling all information and management tasks in large-scale systems. This chapter discusses our preliminary work on decentralized management in which a network of cooperative autonomic managers, each managing a subset of resources, collaborate to manage the entire system.

6.1 Problem Description

With the rapid growth of computing systems, it becomes impractical to use a centralized controller to build a large-scale self-manageable system. First, the resource overhead and time delay for collecting global system information into a centralized location increases dramatically with the size of the managed system. Second, The information and algorithms needed to decide what actions to take are too complex and application dependent to be handled by a centralized controller. Extensive research [131][132] has focused on providing autonomic capabilities to individual system components, such as databases, application servers and middleware components. In general, these autonomic components use an application-level manager that is capable of monitoring and/or predicting performance and managing resources as needed to deliver reliable applications with the expected Quality of Service (QoS). One can envision the use of these or similar components and their autonomic capabilities as the basic building blocks of large distributed systems.

Three questions arise in this context. First, what interactions should take place among individual components, in order to achieve system-level self-management? Implicit in this question is the need for information sharing among different autonomic components. Second, what type of network should be used to support the interactions? Implicit in this question is the need for the network to be highly scalable and robust

to failures. Third, how should autonomic managers be designed to interact with other components, and enhance their ability to autonomically use the many resources available in distributed systems? Implicit in this question is the need for cooperation among managers to efficiently collect and share information about resources. The answers to these three questions can be informed by results from network science. This work proposes a network-science inspired approach for distributed-system self-management arising from interactions among the autonomic components deployed in the system. The key features of the proposed design of autonomic distributed systems are the effective use of autonomic components' limited monitoring and communicating capabilities, and their interactive adaptation to the surrounding environment on the basis of information provided through an overlay network. The proposed distributed autonomic system model has the following properties:

- Self-adaptation: The system can dynamically respond to a changing environment to provide individual application managers with information and resources needed for achieving the desired QoS.

- Self-organization: The designed decentralized coordination mechanism enables the system to adapt to dynamic changes without external intervention. The global optimization is achieved through local autonomous decisions and interactions among local managers based on local information.

- Robustness: There are no central resources that could become single points of failure or performance bottlenecks. Reconfiguration mechanisms for monitoring resources and building neighboring managers effectively deal with dynamic resource availability.

6.2 Decentralized Autonomic Management Architecture

6.2.1 Generic Autonomic Element Model

We consider a highly dynamic distributed computing system consisting of a large collection of autonomic components (called Autonomic Elements) which can join and leave the system at any time. Each Autonomic Element (AE) consists of one or more managed elements (e.g. jobs and resources) and an autonomic manager (AM). The behaviors of the components are independently managed by their autonomic managers

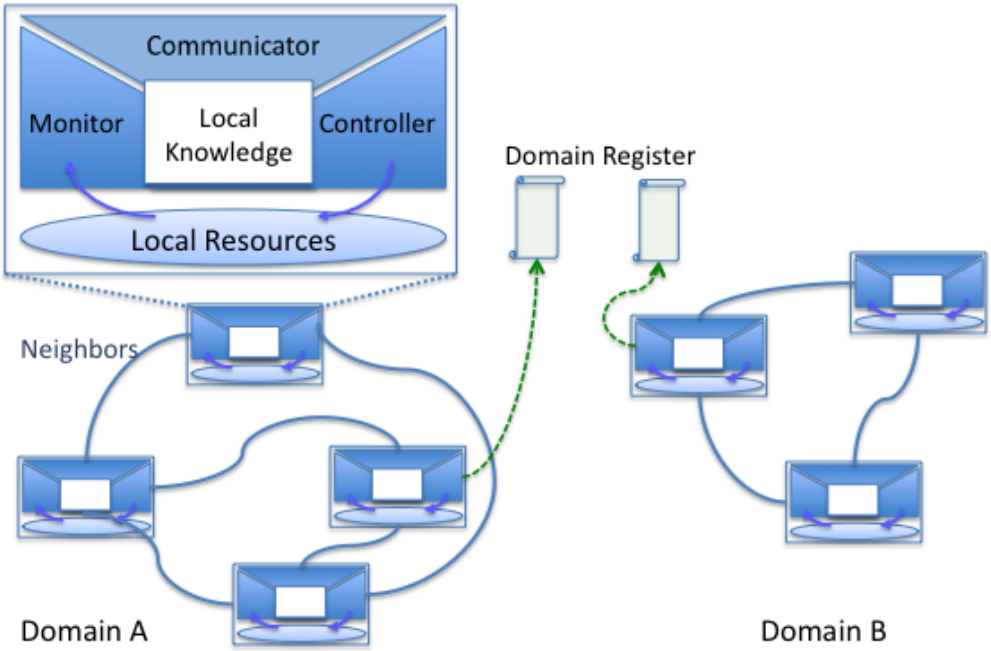

Figure 6-1. A decentralized autonomic system consisting of Autonomic Managers (AMs) across two domains, each with a registry indexing resources in the domain. Each AM contacts its domain registry to choose both the resources to monitor (called local resources) and other AMs (called neighbors) to exchange local information.

(AMs) based on monitored information. For AMs to make optimal decisions towards desired states, they require global knowledge of the changing system environment. However, in large distributed systems it is not scalable to collect and provide global knowledge through a central location.

To solve this problem, individual AMs are extended to monitor not only their managed elements but also a small piece of their surrounding environment (hereon called local resources). As explained in Section 6.2.3, local knowledge, once shared among AMs, provides the needed global knowledge to each AM. The AM architecture (Figure 6-1) consists of several components and a local knowledge base where the data shared by them are stored. The components are the following:

Monitor: it collects, aggregates and filters the status information from its managed elements and its local resources.

Controller: it manages the element' behavior based on analysis and prediction using the local knowledge.

Communicator: it supports information exchanges with other autonomic managers.

As Figure 6-1 shows, each AM only has a local view of the whole environment. However, interaction among the cooperative managers provides every AM with a global view of the system, as explained next.

6.2.2 Distributed Domain Registry

The computing resources of the system are organized into domains which may correspond to administrative domains or could conceivably be smaller in size (e.g. the computers of the ACIS Lab). A distributed domain registry infrastructure is designed to provide scalable and reliable neighbor AM discovery and resource location services for AMs. Each registry maintains an index of resources and a list of existing AMs in its domain. When an autonomic component joins the domain, its AM registers its unique ID to the registry and chooses some existing AMs to communicate with and selects some resources in the domain as its local resources.

To improve reliability, nearby domain registries periodically exchange information so that the lists of local resources and AMs stored in each registry are replicated in some places. Domain registries simplify the neighbor relationship management in autonomic systems, and provide useful domain information, such as the total number of resources and AMs present in the domain.

6.2.3 Autonomic Manager (AM) Network Building

When an AM joins a domain it selects m existing AMs in the same domain as its potential neighbors. AMs in the same neighborhood cooperate with each other by exchanging information. The neighbor selection can take place randomly, or preferentially which means that some AMs are more attractive and have a better chance to get neighbors. When departing from its domain, an AM unregisters itself by deleting its ID from the domain registry and sending a farewell message to all the neighbors to

terminate the relationship. In case an AM needs other domain's information, it can ask its domain registry for nearby registries to contact AMs in other domains. By building a "cross-domain" neighborhood, AMs can quickly get information from other domains .

Local resource claim and disclaim: Each AM randomly selects a number of resources from the domain registry and claims them as its local resources by marking the corresponding entries in the resource list with its ID. When new AMs come into the domain, they try to select the unclaimed resources in order to maximize the total number of monitored resources. An AM disclaims its resources by unmarking them in the registry before its departure. The number of resources an AM can monitor is bounded by its communication and computation capability.

Information sharing and filtering: During its lifecycle, each AM becomes a dynamic information source by updating its local resources' time-changing status information to the local knowledge base. This local information can be propagated through multi-AM cooperation, where AMs periodically exchange messages with their neighbors. Every AM that receives a message from a neighbor must store it and later forward it to its other neighbors.

Two approaches are used together to reduce the number of messages transmitted among the AMs. One is to define an obsolescence relation [133] between messages: a message m_1 is recognized as obsolete by an AM if it has message m_2 containing more recent information that subsumes m_1. As a result, m_1 is no longer needed and can be safely discarded. The other way is to evaluate each message's value [134] indicating how useful the message is, and drop the low-value messages preferentially. In the case of different AMs having distinct interests in information, it is desirable to partition the AMs into disjoint groups and then disseminate information within groups. This can be easily achieved using domain registries by adding a field in the registration list to record each AM's type of interest, which helps AMs choose their neighbors only from those of the same type.

6.2.4 AM Network Dynamism

The AM neighborhoods define the topology of a dynamic overlay network that changes as AMs continuously join and leave the system, in a manner similar to a peer-to-peer network [135]. The AMs must adapt their behaviors and interactions to the changing stateg. For example, an AM leaving or crashing may cause serious effects - claimed local resources may be no longer monitored by anyone, and some AMs may become isolated from others. To prevent and repair the damage, the following mechanisms are proposed.

AM departure and failure detection: If an AM voluntarily decides to leave, it informs all the neighbors by sending them a farewell message. In the case of AM or network failure, since neighbors periodically communicate with each other, each AM measures the interval between two successive messages sent from the same neighbor and sets a timeout to detect the failure.

Dynamic resource claim: By periodically checking the domain registry, AMs can obtain the dynamic domain information such as the number of claimed and unclaimed resources and the total number of AMs currently in the system, and then adjust the number of resources it should monitor to balance the monitoring load over the network. However, the information provided by domain registries might be incorrect because of AM's unpredictable failure. To solve this problem, once an AM detects its neighbor's failure, it informs the domain registry and reclaims the resources that became unmonitored because of the failure.

Dynamic neighborhood building: When an AM is informed of a neighbor's departure or detects a neighbor's failure, it chooses another AM as its new neighbor with probability p (set to 0.5 as explained in Section 6.3.3). This simple mechanism allows AMs to maintain network connectivity by establishing new neighborhood over the network.

6.3 Analytical Evaluation

6.3.1 Network Model

We use the conceptual framework and notations from complex random network theory [136][137] to model the system and analyze the structural organization.
The decentralized autonomic system is modeled as a network in which each AM is represented by a node, and two nodes are linked if they are neighbors. The ”degree” of a node represents the number of neighbors the node has. We use ”local load” to indicate the number of resources claimed by an AM.

Considering a set of nodes which join and leave the domain dynamically and have distinct identifiers, we use the following notations where t denotes an instant in time,

$n(t)$: the total number of nodes at time t.

$r(t)$: the total number of resources at time t.

m_i: the initial number of neighbors that the ith node connects to when joining the network.

$k_i(t)$: the degree of the ith node at time t.

$o_i(t)$: the local load of the ith node at time t.

The first two parameters describe the entire network and can be obtained directly from the domain registry, while the rest of the parameters describe the behavior of individual nodes.

6.3.2 Node Joining and Neighbor Selection

Consider the case where the network starts with one node, and then at each step, a new node joins and connects to m existing nodes. At time t the network has a total of $n(t)$ nodes ($n(t) \gg m$, for a large system). The following equations can be easily derived (see [137]).

Total number of links:

$$e(t) = n(t) \times m - \frac{m^2 + m}{2} \approx n(t) \times m \tag{6–1}$$

Average degree:

$$\overline{k}(t) = \frac{2e(t)}{n(t)} \approx 2m \tag{6–2}$$

Diameter (maximal shortest-path length between any two nodes):

$$d(t) = \frac{\ln n(t)}{\ln \overline{k}(t)} \approx \frac{\ln n(t)}{\ln 2m} \tag{6–3}$$

Equation 6–3 shows that the shortest-path length between any two nodes is small even for a large network. This ”small world effect” [138] ensures that local information of one node can be propagated to any other node very quickly even in large networks. Given the total number of nodes, Equation 6–3 provides a way to achieve any expected d by using m as the tuning parameter.

Different neighbor selection policies result in different network degree distributions. The random selection results in exponential degree distribution. In contrast, the preferential linking (the likelihood of connecting to a node is proportional to the node's degree) leads to a power-law degree distribution, also known as the Barabasi-Albert model or scale-free network [136]. The major differences between these two networks are their robustness against random network errors as discussed in the following section.

6.3.3 Node Leaving and Neighborhood Rebuilding

Since nodes may leave the network at any time, we need to examine the robustness of the network. The effect of random damage on networks was simulated in [136] and the results show that scale-free networks display a high degree of tolerance against random failures. For exponential networks, Equation 6–4 indicates that average degree decreases linearly with growing f (the fraction of removed nodes), which in turn increases network diameter (see Equation 6–3); thus it is increasingly difficult for the remaining nodes to communicate with each other.

$$\overline{k\prime} = \overline{k}(1-f) \tag{6–4}$$

$$\overline{k\prime} = \frac{2e\prime}{n\prime} \approx \frac{2(\frac{\overline{k}n}{2} - (1-p)\overline{k}fn)}{n(1-f)} \xrightarrow{p=0.5} \overline{k} \tag{6–5}$$

A dynamic neighborhood rebuilding mechanism is proposed to avoid this impact. When a node leaves the network, a fraction p of its neighbors establish new relationships with other nodes. Equation 6–5 indicates that by choosing p equal to 0.5 the average degree can remain approximately constant, so does the network diameter.

6.3.4 Local Load Adjustment

We use $\overline{o}(t) = \frac{r(t)}{n(t)}$ to express the average ratio of the number of resources to the network size at time t. To balance the load on all the nodes, when a node i joins the network, $o_i(t)$ is initialized as follows:

$$o_i(t) = \lceil \overline{o(t)} \rceil \tag{6–6}$$

The maximum number of resources each node can monitor is bounded so as to avoid overloading and also ensure that the resources' dynamic status information can be timely collected. Because the value of $o(t)$ may change as the network size and resource availability vary, each node periodically compares its current observation degree with $o(t)$ and adjusts it accordingly.

6.3.5 Communication Cost

Each node in the network sends messages to its neighbors at constant time interval. With information filtering, the message size s_i can be bound to a fixed value S. During a time unit, the global communication cost of the network is

$$C = \sum k_i(t)s_i \leq 2e(t) \cdot S \approx 2m \cdot n(t) \cdot S \tag{6–7}$$

which grows linearly with the network size. But from the perspective of a single node, the average communication cost stays almost constant.

6.4 Case Study

6.4.1 Background

In order to validate the proposed model, we used In-VIGO [3] middleware to implement a Decentralized Autonomic Virtual Application Management (DAVAM) system. In-VIGO is a grid-computing infrastructure that uses virtualization technologies to provide secure application execution environments. In this context, applications are themselves virtualized and execute in virtual machines, hence the DAVAM acronym.

Figure 6-2 provides a high-level view of the role of the autonomic Virtual Application Manager (AVAM) in In-VIGO (detailed in Chapter 3). Typically, a user initiates an application session in In-VIGO to run one or more instances of a computational tool on grid resources. Each application session is managed by a middleware component, called the Virtual Application Manager (VAM). Following the model discussed in Section 3.3, autonomic features including self-optimization and self-healing are integrated into the AVAM. It relies on monitoring of job and resource conditions, predicting violations of user- and/or system-expected execution times and restarting jobs in resources capable of delivering acceptable times.

To achieve desired performance, each AVAM requires global knowledge of the time-varying resource status information. However, the centralized approach proposed in [139] using a global controller to collect and maintain the knowledge of the whole system status does not scale well in large-scale distributed systems. Instead, a DAVAM system is constructed with distributed, cooperative AVAMs, as discussed next.

6.4.2 Cooperative AM

Figure 6-2 shows the major functions implemented in an AVAM and their information flow. The local knowledge base stores information such as dynamic local resources' status, application run-time performance, the list of the neighbors and local resources

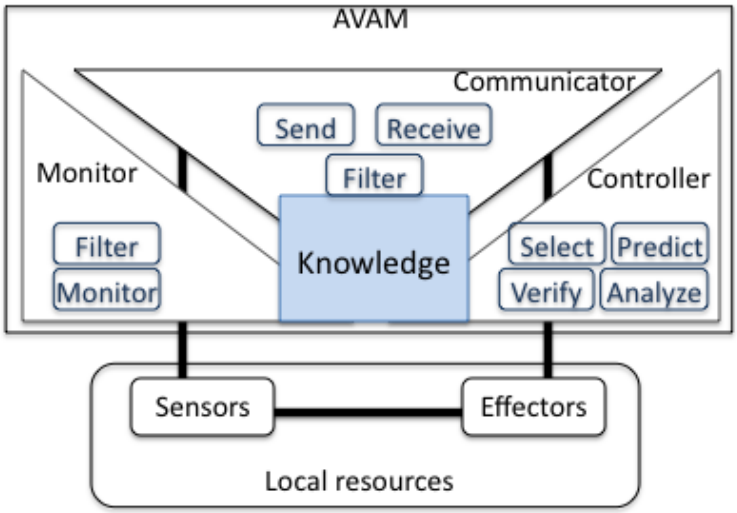

Figure 6-2. The functions and information flow of a cooperative Autonomic Virtual Application Manager.

claimed by the AVAM. A controller is designed to control the internal behavior and a communicator to manage the interactions with other AVAMs. The functions implemented in the AVAM are similar to those described in Section 3.3.2. I explain them here for your convenience.

6.4.3 Controller

The controller is responsible for controlling the application execution to achieve reliable and optimized performance. The following lists the functions implemented in the controller:

Predict: To choose the appropriate resources for the tool execution, the controller needs to know the specific resource requirements for each given job. A memory-based learning algorithm citeavam[81] is used to predict resource utilization information, such as CPU cycles, CPU utilization and memory utilization. The basic idea behind this algorithm is that the resources consumed by a particular job often depend on the input parameters supplied to the tool. Therefore, the ”similarity” of two jobs is defined by the distance metric of two sets of input values and resource consumption is predicted based on the tool execution history.

Select: The controller scans through the list of resources in the local resource table and ranks them based on the job's resource requirements and the resources' processing capacity. The score calculated as follows reflects how well a resource can host the job. The resources with zero score are not considered since they cannot satisfy the deadline specified by the user. If the resource is a virtual machine, the predicted runtime is also divided by its physical host's load.

$$predicted\_runtime_i = \begin{cases} \dfrac{job\_cycles}{CPU\_speed_i} & \text{if } job\_load \leq 1 - CPU\_load; \\ \dfrac{job\_cycles}{CPU\_speed_i} \times (1 + CPU\_load) & \text{otherwise.} \end{cases} \tag{6–8}$$

$$score_i = \begin{cases} 1 - \dfrac{predicted\_runtime_i}{deadline_i} & \text{if } predicted\_runtime_i \leq deadline_i \\ 0 & \text{otherwise} \end{cases} \tag{6–9}$$

The controller can choose different resource selection policies for different purposes. In our case, to optimize the performance of the jobs it manages, the controller strives to select the resource with the highest score. However, resource contention may happen if multiple AVAMs try to submit jobs to the same "best" resource simultaneously. A so-called ϵ-random rule is used to deal with this problem. A randomly generated small number ϵ distributed evenly in the range [-0.1, 0.1] is added to each resource's score, and then Select function ranks the resource list with these modified scores. By setting a small number ϵ, the ϵ-random rule is able to mitigate resource contention to a certain extent.

Verify: After a resource is selected, this function checks the current status of the resource and verifies whether its score is still valid. If not, the controller selects the next candidate resource in the ranked list and repeats this verification process.

Analyze: After a job is submitted to the chosen resource, the monitor keeps collecting the job's running status (e.g., current CPU time, elapsed time, and CPU utilization consumed by the job), which is used by Analyze function to estimate the job's progress and predict the finishing time. The predicted job finishing time is given by the following formula (see [139] for details).

$$finish\_time = elasped\_time + \frac{job\_cycles - finished\_cycles}{CPU\_speed} \times CPU\_load \qquad (6\text{–}10)$$

If any performance problem is detected by the Analyze function, the controller is responsible for taking appropriate actions. For example, if it is predicted that the job cannot finish before the deadline, the controller will try to find a better resource that can satisfy the job requirements and reschedules the job to that resource.

In the case when all the resources in one domain are heavily loaded and cannot satisfy job requirements, the controller takes the following steps to handle this situation: it contacts nearby domain registries to select several AVAMs in other domains as its "cross-domain" neighbors; and it communicates with these neighbors so as to quickly get the resource information in other domains and determine on which resource it can submit the job. Although inter-domain communication may be slower, only a few interactions are required in this cross-domain neighbor selection, and it can greatly speed up the job's execution in this scenario, as demonstrated by the experiment in Section 6.6.

6.4.4 Monitor and Communicator

The monitor periodically collects dynamic local resources' status information (e.g. CPU utilization, memory utilization, and average load) and updates it to the local knowledge base. In addition, the monitor checks every job listed in the knowledge base periodically. If the job finishes successfully, the monitor collects some statistic data about this execution (e.g. the application's input parameters, performance and resource

usage), and reports it to the knowledge base for historical records. If the job is still running, the monitor feeds the monitored data about the resource and the job to the controller for estimating the progress of the given job.

The Communicator is responsible for sending and receiving messages to and from neighbors. There are four types of messages exchanged between neighbors.

Joining/leaving: An AVAM sends messages to its neighbors to notify its arrival or departure. The messages carry the sender's ID so that the receiver can add/delete it to or from its neighbor list.

Local resource table: Each AVAM has its own current view of the resources' status and stores it in a local resource table. To disseminate this information, every AVAM periodically (every 10 seconds in our implementation) sends its local resource table to the neighbors.

Rewiring: Before leaving, an AVAM selects a fraction p (set to 0.5 in our case) of its neighbors and sends them rewiring messages. The receivers then choose some other AVAMs as their new neighbors.

6.4.5 Information Filtering

The resources' status information collected by an AVAM's monitor and communicator must be filtered before being added to the local resource table to reduce the message size. To realize this, each record has an age attribute to indicate the time elapsed since the last update. If two records contain the same resource's information, the older one gets filtered out.

Information filtering also happens by purging the lower-values records from the table. Concentrating on CPU-intensive applications, AVAMs are interested in resources with high CPU processing power. Thus, the value of the ith resource record is defined as follows. If CPU utilization stays below 100%, the CPU capacity is calculated by the CPU speed and utilization; otherwise, it is computed using the CPU load (the queue length

of the runnable processes). A weight of 0.01 is used to make these two measurements comparable.

$$value_i = \begin{cases} CPU\_speed_i \times (1 - CPU\_utilization_i) & \text{if } CPU\_utilization_i \leq 1 \\ \dfrac{CPU\_speed_i}{CPU\_load_i} \times 0.01 & \text{otherwise} \end{cases} \tag{6–11}$$

If a monitored resource is a virtual machine, its CPU processing power is affected by the CPU load of its physical host. In such case, the AVAM monitors the status of both the virtual machine and the physical host. The value calculated as above is reduced by a factor that is inversely proportional to the physical host's load.

Due to the dynamic characteristics of grid resources, the older a resource record becomes, the less accurate it is. Therefore, while evaluating a resource record during the table purging process, the record's value is reduced by a factor corresponding to its age, represented as $\alpha = 1 - \frac{age}{max}$, where the max is set to 60 seconds in our implementation. With this information filtering, a local resource table's size is optimized by only retaining the resources with high CPU processing capability.

6.5 Experimental Evaluation

6.5.1 Experimental Setup

The experiments were conducted on a subset of the In-VIGO system. The computer resources consist of 200 VMware-server virtual machines (each has 128 MB memory and runs Red Hat 7.3) hosted on a cluster of dual 2.4GHz hyper-threaded Xeon nodes.

A resource can be used to host a single AVAM and multiple application jobs simultaneously. In the experiments, a considerable amount of background load was also introduced into the resources by launching CPU-intensive jobs. Dynamic loading environments were created by randomly choosing and loading different subsets of the resources (100 randomly chosen resources, unless otherwise noted) every 50 seconds.

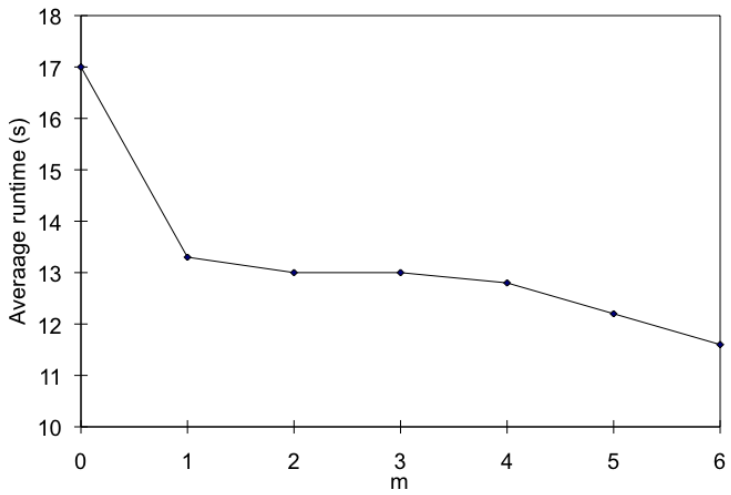

Figure 6-3. The comparison of the execution time of TunProb jobs with different number of neighbors m during 150 seconds.

The domain registries are implemented with MySQL. TunProb (Numerical Calculation of the Transmission Probability for One-Dimensional Electron Tunneling), a tool available on the In-VIGO portal, is used as an application benchmark representative of CPU-intensive workloads. In the experiments each AVAM was used to manage the execution of one or more instances of TunProb.

The DAVAM system initialization process starts with one AVAM. Then at each increment of time (one second) one new AVAM is started until the expected system size is reached. Each AVAM establishes connections with m randomly-chosen existing AVAMs in its domain. From the equation 6–2 we can see that setting m a small value (2-4) is good enough for an AVAM network whose size is smaller than 100. This is also confirmed by the following experiments. Each AVAM monitors up to five virtual machines as its local resources, and updates their status (CPU load, CPU utilization and free memory size) and their physical hosts' load in its local resource table every ten seconds. AVAM neighbors exchange their local resource tables every ten seconds and the table can only keep up to ten resource records.

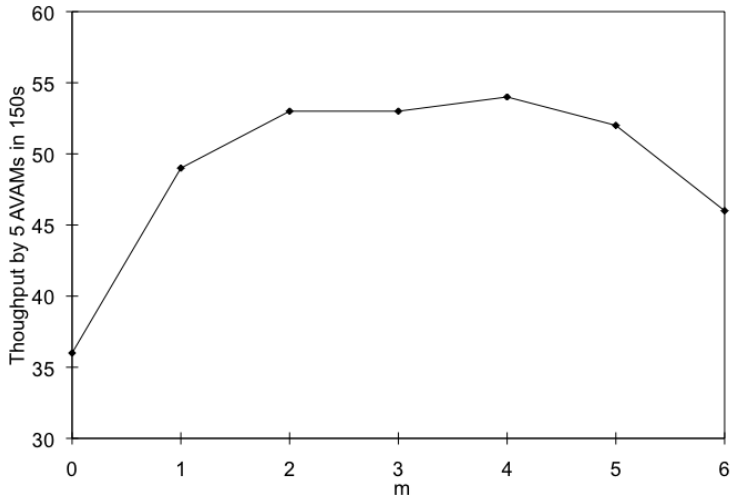

Figure 6-4. The comparison of the TunProb jobs finished by 5 AVAMs with different number of neighbors m during 150 seconds.

6.5.2 Experimental Evaluation of Efficiency

The efficiency of the DAVAM system is reflected by each AVAM being able to quickly obtain the current status of the entire system and find good resources for its jobs. The first experiment investigates how the performance changes with different numbers of neighbors each AVAM contacts when joining the domain. Fifty AVAMs were initially started in the domain, and 10 seconds later another five AVAMs joined the same domain and each selected (a value between 0 to 6) neighbors to communicate with. After ten seconds of their arrivals, the five AVAMs began to submit jobs continuously until they left the domain 140 seconds later.

Figure 6-3 and Figure 6-4 compares the average job runtime and the overall throughput (the total number of jobs completed by the 5 AVAMs) with different values of m. As expectable, the worst performance occurs when each AVAM does not have any neighbors and only knows the status of its local resources. As the value of m increases, the performance improves because AVAMs can learn more resources' information through interaction with their neighbors and select resources more wisely. Figure 6-4 also indicates that, when the value of m exceeds five, the throughput drops because the benefit from contacting more neighbors is outweighed by communication overhead.

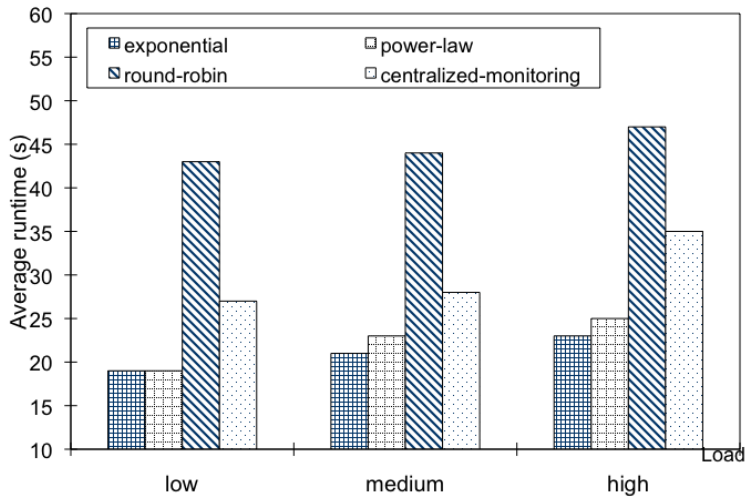

Figure 6-5. The comparison of the jobs average execution time with DAVAM and the centralized approaches in three different loading environments.

6.5.3 Experimental Evaluation of Scalability

In the second experiment, we studied the system scalability by comparing the performance of DAVAM with centralized monitoring and round-robin approaches. Forty AVAMs join the domain and each one submits jobs continuously for 150 seconds. In the DAVAM approach, each AVAM selects two neighbors. The neighbor selections, with and without preference, lead to two types of networks, power-law and exponential networks[136], respectively.

The *centralized-monitoring* approach uses a central monitor to collect and store resources' dynamic status information. Each AVAM chooses the best resource currently available in the database to submit its jobs. The round-robin approach does not need any resource status information as the AVAMs choose resources from the database in a round-robin manner. The experiments were conducted in three different loading environments: low, medium and high, in which 30%, 50% and 70% of randomly chosen resources from the domain were loaded with CPU-intensive processes, respectively.

Figure 6-5 and Figure 6-6 shows the average job runtime and the overall throughput of the different approaches. Both exponential and power-law AVAM networks deliver similar best performance because the small world property makes sure that each

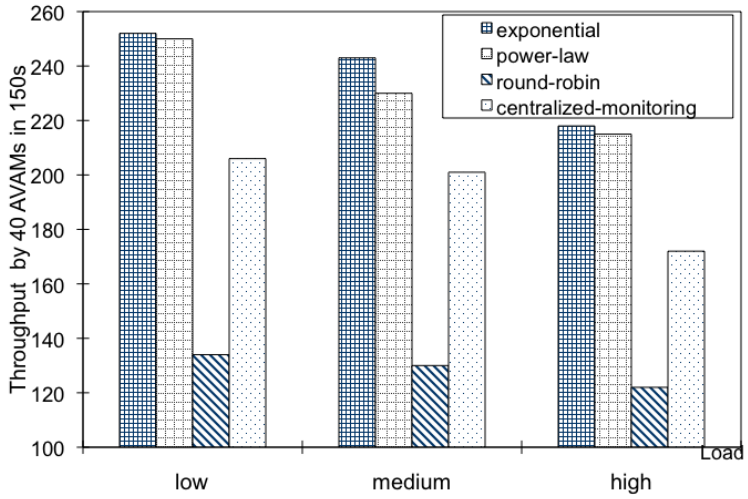

Figure 6-6. The comparison of the jobs finished by 40 AVAMs for the DAVAM and the centralized approaches in three different loading environments.

AVAM in the network can obtain the latest system-wide resource status very quickly. Furthermore, the ϵ-random resource selection mechanism avoids resource contention among multiple AVAMs. In contrast, the centralized monitoring approach suffers from database-access contention between the AVAMs and the central monitor, and hence behaves much worse than DAVAM. The round-robin approach gives the worst performance because it does not consider any dynamic status information for resource selection.

6.5.4 Experimental Evaluation of Robustness

The third experiment studies the robustness of the DAVAM approach, where the system-level information is constructed by the distributed cooperative AVAMs, in contrast with the centralized approach, where a central database is used to store the global knowledge. In the experiment, 50 AVAMs were started in a single domain at the same time. After 200 seconds, half of them left and the other half continued to work and submit jobs for another 200 seconds.

In DAVAM the remaining AVAMs react to system changes by contacting new neighbors and reclaiming resources from the domain registry. The neighborhood rebuilding mechanism maintains the DAVAM network connectivity, and the resource

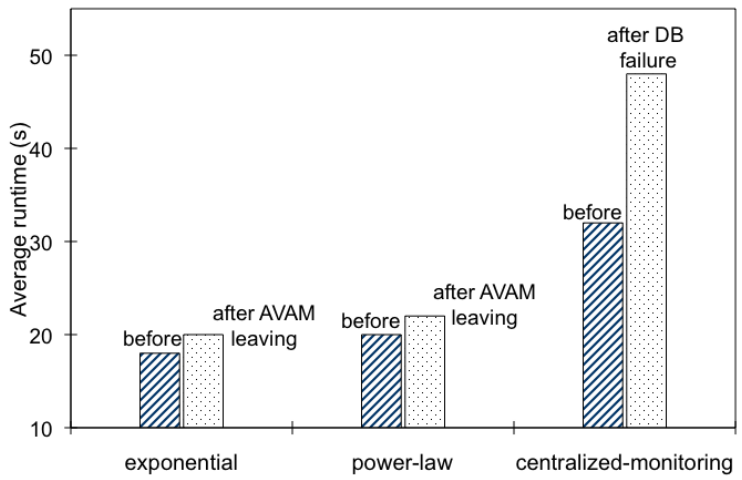

Figure 6-7. TunProb jobs average execution time before and after AVAM leaving, for DAVAM exponential and power-law networks, and before and after failure of a central database when it is used for centralized monitoring.

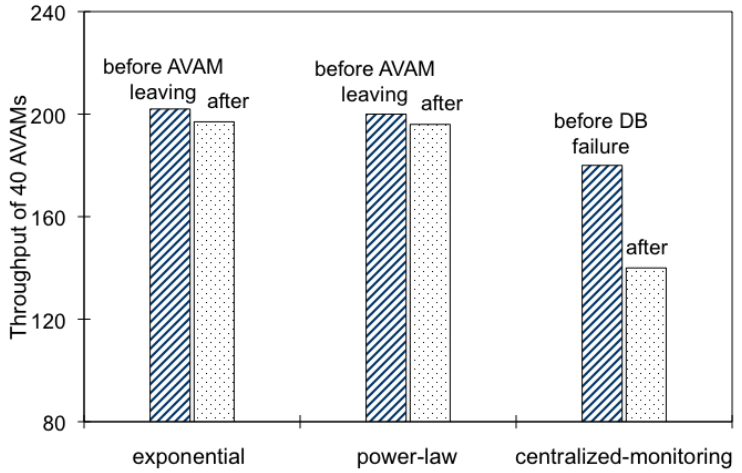

Figure 6-8. The total number of TunProb jobs finished by 25 AVAMs before and after AVAM leaving, for DAVAM exponential and power-law networks, and before and after failure of a central database when it is used for centralized monitoring.

reclaiming ensures that most of the resources are still monitored by at least one AVAM. Figure 6-7 and Figure 6-8 compares the average job runtime and the overall throughput by the 25 AVAMs before and after the other half of the AVAMs left the domain. The results show that, for both exponential and power-law networks, the performance of the remaining AVAMs is almost unaffected even if a high number of AVAMs left the system.

For the centralized monitoring approach, on the contrary, if the central database fails, none of the AVAMs can retrieve any new information from the database, so they have to continue using the resources chosen before the database failure. Figure 6 shows that, without the dynamic resource status information provided by the database, the performance drops dramatically. Similar effects can be observed if the central monitor fails.

6.6 Discussion

It may possible to design a hierarchical system to circumvent scalability issues caused by a purely centralized approach and also achieve the similar performance with the p2p approach. However, in a dynamic environment where nodes can join and leave at any time, it is very difficult to construct and maintain a balanced, optimal hierarchical structure. Moreover, the supernodes (root notes) at the top level in the hierarchical system can potentially cause single-point system failures and/or lead to isolated nodes in the system. Although replication can compensate for potential unstable behavior of a supernode, it will add resource costs and communication overhead to keep replicas consistent.

6.7 Related Work

Agent-based [140][141][142] modeling is a very natural and flexible way to model distributed interconnected systems. In [143] several distributed and self-organizing algorithms are proposed for placement of services on servers. For each service a service manager is instantiated to create multiple “ants” (agents) and send them out to the server network. The ant travels from one server to another, choosing the servers along the path based on locally available information. The ant then finally makes a decision, based on the knowledge it has accumulated on its travel. Service manager and the spawned ants work with local information, which ensures scalability. Similarly, Messor [142] proposed to use “ants” wandering over the network to explore load

conditions. The goal is to achieve load balancing by ants moving jobs from the most overloaded node to underloaded ones.

In our system, each autonomic component can be identified as an agent, and the autonomic system as a multi-agent system. Each autonomic component is both cooperative (sharing its local knowledge with neighbors) and selfish (trying to find and allocate the best resources for its own jobs). The authors in [140] claim that no obvious gain can be achieved from communication between agents. The reason is that if all the agents have a "better" picture of the whole system, they all tend to use the best resources and thus cause competition. In contrast, the resource verification and ϵ-random selection mechanisms applied to our system can prevent this problem and their effectiveness is proved by the experiments.

The peer-to-peer model offers an alternative to the traditional client-server model for many large-scale applications in distributed setting. Epidemic (or gossip) algorithms [144][134] have proved to be effective solutions for disseminating information in large-scale systems. The basic idea is that each process periodically chooses a random subset of processes in the system and sends them the new information it has received. One of the issues underlying the deployment of traditional epidemic algorithms is that they rely on each process having knowledge of the global membership which is not realistic for large groups of processes. The reliable and efficient information dissemination in our system is inherited from the simple neighbor relationship establishment over the system and dynamic neighborhood rebuilding mechanism. The membership protocol is very simple to deploy with support from the decentralized domain registry service.

6.8 Conclusions

This chapter presents an autonomic computing system in which multiple autonomic components collaborate to optimize the behavior of the system. A general autonomic manager model is designed to control the managed elements' internal state and

manage its interactions with the surrounding environment. The autonomic manager is lightweight, making it suitable for many distributed systems. Each has a local view of the system state and communicates periodically its partial knowledge to its neighbors, thus contributing to building a common, shared global view of the system state. A decentralized registry provides scalable and reliable neighbor and resource discovery service for the system. The overlay network structured by the neighbor relationships is demonstrated to be highly reliable and efficient. The results show that the decentralized and cooperative nature of the system yields a number of desirable properties, including efficiency, robustness, and scalability under a highly dynamic environment. The design decisions were motivated by results from network science which also provided a basis for the analytical and experimental evaluation of the described distributed autonomic management system. Future advances in network science and autonomic computing will benefit each other, to manage complex IT-enabled systems and to address new management challenges of networked of physical and social entities, respectively.

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

The main objective of this dissertation is to achieve autonomic application and resource management in virtualized distributed computing systems, targeting grids and data centers. Such environments are typically running a large variety of applications with different resource requirements and dynamic workloads, which pose great challenges for providing desired properties such as performance guarantees, efficient use of resources, and reduced operational and management costs. A novel design of two-level control system is proposed to reduce the management complexity and allow independent and flexible optimization and adaptation. By incorporating methods and techniques from machine learning, optimization, and control theory, the proposed control systems achieve to increase the ability to adapt to an unpredictable environment and manage to maximize application performance and minimize associated costs.

In the context of grid environments, the main contribution of our work is to support workloads that require a specified quality of service on grid resources with varying loads and unpredicted availability. An implementation of the two-level control system is deployed in the In-VIGO system, a grid system for scientific applications and extensively utilizes virtualization technologies. A local controller activated for each application session applies a memory-based learning approach to predict its job's resource needs and allocates the proper resources for the job using the resource status information provided by the global controller. The local controller also monitors jobs' performance and determines control actions such as rescheduling a job to better resources when the predicted job performance cannot meet users' requirements. The autonomic application management system is tested for the applications that generate CPU-intensive jobs with short execution times. In the scenarios considered in this work, the results confirm the following points: (1)The online learning algorithm is fast and the prediction error is low (below 10%); (2)The two-level control system can quickly respond to the dynamic

changes of resource status and improve the execution performance significantly compared to the approach without using resource information; (3)With the prompt rescheduling control action, the control system can effectively recover from performance faults and execution failures.

The second part of our work focuses on addressing challenges in automating resource management in a virtualized data center environment. Compared to a grid environment, the resources in data centers are typically more reliable and located at a centralized location. However, the applications hosted in data centers usually have highly dynamic workloads that are hard to predict. In addition, the relationship between the workloads and their resource needs are more complex and most of time cannot be described using linear models. The fuzzy-logic-based approaches proposed in this dissertation provide a generic solution to learning the relationship without making underlying assumptions on workload characteristics or system behaviors. The experimental evaluation shows that it can efficiently model the nonlinear relationship between workloads and resource needs with dynamically changing operating conditions very fast. The resource cost consumed by each virtual machine and its hosted application is significantly reduced while the application performance is still guaranteed.

Besides optimizing resource use to reduce costs, several other aspects of data centers such as power consumption and hotspots issues become increasingly important. In determining virtual machine placement to physical hosts, a multi-objective optimization solution is proposed for simultaneously optimizing several possibly conflicting objectives. A fuzzy-logic-based evaluation approach is used to conveniently combine different objectives and an improved genetic algorithm is applied for searching the placement solution space globally. Confirmed by the simulation-based experiments, the proposed approach seeks good balance among different objectives and shows superior performance compared with well-known bin-packing algorithms and single-objective approaches. Dynamic virtual machine migration is used to adapt to changes of system

conditions and application workloads. The information across both the virtualization layer and the physical resource layer are utilized in determining when, which and where to move virtual machines. A two-level detection is applied in the controller for fast detection while avoiding unnecessary migrations. The multi-objective optimization approach is also applied in selecting hosts for migrated virtual machines. System stabilization is considered in the design of the control system. The experiments conducted on an IBM BladeCenter confirmed that the applied stabilization technique significantly reduces the number of virtual machine migrations. In addition, the multi-objective selection performs better than single-objective with respect to power consumption, thermal distribution, resource usage, and migration overhead in the scenarios considered in this work.

The above summarized the major accomplishments in this dissertation, however, there are several limitations in this work that are worth discussing.

1. In the work on autonomic application management in grid systems, the local controller monitors and predicts the progress of its jobs running on a grid resource. If the job is predicted to exceed the deadlines, the controller stops the current job and restarts it on a better resource where the job is expected to finish sooner. This stop-restart job rescheduling mechanism works well in the case that the performance fault is detected in the early stage of a job's execution or the execution time of the job is very small (e.g., less than tens of minutes). If a job has been executed for a long time (e.g., hours or even days) before it is stopped by the controller, all the work that has been done is lost. An alternative way is to save the current job's status (i.e., checkpointing) and migrate the job. Compared to process migration, virtual machine migration provides a way of conveniently moving workloads without modifying applications. However, because of the considerable migration overhead, the controller must carefully choose between the stop-restart and suspend-restart rescheduling approaches.

2. In the two-level resource control of virtualized data centers, local controllers predict resource needs of their virtual machines using fuzzy modeling or fuzzy prediction approaches and the global controller allocates resources to virtual machines based on the estimation. If the estimated resource demands are smaller than actual needs, the application's performance will suffer due to the resource scarcity. On the other hand, if the estimated value is much larger than the actual needs, the resources are wasted so that the data center's profit is reduced. Therefore, the prediction accuracy is expected to have an impact on the system performance,

which is not fully explored in this dissertation (the current work uses a simple correction action which allocates a predefined percentage of resources in case of performance degradation).

3. In the work of virtual machine placement, the thermal management is simplified by considering only CPU temperature and the power consumption only considers the power consumed by the CPUs on the blade nodes. However, there is a considerable amount of power consumed by CPU fan and cooling facilities in a data center, which is also affected by workload intensity and distribution. Beside CPU temperature which is largely determined by CPU activity, the thermal distribution of a data center is highly affected by heat recirculation, a common phenomenon in data centers.

4. The centralized management at the second level of the proposed control system would become bottleneck for extremely large-scale data centers such as cloud computing data centers. Chapter 6 investigated decentralized and cooperative resource management in the context of the In-VIGO grid system. However, this approach would not perform well in a data center environment because of the following issues: 1) How to resolve the conflicts among different local controllers. For example, the same server may be chosen as destination by multiple controllers to migrate their virtual machines and become overloaded. 2) How to deal with the delay of information sharing among local controllers. The accuracy and promptness of resource information are critical to data center environments. 3) Random neighbor selection probably won't be the optimal way for constructing a management network. The position of the servers will affect the system performance due to network latency.

Each of the above limitations is a potential topic to be investigated in the future work. In addition, with the idea of Infrastructure-as-a-service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS), cloud computing emerges as a mainstream platform. Cloud computing has its connection to grid computing and virtualized data centers but with its distinct characteristics, which makes it quite challenging to extend our work proposed in this dissertation to clouds. The following is the definition of cloud computing from Ian Foster, "A large-scale distributed computing paradigm that is 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."[145]. Different from grid computing which provides an infrastructure that delivers storage and compute resources, cloud computing shifts to

the one that is economy based, aiming to deliver more abstract resources and services (usually through virtualization). Compared with virtualized data center environment considered in this work, cloud computing provides different levels of services, which makes monitoring more challenging. Users are only exposed to high-level servers and do not have the detailed information about the resource status. The same problems potentially exist for Cloud developers and administrators, as the abstract/unified resources usually go through virtualization and some other level of encapsulation, and tracking the issues down the software/hardware stack might be more difficult. Network latency and bandwidth may also raise issues in cloud computing since users access their services through wide area networks. To ensure a good level of QoS delivered to the end users will be one of the major challenges for cloud computing.

REFERENCES

[1] "Ibm solutions grid for business partners." URL:http://joung.im.ntu.edu.tw/teaching/distributed_systems/documents/IBM_grid_wp.pdf.

[2] N. H. Kapadia and J. A. B. Fortes, "Punch: An architecture for web-enabled wide-area network-computing," *Cluster Computing*, vol. 2, no. 2, pp. 153–164, 1999.

[3] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, "From virtualized resources to virtual computing grids: the in-vigo system," *Future Generation Computer Systems*, vol. 21, pp. 896–909, June 2005.

[4] "How do you define the data center." URL:http://www.networkworld.com/newsletters/datacenter/2006/0206datacenter1.html?page=1.

[5] "How streaming the os and applications changes the rules for server provisioning." URL:www.dell.com/downloads/global/power/ps3q07-20070601-Citrix.pdf.

[6] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 303–314, 2005.

[7] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 103–116, 2001.

[8] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *SIGARCH Comput. Archit. News*, vol. 35, pp. 13–23, June 2007.

[9] Q. Tang, S. K. S. Gupta, and G. Varsamopoulos, "Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1458–1472, 2008.

[10] "Estimating total power consumption in servers in the u.s. and the world." URL:http://www.networkworld.com/newsletters/datacenter/2006/0206datacenter1.html?page=1.

[11] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the art of virtualization," in *Proc. the ACM Symposium on Operating Systems Principles*, October 2003.

[12] "Vmware: Server and datacenter virtualization products." URL:http://www.vmware.com/products/datacenter-virtualization.html.

[13] "Autonomic computing at ibm." URL:http://www.research.ibm.com/autonomic/.

[14] "autonomic nervous system." URL:http://en.wikipedia.org/wiki/Autonomic_nervous_system.

[15] "Autonomic computing blueprint." URL:http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf.

[16] Y. Diao, J. L. Hellerstein, and S. Parekh, "Using fuzzy control to maximize profits in service level management," *IBM Systems Journal*, vol. 41, 2002.

[17] M. Sugeno and T. Yasukawa, "A fuzzy-logic-based approach to qualitative modeling," *IEEE Transactions on Fuzzy Systems*, vol. 1, 1993.

[18] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "On the use of fuzzy modeling in virtualized data center management," in *Proceedings of the Fourth International Conference on Autonomic Computing*, (Washington, DC, USA), pp. 25–, IEEE Computer Society, 2007.

[19] M. Sakawa, *Genetic Algorithms and Fuzzy Multiobjective Optimization*. Norwell, MA, USA: Kluwer Academic Publishers, 2001.

[20] J. P. Jones, B. Nitzberg, and B. Henderson, "Workload management: More than just job scheduling," *Cluster Computing, IEEE International Conference on*, vol. 0, p. 113, 2001.

[21] Y. Etsion and D. Tsafrir, "A short survey of commercial cluster batch schedulers," tech. rep., Hebrew University, 2005.

[22] "Maui cluster scheuler." URL: http://www.clusterresources.com/products/maui/.

[23] "Loadleveler cluster scheuler." URL: http://www-03.ibm.com/systems/clusters/software/loadleveler/index.html.

[24] A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten, "Portable batch system: External reference specification," tech. rep., MRJ Technology Solutions, November 1999.

[25] "Lsf cluster scheduler." URL: http://www.platform.com/workload-management/high-performance-computing/lp.

[26] J. P. Jones and B. Nitzberg, "Scheduling for parallel supercomputing: A historical perspective of achievable utilization," in *Proceedings of the Job Scheduling Strategies for Parallel Processing*, IPPS/SPDP '99/JSSPP '99, (London, UK), pp. 1–16, Springer-Verlag, 1999.

[27] D. Feitelson and A. Weil, "Utilization and predictability in scheduling the ibm sp2 with backfilling," *Parallel Processing Symposium, International*, vol. 0, p. 0542, 1998.

[28] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, pp. 529–543, 2001.

[29] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," Tech. Rep. UCB/EECS-2009-55, EECS Department, University of California, Berkeley, Apr 2009.

[30] D. Zotkin and P. J. Keleher, "Job-length estimation and performance in backfilling schedulers," in *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, HPDC '99, (Washington, DC, USA), IEEE Computer Society, 1999.

[31] W. Tang, N. Desai, D. Buettner, and Z. Lan, "Analyzing and adjusting user runtime estimates to improve job scheduling on the blue gene/p," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pp. 1 –11, 2010.

[32] S.-H. Chiang, A. C. Arpaci-Dusseau, and M. K. Vernon, "The impact of more accurate requested runtimes on production job scheduling performance," in *Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '02, (London, UK), pp. 103–127, Springer-Verlag, 2002.

[33] W. A. Ward, Jr., C. L. Mahood, and J. E. West, "Scheduling jobs on parallel systems using a relaxed backfill strategy," in *Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '02, (London, UK), pp. 88–102, Springer-Verlag, 2002.

[34] S.-H. Chiang, A. C. Arpaci-Dusseau, and M. K. Vernon, "The impact of more accurate requested runtimes on production job scheduling performance," in *Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '02, (London, UK), pp. 103–127, Springer-Verlag, 2002.

[35] Y. Zhang, M. S. Squillante, and R. K. Sahoo, "Performance implications of failures in large-scale cluster scheduling," in *In 10th Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 233–252, 2004.

[36] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam, "Fault-aware job scheduling for bluegene/l systems," in *In IEEE IPDPS, Intl. Parallel and Distributed Processing Symposium*, pp. 64–73, 2004.

[37] W. Tang, Z. Lan, N. Desai, and D. Buettner, "Fault-aware, utility-based job scheduling on blue, gene/p systems," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pp. 1 –10, 2009.

[38] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov, "Adaptive computing on the grid using apples," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 4, pp. 369–382, 2003.

[39] R. Wolski, "Dynamically forecasting network performance using the network weather service," *Cluster Computing*, vol. 1, no. 1, pp. 119–132, 1998.

[40] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor - a hunter of idle workstations," in *Proc. 8th International Conference on Distributed Computing Systems*, (San Jose, CA, USA), pp. 104–111, June 1988.

[41] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny, "Checkpoint and migration of unix processes in the condor distributed processing system," tech. rep., University of Wisconsin, Madison, 1997.

[42] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-crummey, D. Reed, L. Torczon, and R. Wolski, "The grads project: Software support for high-level grid application development," *International Journal of High Performance Computing Applications*, vol. 15, pp. 327–344, 2001.

[43] R. L. Ribler, H. Simitci, and D. A. Reed, "The autopilot performance-directed adaptive control system," *Future Gener. Comput. Syst.*, vol. 18, pp. 175–187, September 2001.

[44] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, C. Koelbel, B. Liu, X. Liu, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, C. Mendes, A. Olugbile, J. M. Patel, D. Reed, Z. Shi, O. Sievert, H. Xia, and A. YarKhan, "New grid scheduling and rescheduling methods in the grads project," *Int. J. Parallel Program.*, vol. 33, pp. 209–229, June 2005.

[45] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Softw. Pract. Exper.*, vol. 32, pp. 135–164, February 2002.

[46] K. Keahey, I. Foster, T. Freeman, and X. Zhang, "Virtual workspaces: Achieving quality of service and quality of life on the grid," *Scientific Programming*, vol. 13, 2005.

[47] S. Santhanam, P. Elango, A. Arpaci-Dusseau, and M. Livny, "Deploying virtual machines as sandboxes for the grid," in *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*, (Berkeley, CA, USA), pp. 7–12, USENIX Association, 2005.

[48] M. A. Vouk, A. Rindos, S. F. Averitt, J. Bass, M. Bugaev, A. Kurth, A. Peeler, H. E. Schaffer, E. D. Sills, S. Stein, J. Thompson, and M. Valenzisi, "Using vcl technology to implement distributed reconfigurable data centers and computational services for educational institutions," *IBM J. Res. Dev.*, vol. 53, pp. 509–526, July 2009.

[49] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating system support for planetary-scale network services," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, (Berkeley, CA, USA), pp. 19–19, USENIX Association, 2004.

[50] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," in *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, (New York, NY, USA), pp. 289–302, ACM, 2007.

[51] X. Wang, D. Lan, G. Wang, X. Fang, M. Ye, Y. Chen, and Q. Wang, "Appliance-based autonomic provisioning framework for virtualized outsourcing data center," *Autonomic Computing, International Conference on*, vol. 0, p. 29, 2007.

[52] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," in *ICAC '08: Proceedings of the 2008 International Conference on Autonomic Computing*, (Washington, DC, USA), pp. 3–12, IEEE Computer Society, 2008.

[53] E. Kalyvianaki and T. Charalambous, "On dynamic resource provisioning for consolidated servers in virtualized data centers," in *the 8th International Workshop on Performability Modeling of Computer and Communication Systems*, 2007.

[54] "Xenserver." URL:http://www.citrix.com/English/ps2/products/product.asp?contentID=683148.

[55] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *ICDCS '10: Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems*, (Washington, DC, USA), pp. 62–73, IEEE Computer Society, 2010.

[56] M. Y. Lim, F. Rawson, T. Bletsch, and V. W. Freeh, "Padd: Power aware domain distribution," in *ICDCS '09: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, (Washington, DC, USA), pp. 239–247, IEEE Computer Society, 2009.

[57] A. Verma, P. Ahuja, and A. Neogi, “pmapper: power and migration cost aware application placement in virtualized systems,” in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware ’08, (New York, NY, USA), pp. 243–264, Springer-Verlag New York, Inc., 2008.

[58] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, “Entropy: a consolidation manager for clusters,” in *VEE ’09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, (New York, NY, USA), pp. 41–50, ACM, 2009.

[59] A. V. Timothy Wood, Prashant Shenoy and M. Yousif., “Black-box and gray-box strategies for virtual machine migration,” in *Proceedings of the Fourth Symposium on Networked Systems Design and Implementation*, 2007.

[60] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu, “Feedback performance control in software services,” *IEEE Control Systems Magazine*, vol. 23, p. 2003, 2003.

[61] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, “Autonomic resource management in virtualized data centers using fuzzy logic-based approaches,” *Cluster Computing*, vol. 11, pp. 213–227, September 2008.

[62] Z. W. X. Z. S. Singhal, “Utilization and slo-based control for dynamic sizing of resource partitions,” *Lecture Notes in Computer Science*, 2005.

[63] X. Zhu, Z. Wang, and S. Singhal, “Utility-driven workload management using nested control design,” in *Control Design, American Control Conference*, 2006.

[64] W. Xu, X. Zhu, S. Singhal, and Z. Wan, “Predictive control for dynamic resource allocation in enterprise data centers,” in *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, 2006.

[65] Y. Wang, X. Wang, M. Chen, and X. Zhu, “Partic: Power-aware response time control for virtualized web servers,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, pp. 323 –336, feb 2011.

[66] X. Wang, M. Chen, C. Lefurgy, and T. Keller, “Ship: Scalable hierarchical power control for large-scale data centers,” in *Parallel Architectures and Compilation Techniques, 2009. PACT ’09. 18th International Conference on*, pp. 91 –100, sept 2009.

[67] L. Zhang and D. Ardagna, “Sla based profit optimization in autonomic computing systems,” in *ICSOC ’04: Proceedings of the 2nd international conference on Service oriented computing*, (New York, NY, USA), pp. 173–182, ACM, 2004.

[68] X. Zhu and S. Singhal, “Optimal resource assignment in internet data centers,” in *MASCOTS ’01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, (Washington, DC, USA), p. 61, IEEE Computer Society, 2001.

[69] A. Chandra, W. Gong, and P. Shenoy, “Dynamic resource allocation for shared data centers using online measurements,” in *SIGMETRICS ’03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, (New York, NY, USA), pp. 300–301, ACM, 2003.

[70] P. S. A. Chandra, P. Goyal, “Quantifying the benefits of resource multiplexing in on-demand data centers,” in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001.

[71] M. N. Bennani and D. A. Menasce, “Resource allocation for autonomic data centers using analytic performance models,” in *ICAC ’05: Proceedings of the Second International Conference on Automatic Computing*, (Washington, DC, USA), pp. 229–240, IEEE Computer Society, 2005.

[72] G. Tesauro and J. O. Kephart, “Utility functions in autonomic systems,” in *ICAC ’04: Proceedings of the First International Conference on Autonomic Computing*, (Washington, DC, USA), pp. 70–77, IEEE Computer Society, 2004.

[73] J. O. Kephart, H. Chan, R. Das, D. W. Levine, G. Tesauro, F. Rawson, and C. Lefurgy, “Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs,” in *ICAC ’07: Proceedings of the Fourth International Conference on Autonomic Computing*, (Washington, DC, USA), p. 24, IEEE Computer Society, 2007.

[74] R. Das, J. O. Kephart, C. Lefurgy, G. Tesauro, D. W. Levine, and H. Chan, “Autonomic multi-agent management of power and performance in data centers,” in *AAMAS ’08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, (Richland, SC), pp. 107–114, International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[75] E. M. Elnozahy, M. Kistler, and R. Rajamony, “Energy-efficient server clusters,” in *In Proceedings of the 2nd Workshop on Power-Aware Computing Systems*, pp. 179–196, 2002.

[76] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, *Dynamic cluster reconfiguration for power and performance*, pp. 75–93. Norwell, MA, USA: Kluwer Academic Publishers, 2003.

[77] I. Foster, C. Kesselman, and S. Tuecke, “The anatomy of the grid: Enabling scalable virtual organizations,” *Int. J. High Perform. Comput. Appl.*, vol. 15, pp. 200–222, August 2001.

[78] M. Livny and D. Thain, “Caveat emptor: Making grid services dependable from the client side,” in *Proc. 2002 Pacific Rim International Symposium on Dependable Computing (PRDC 02)*, pp. 117–118, 2002.

[79] D. Thain and M. Livny, “Building reliable clients and servers,” in *The Grid: Blueprint for a New Computing Infrastructure* (I. Foster and C. Kesselman, eds.), pp. 224–244, New York, NY: Morgan Kaufmann, 2003.

[80] I. Chung and J. Hollingsworth, “Using information from prior runs to improve automated tuning systems,” in *Proc. the ACM/IEEE SC2004 Conference*, pp. 261–278, 1999.

[81] N. Kapadia, J. Fortes, and C. Brodley, “Predictive application-performance modeling in a computational grid environment,” in *Proc. IEEE International Symposium on High Performance Distributed Computing (HPDC 1999).*, 1999.

[82] P. Dinda, “The statistical properties of host load,” *ACM Trans. Comput. Syst.*, vol. 20, no. 1, pp. 25–48, 2002.

[83] W. Pfeiffer and N. J. Wright, “Modeling and predicting application performance on parallel computers using hpc challenge benchmarks,” in *22nd IEEE International Parallel and Distributed Processing Symposium, Hyatt Regency Hotel, Miami, FL, 2008*, 2008.

[84] A.Snavely, X.Gao, C.Lee, L.Carrington, N.Wolter, J.Labarta, J.Gimenez, and P.Jones, “Performance modeling of hpc applications,” in *Parallel Computing*, 2003.

[85] P. A. Dinda, “The statistical properties of host load,” *Sci. Program.*, vol. 7, no. 3-4, pp. 211–229, 1999.

[86] M. V. Devarakonda and R. K. Iyer, “Predictability of process resource usage: A measurement-based study on unix,” *IEEE Trans. Softw. Eng.*, vol. 15, no. 12, pp. 1579–1586, 1989.

[87] J. Albrecht, D. Patterson, and A. Vahdat, “Distributed resource discovery on planetlab with sword,” in *In WORLDS*, 2004.

[88] M. Livny and M. Solomon, “Matchmaking: Distributed resource management for high throughput computing,” in *In Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pp. 28–31, 1998.

[89] S. S. Vadhiyar and J. J. Dongarra, “A performance oriented migration framework for the grid,” in *CCGRID ’03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, (Washington, DC, USA), p. 130, IEEE Computer Society, 2003.

[90] G. Venkitachalam and B. Lim, “Virtualizing i/o devices on vmware workstation’s hosted virtual machine monitor,” in *USENIX Annual Technical Conference*, 2001.

[91] J. Dike, “A user-mode port of the linux kernel,” in *Proc. 4th Annual Linux Showcase and Conference*, 2000.

[92] L. Zadeh, “Fuzzy sets,” in *Information and Control*, pp. 338–353, 1965.

[93] S. Chiu, “Fuzzy model identification based on cluster estimation,” *Journal of Intelligent and Fuzzy Systems*, vol. 2, no. 3, pp. 267–278, 1994.

[94] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[95] Sun Inc., *Java Petstore*.

[96] D. Mosberger and T. Jin, “httperf—a tool for measuring web server performance,” *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, 1998.

[97] “1998 world cup web site access logs.” URL: ita.ee.lbl.gov/html/contrib/WorldCup.html.

[98] *Real-time Web Log Replayer*.

[99] “Hp-ux workload manager.” URL:http://docs.hp.com/en/5990-8153/ch05s12.html.

[100] J. Rolia, L. Cherkasova, and C. McCarthy, “Configuring workload manager control parameters for resource pools,” in *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pp. 127–137, 2006.

[101] T. Abdelzaher, K. G. Shin, and N. Bhatti, “Performance guarantees for web server end-systems: A control-theoretical approach,” *IEEE Transactions on Parallel and Distributed Systems*, 2002.

[102] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. M. Vahdat, “Model-based resource provisioning in a web service utility,” in *USITS’03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, (Berkeley, CA, USA), pp. 5–5, USENIX Association, 2003.

[103] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, “An analytical model for multi-tier internet services and its applications,” in *SIGMETRICS ’05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, (New York, NY, USA), pp. 291–302, ACM, 2005.

[104] X. Liu, X. Zhu, S. Singhal, and M. F. Arlitt, “Adaptive entitlement control of resource containers on shared servers,” in *Integrated Network Management*, pp. 163–176, IEEE, 2005.

[105] G. Tesauro, “Online resource allocation using decompositional reinforcement learning,” in *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference* (M. M. Veloso and S. Kambhampati, eds.), pp. 886–891, AAAI Press / The MIT Press, 2005.

[106] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, “On the use of hybrid reinforcement learning for autonomic resource allocation,” *Cluster Computing*, vol. 10, no. 3, pp. 287–299, 2007.

[107] D. Gmach, J. Rolia, and L. Cherkasova, “Satisfying service level objectices in a self-managing resource pool,” in *Proceedings of the 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO ’09, (Washington, DC, USA), pp. 243–253, IEEE Computer Society, 2009.

[108] B. Urgaonkar and A. Chandra, “Dynamic provisioning of multi-tier internet applications,” in *Proceedings of the Second International Conference on Automatic Computing*, (Washington, DC, USA), pp. 217–228, IEEE Computer Society, 2005.

[109] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, “No ”power” struggles: coordinated multi-level power management for the data center,” *SIGARCH Comput. Archit. News*, vol. 36, pp. 48–59, March 2008.

[110] A. Krum and F. Kreith, *The CRC Handbook of Thermal Engineering*. CRC Press, 2000.

[111] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, “Energy management for commercial servers,” *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[112] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, “Making scheduling ”cool”: temperature-aware workload placement in data centers,” in *ATEC ’05: Proceedings of the annual conference on USENIX Annual Technical Conference*, (Berkeley, CA, USA), pp. 5–5, USENIX Association, 2005.

[113] Y.-W. B. Chia-Hung Lien and M.-B. Lin, “Estimation by software for the power consumption of streaming-media servers,” *IEEE Trans. on Instrumentation and Measurement*, vol. 56, no. 5, 2007.

[114] K. Kant and J. Alexander, “Proactive vs. reactive idle power control,” in *Proceeding of Design and Test Technology Conference*, 2008.

[115] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[116] E. Falkenauer and A. Delchambre, “A genetic algorithm for bin packing and line balancing,” in *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pp. 1186 –1192 vol.2, may. 1992.

[117] R. R. Yager, “On ordered weighted averaging aggregation operators in multicriteria decisionmaking,” *IEEE Trans. Syst. Man Cybern.*, vol. 18, no. 1, pp. 183–190, 1988.

[118] L. Ramos and R. Bianchini, “C-oracle: Predictive thermal management for data centers,” in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pp. 111 –122, feb. 2008.

[119] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin packing: a survey*, pp. 46–93. Boston, MA, USA: PWS Publishing Co., 1997.

[120] B. Urgaonkar, A. Rosenberg, and P. Shenoy, “Application placement on a cluster of servers,” *International Journal on Foundations of Computer Science*, vol. 18, no. 5, 2007.

[121] M. Cardosa, M. R. Korupolu, and A. Singh, “Shares and utilities based power consolidation in virtualized server environments,” in *IM’09: Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management*, (Piscataway, NJ, USA), pp. 327–334, IEEE Press, 2009.

[122] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, “Server workload analysis for power minimization using consolidation,” in *USENIX’09: Proceedings of the 2009 conference on USENIX Annual technical conference*, (Berkeley, CA, USA), pp. 28–28, USENIX Association, 2009.

[123] K. Le, R. Bianchini, M. Martonosi, and T. Nguyen, “Cost- and energy-aware load distribution across data centers,” in *Proceedings of the Workshop on Power-Aware Computing and Systems*, 2009.

[124] R. K. Sharma, C. E. Bash, C. D. Patel, R. J. Friedrich, and J. S. Chase, “Balance of power: Dynamic thermal management for internet data centers,” *IEEE Internet Computing*, vol. 9, no. 1, pp. 42–49, 2005.

[125] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, “Load balancing and unbalancing for power and performance in cluster-based systems,” in *In Proceedings of the International Workshop on Compilers and Operating Systems for Low Power*, 2001.

[126] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu, “Delivering energy proportionality with non energy-proportional systems: optimizing the ensemble,” in *HotPower’08: Proceedings of the 2008 conference on Power aware computing and systems*, (Berkeley, CA, USA), pp. 2–2, USENIX Association, 2008.

[127] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu, “Power-aware qos management in web servers,” in *RTSS ’03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, (Washington, DC, USA), p. 63, IEEE Computer Society, 2003.

[128] N. Bobroff, A. Kochut, and K. Beaty, “Dynamic placement of virtual machines for managing sla violations.,” in *Integrated Network Management*, pp. 119–128, IEEE, 2007.

[129] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, “Dynamic placement for clustered web applications,” in *WWW ’06: Proceedings of the 15th international conference on World Wide Web*, (New York, NY, USA), pp. 595–604, ACM, 2006.

[130] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan, “vmanage: loosely coupled platform and virtualization management in data centers,” in *ICAC ’09: Proceedings of the 6th international conference on Autonomic computing*, (New York, NY, USA), pp. 127–136, ACM, 2009.

[131] H. Liu, “A component-based programming model for autonomic applications,” in *ICAC ’04: Proceedings of the First International Conference on Autonomic Computing*, (Washington, DC, USA), pp. 10–17, IEEE Computer Society, 2004.

[132] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, A. Segal, and J. O. Kephart, “Autonomic computing: Architectural approach and prototype,” *Integr. Comput.-Aided Eng.*, vol. 13, no. 2, pp. 173–188, 2006.

[133] J. Pereira, L. Rodrigues, and R. Oliveira, “Semantically reliable multicast: Definition, implementation, and performance evaluation,” *IEEE Trans. Comput.*, vol. 52, no. 2, pp. 150–165, 2003.

[134] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulieacute, “Epidemic information dissemination in distributed systems,” *Computer*, vol. 37, no. 5, pp. 60–67, 2004.

[135] R. Steinmetz and K. Wehrle, *Peer-to-Peer Systems and Applications (Lecture Notes in Computer Science)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[136] A. Reka and Barabasi, “Statistical mechanics of complex networks,” *Rev. Mod. Phys.*, vol. 74, pp. 47–97, June 2002.

[137] S. Dorogovtsev and J. Mendes, “Evolution of networks,” *Advances in Physics*, vol. 51, p. 1079, 2002.

[138] D. J. Watts and S. H. Strogatz, “Collective dynamics of ’small-world’ networks.,” *Nature*, vol. 393, pp. 440–442, June 1998.

[139] J. Xu, S. Adabala, and J. A. B. Fortes, “Towards autonomic virtual applications in the in-vigo system,” in *Proc. Second International Conference on Autonomic Computing (ICAC 2005)*, pp. 15–26, 2005.

[140] N. R. Jennings, “An agent-based approach for building complex software systems,” *Commun. ACM*, vol. 44, no. 4, pp. 35–41, 2001.

[141] A. Schaerf, Y. Shoham, and M. Tennenholtz, “Adaptive load balancing: A study in multi-agent learning,” *Journal of Artificial Intelligence Research*, vol. 2, pp. 475–500, 1995.

[142] A. Montresor, H. Meling, and O. Babaoglu, “Messor: Load-balancing through a swarm of autonomous agents,” in *In Proceedings of 1st Workshop on Agent and Peer-to-Peer Systems*, pp. 125–137, 2002.

[143] A. Andrzejak, A. Andrzejak, S. Graupner, S. Graupner, V. Kotov, V. Kotov, H. Trink, and H. Trinks, “Algorithms for self-organization and adaptive service placement in dynamic distributed systems,” tech. rep., 2002.

[144] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, “Bimodal multicast,” *ACM Trans. Comput. Syst.*, vol. 17, no. 2, pp. 41–88, 1999.

[145] I. T. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud computing and grid computing 360-degree compared,” *CoRR*, vol. abs/0901.0131, 2009.

BIOGRAPHICAL SKETCH

Jing Xu grew up in Jilin, a beautiful city in Northeast China. At the age of nineteen, she left her hometown in pursuit of higher education, which has continued till today. Through five years of study at the University of Science and Technology of China, she earned the bachelor degree from the Department of Automation with honor. Afterwards, she began her Ph.D. study with Prof. José Fortes in ACIS lab at the University of Florida, where she has been doing research in the areas of distributed/grid computing, autonomic computing, and virtualization. She received her Ph.D. in electrical and computer engineering from the University of Florida in 2011. Currently, she is an adjunct faculty in the School of Computer and Information Sciences, Florida International University, Miami.

Jing has been married to Ming since May, 2008. They have two wonderful children, Eileen and Elwen.