A Two-Tiered On-Demand Resource Allocation Mechanism for VM-Based Data Centers

Ying Song, Yuzhong Sun, *Member*, *IEEE*, and Weisong Shi, *Senior Member*, *IEEE*

Abstract—In a shared virtual computing environment, dynamic load changes as well as different quality requirements of applications in their lifetime give rise to dynamic and various capacity demands, which results in lower resource utilization and application quality using the existing static resource allocation. Furthermore, the total required capacities of all the hosted applications in current enterprise data centers, for example, Google, may surpass the capacities of the platform. In this paper, we argue that the existing techniques by turning on or off servers with the help of virtual machine (VM) migration is not enough. Instead, finding an optimized dynamic resource allocation method to solve the problem of on-demand resource provision for VMs is the key to improve the efficiency of data centers. However, the existing dynamic resource allocation methods only focus on either the local optimization within a server or central global optimization, limiting the efficiency of data centers. We propose a two-tiered on-demand resource allocation mechanism consisting of the local and global resource allocation with feedback to provide on-demand capacities to the concurrent applications. We model the on-demand resource allocation using optimization theory. Based on the proposed dynamic resource allocation mechanism and model, we propose a set of on-demand resource allocation algorithms. Our algorithms preferentially ensure performance of critical applications named by the data center manager when resource competition arises according to the time-varying capacity demands and the quality of applications. Using Rainbow, a Xen-based prototype we implemented, we evaluate the VM-based shared platform as well as the two-tiered on-demand resource allocation mechanism and algorithms. The experimental results show that Rainbow without dynamic resource allocation (Rainbow-NDA) provides 26 to 324 percent improvements in the application performance, as well as 26 percent higher average CPU utilization than traditional service computing framework, in which applications use exclusive servers. The two-tiered on-demand resource allocation further improves performance by 9 to 16 percent for those critical applications, 75 percent of the maximum performance improvement, introducing up to 5 percent performance degradations to others, with 1 to 5 percent improvements in the resource utilization in comparison with Rainbow-NDA.

Index Terms—Data centers, virtual machines, on-demand resource allocation, optimization, algorithm, model

1 Introduction

We have witnessed the rapid growth of data centers in the past few years, and expect the number of data centers will triple by 2020. Fighting for green data centers is the biggest challenge faced by computer scientists and practitioners. Google argues that more requests served by the same platform is another path to green data centers [55]. Thus, one of the efficient solutions for the green data centers is improving the throughput as well the resource utilization by server consolidation based on virtualization technology, which is verified by several previous efforts [1] and our previous work [35]. With effective isolation and agile resource management provided by virtualization technology, virtualized data center is also the infrastructure of most cloud platforms. In such a shared virtual computing environment, dynamic load changes as well as different quality requirements of applications in their lifetime give rise to dynamic and various capacity demands (e.g., computing, storage, and communication capacities), which result in lower resource utilization and lower application quality using the existing static resource allocation. Furthermore, the total required capacities of all the hosted applications in current enterprise data centers (e.g., Google) may surpass the capacities of the platform. Thus, the existing techniques by turning on or off servers with the help of virtual machine (VM) migration is not enough. Instead, finding an optimized dynamic resource allocation method in server consolidation scenarios to solve the problem of on-demand resource allocation is the key to improve the resource utilization and throughput, so as to optimize power efficiency of data centers.

Most contemporary virtual machine monitors (VMMs, such as Xen [3] and VMware [40]) provide the technical support [41] rather than strategies for on-demand resource allocation to VMs. To optimize the usage of resources and improve the quality of the hosted applications, many researchers [30], [40] are focusing on the local resource allocation to VMs within a server. However, in a VM-based data center, a number of VMs distributed onto various servers host copies of the same application. Each server independently allocates resources to VMs using its local optimization, which may readily result in unbalance of resource allocation among applications so as to limit the efficiency of data centers. This means that local optimization cannot always lead to global optimization [23]. To

- *Y. Song and Y. Sun are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, No. 6 Kexueyuan South Road, Zhongguancun, Haidian District, Beijing 100190, China. E-mail: songying@ncic.ac.cn, yuzhongsun@ict.ac.cn.*
- *W. Shi is with the Department of Computer Science, Wayne State University, 5057 Woodward Ave, Suite 14102, Detroit, MI 48202. E-mail: weisong@wayne.edu.*

Manuscript received 13 June 2010; revised 7 Dec. 2010; accepted 20 May 2011; published online 16 June 2011.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2010-06-0086.
Digital Object Identifier no. 10.1109/TSC.2011.41.

optimize resource allocation in the data center, it is necessary to provide a global resource scheduling. However, current VMMs do not support the resource allocation to VMs residing in remote servers. Under this technical limitation, Wang [45] optimized the global resource allocation only using a central controller. Yet, such central optimization has the problems of complexity, single-point failure, and nontimeliness. However, hosted web-based applications with sudden resources demand spikes affect each other because of resource competition. Slow response on the resource allocation cannot introduce the optimized allocation. Thus, optimizing the global resource allocation is a challenge for VM-based data centers.

To optimize the resource allocation in a data center, we propose a two-tiered on-demand resource allocation mechanism, including the local and global resource allocation, based on a two-level control model. A well-designed on-demand resource allocation algorithm may minimize the waste of resources as well as guarantee the quality of the hosted applications. In our study, the local on-demand resource allocation on each server optimizes the resource allocation to VMs within a server taking the allocation threshold into account, while the global on-demand resource allocation optimizes the resource allocation among applications at the macro level by adjusting the allocation threshold of each local resource allocation. To guide the design of the on-demand resource allocation algorithms, we model the resource allocation using optimization theory. These algorithms dynamically allocate resources to VMs according to the time-varying capacity demands and the quality requirements of applications. All the on-demand resource allocation algorithms preferentially ensure performance of the critical applications named by the manager when resource competition arises.

We implement a Xen-based prototype, Rainbow, to evaluate the VM-based server consolidation and the two-tiered on-demand resource allocation algorithms on a workload scenario reflecting the resource demands in a enterprise environment. The experimental results show that server consolidation without dynamic resource allocation (Rainbow-NDA) provides 26 to 324 percent improvements in the application performance, as well as 26 percent higher average CPU utilization than traditional exclusive computing framework (TSF, illustrated in Fig. 1a) in typical enterprise environments. The two-tiered on-demand resource allocation further improves the performance by 9 to 16 percent for those critical applications, which are up to 75 percent of the maximum performance improvement, while introducing up to 5 percent performance impairment to others, with 1 to 5 percent improvements in the resource utilization in comparison with Rainbow-NDA. These results indicate that the our on-demand resource allocation improves the resource utilization and quality of applications with inappreciable overheads.

This paper has the following contributions: 1) We propose a novel two-tiered on-demand resource allocation mechanism with feedback to optimize the resource allocation for VM-based data centers. 2) In order to guide the design of the on-demand resource allocation algorithm, we model the resource allocation using optimization theory.

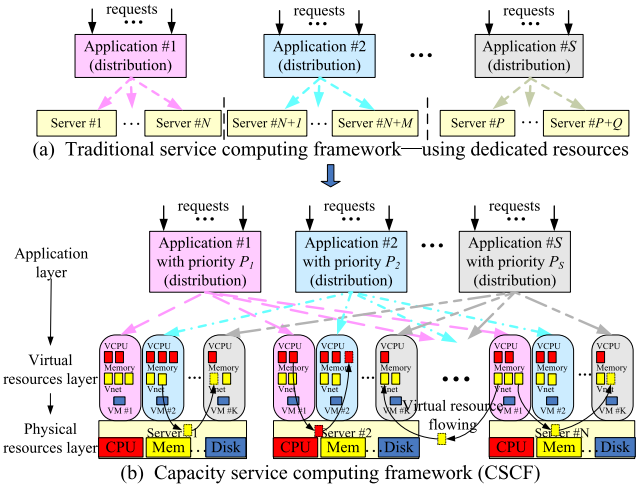


Fig. 1. The evolution of service computing framework.

3) Base on the two-tiered on-demand resource allocation mechanism and model, we propose local and global resource allocation algorithms to optimize the dynamic resource provision for VMs.

2 Background and Related Work

2.1 Background

It's a new trend for enterprise data centers to concurrently provide web-oriented applications. We observe that diverse applications in such data centers may greatly vary in resource intensity. For example, VoD is I/O intensive, and database is CPU intensive. Furthermore, we obtain another observation that diverse applications may have various time-varying capacity demands as the result of request arrival distributions [43], [50]. We also observe that the total required capacities of the hosted applications, for example, Google, may surpass the capacities of the platform. Those three observations motivate our design of a novel capacity service computing framework, CSCF [34] (illustrated in Fig. 1b), to improve the resource utilization of the platform and the quality of application. Different from the traditional service computing framework (TSF, illustrated in Fig. 1a) in which one application runs on a set of dedicated servers, CSCF uses virtualization to isolate concurrent applications in a shared platform. In order to minimize the interaction among the hosted applications due to their competitions for resources, CSCF distributes applications with the same resource intensity onto different servers. In CSCF, all the VMs serving the same application constitute an application domain. CSCF distributes VMs belonging to a single application domain onto various servers, while each server hosts VMs belonging to different application domains. Between the application request layer and the physical resource layer, there are mappings from requests to VMs and from virtual resources in VMs to physical resources in servers. This work focuses on the second mapping to provide on-demand resource allocation among VMs, which may allow better resource utilization and quality of applications compared to previous proposals [30] in scenarios where there are competitions for the same resource by applications with similar resource intensity.

2.2 Related Work

Currently, a large body of research is about managing VM-based data centers, such as HP's SoftUDC [18], Microsoft's DSI initiative [27], and VMware DRS [40]. We classify such research into two subfields: VM lifecycle management and VM-based resource management.

VM lifecycle management [12], [13], [18], [20], [22], [33]. These works manage the VM lifecycle by operations such as VM creating, starting, stopping, and migrating. Virtuoso [33] creates a marketplace in which resource providers sell resources in the form of VM. Entropy [12] performs a globally optimized placement including the VM runnning, migrating, suspending, resuming, and stoping operations according to cluster resource usage. Some researchers focus on using VM migration to provide automatic load balancing [48] as well as to reduce the power waste with the help of turning on or off servers [5], [21], for example, SoftUDC [18] and VMware's VMotion [28]. However, the CPU and network overheads of VM migration may reduce the performance of the hosted applications, and VM migration is the simultaneous reallocation of a set of resources including CPU, memory, and so on, not reallocation of only one type of resource.

VM-based resource management [2], [16], [30], [40]. These works optimize resource allocation to VMs. Our work belongs to this subfield. We classify a large body of works in this subfield into the following three parts:

Providing on-demand resources at the granularity of physical or virtual servers [2], [44]. Oceano [2] dynamically allocates resources for an e-business computing utility at the granularity of a server. Wang et al. [44] dynamically allocate resources to applications via adding or removing VMs on servers. All these works are in contrast to our on-demand resource allocation that controls resource allocation at the granularity of resource component, for example, memory.

The technical support on resource reallocation to VMs [3], [31], [41]. VMware proposes Balloon drivers [41] to overbook memory and dynamically reallocate memory from one VM to another. Xen [3] uses credit scheduler to allocate CPU time slots to vCPU. These technologies provide support to allocate fine-grained resources to VMs. Based on these technical support, our work propose policies for on-demand resource allocation to achieve the goal of high resource utilization and good quality of applications.

Providing on-demand fine-grained resources in a virtualized platform [7], [10], [16], [17], [29], [49]. IBM's PLM [16] and VMware DRS [40] dynamically allocate resources to VMs according to the resource utilization and the statical shares. Padala et al.'s work [30], [29] dynamic allocates CPU and disk based on the VM utilization and application-level QoS metrics. Zhao and Wang [51] introduce Memory Balancer (MEB) which dynamically monitors the memory usage of each VM, predicts its memory needs, and periodically reallocates memory. In Menasce's work [26], the authors present an autonomic controller to optimize a utility function within a physical server for the virtualized environment. In Weng's work [47], the authors present a hybrid scheduling framework for the CPU scheduling in VMM. They adopt different scheduling algorithms correspondingly for two types of VMs. Xu et al. [49] propose a two-level resource management system with local controllers at the VM level and a global controller at the server level. These local and global controllers only correspond to the local resource scheduler in our work limiting the scope of resource allocation within a server. Such two-level resource management could not optimize resource provision for various applications hosted in the entire system with several servers. We also propose the similar two-level schedulers [54] to Xu's work in 2007. Besides the server-level scheduler, this work proposes a system-level scheduler, named global resource scheduler to optimize resource allocation in the entire system, which is the next step of Xu's work and our previous work [54]. In Cunha et al.'s work [7] and Jung et al.'s work [17], the authors address the dynamic resource allocation in multitier virtualized service hosting platforms. Wang et al. [46] evaluate the overhead of a dynamic allocation scheme. All the above works only focus on the resource allocation among VMs within a server ignoring the resource optimization among applications in the entire data center. In this paper, we not only care about the local scheduling in a server but also deal with the global scheduling to optimize the resource allocation among applications.

In Wang et al.'s work [45], the authors optimize the global resource allocation for multitier applications. This optimization is a central control, which has the problems of complexity (collecting and computing resource allocation to each VM hosted in every VMM), availability (the single point failure), and nontimeliness (the execution intervals (22 minutes) could not be small enough because of the scalability). However, the hosted web-based applications may interact because of their sudden demands on resources. Such slow response on the fine-grained resource allocation could not satisfy the sudden changes of resource requirement in realistic workloads. In contrast, our work attempts to address these issues using a two-tiered resource allocation mechanism. The local resource scheduler controlling resource allocation locally with the simple function works in small intervals (e.g., 1 second) in each server, which could quickly respond to the sudden changes of resource demand by the hosted applications. The global scheduler controlling resource allocation globally with the simple function works in 1 or 5 minutes intervals as a complement to the local scheduler. All these schedulers work independently. Any scheduler's failure (even the global scheduler) could not lead to the failure of the resource allocation in the system.

To the best of our knowledge, no other works propose the same two-tiered on-demand resource allocation mechanism and algorithms as ours to optimize the fine-grained resource allocation in VM-based data centers.

3 Two-Tiered On-Demand Resource Allocation Mechanism and Control Model

We aim to propose on-demand resource allocation strategies. A resource allocation strategy should solve four problems: Which resource will be allocated? When will such resource be allocated? Which VMs will be the source or target of allocating? How many resources will be

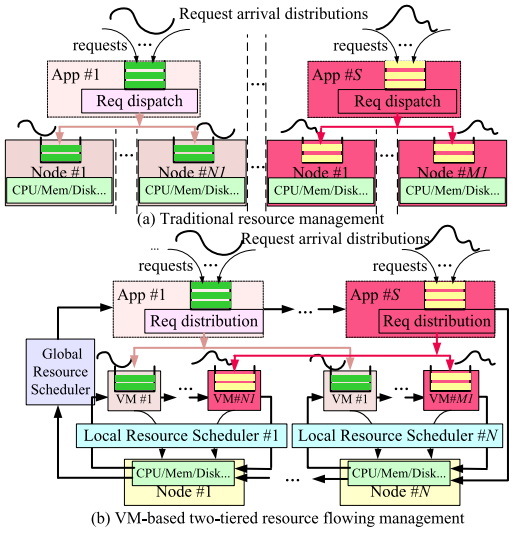


Fig. 2. Two-tiered on-demand resource allocation mechanism versus traditional resource management. (Virtual resource layer results in two-tiered resource allocation.)

allocated? To answer these problems, we first design the resource allocation mechanism, then we use control theory to model and design our feedback-driven two-tiered resource allocation.

Inspired by the multilevel optimization-based techniques for managing distributed computing systems (e.g., Cluster Reserves [52], Neptune [53], MBRP [56], and Kandasamy et al.'s work [19]), we introduce the multilevel management idea into the on-demand resource allocation in VM-based computing environment. Virtualization technology can encapsulate an application into an operating system and dynamically allocate resources to the hosted applications agilely with more abundant feedback information, for example, characteristics of application workloads and resource utilizations of VMs, compared with the traditional large-scaled distributed computing environment. Kandasamy et al. [19] propose a structure of three level controllers aimed at operating the cluster in energy-efficient fashion while satisfying the QoS goal. In their work, only one level controller controls the resource allocation in each computer, and other two levels are responsible for workload distribution. However, we do not care about the workload distribution, our two-tiered controllers both focus on the resource allocation in a virtualized computing environment.

3.1 Two-Tiered Resource Allocation Mechanism

The goal of on-demand resource allocation is to optimize the dynamic resource provision for VMs. Fig. 2 illustrates the two-tiered on-demand resource allocation mechanism (Fig. 2b) as well as the traditional resource management (Fig. 2a). After comparing Figs. 2a and 2b, we can easily find that the two-tiered on-demand resource allocation mechanism differs from the traditional resource management in adding a resource management level for VMs. Each application ("application #1" ..."application #S" in Fig. 2b) has multiple instance copies each of which is encapsulated in a VM. The VMs hosting instance copies of the same application constitute an application domain. Each server hosts VMs belonging to multiple application domains. In such a scenario, workloads in VMs residing in the same server are time varying and different from each other, resulting in the requirement of dynamic resource allocation among VMs within a server. Furthermore, workloads of each application running on various servers are also time varying and different from other applications, resulting in the requirement of on-demand resource allocation among applications. Based on the technical support on dynamic resource allocation to VMs within a server provided by the current VMMs, we can independently control resource allocation to VMs in each server. However, only such independent control in each server may readily result in unbalance of resource allocation among applications so as to limit the efficiency of data centers. Currently, there is no technological support on the resource allocation by VMM on a server to a VM residing in another server. Thus, it is necessary to provide a global resource optimization based on the existing virtualization technology in such shared environment. We propose a two-tiered on-demand resource allocation mechanism implemented by the local resource scheduler and global resource scheduler in Fig. 2b.

The local resource scheduler controls resource allocation to VMs within a server. It adds a set of on-demand resource allocation algorithms based on the technical support on dynamic resource allocation provided by the existing VMMs. To maintain high resource utilization as well as guarantee the quality of applications, the local resource scheduler automatically optimizes the resource allocation to VMs via adjusting CPU time slots and memory assigned to each VM, according to its resource utilization as well as quality and *activity* of the application hosting in the VM. (Activity denotes the threshold of resource allocation, e.g., if the CPU activity of an application is 90 percent, some CPU resource will be allocated to it when its CPU utilization reaches 90 percent.) To allocate resources in a timely manner, the local resource scheduler works at short intervals, for example, 1 second. In the local resource scheduler, the activity of an application is the resource utilization threshold. When the resource utilization of a VM hosting such an application reaches the threshold, some resources should be allocated to the application. Thus, activities are the key parameters effecting the resource allocation.

The global resource scheduler indirectly controls resource allocation among applications in the entire system by adjusting activities of the applications in each local scheduler. In our scenario, each application may have several copies running on multiple servers so that each application can indiscriminately use resources of these servers via on-demand resource allocation controlled by the local scheduler. Adjusting activities of applications influences resource allocation among VMs within each server, resulting in dynamic resources allocation among these applications. The global resource scheduler systematically optimizes resource allocation among applications by periodically adjusting the activity of each application. Thus, the activities of applications are the bridge between the local resource scheduler and the global resource scheduler in our mechanism. Differing from the local resource scheduler, the

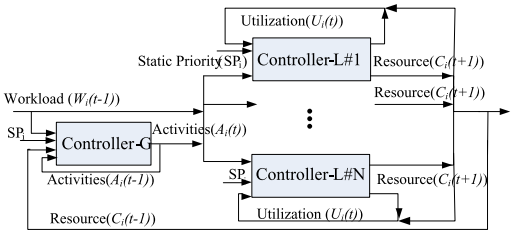


Fig. 3. Two-level feedback control model consisting of Controller-L and Controller-G (of local resource scheduling and global resource scheduling).

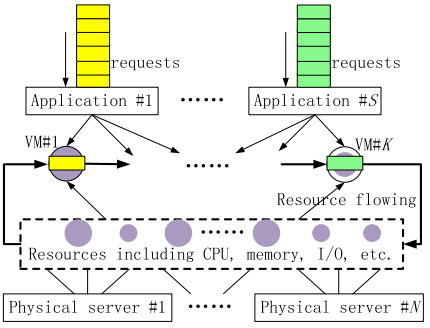


Fig. 4. The on-demand resource allocation problem.

global resource scheduler optimizes the resource allocation at a longer interval, for example, 30 seconds.

3.2 Two-Level Control Model

We use the control theory as the basis of modeling and designing our feedback-driven closed-loop resource allocation algorithms. An object to be controlled is typically represented as an input-output system, where the inputs are the control knobs and the outputs are the metrics being controlled. We employ a two-level control model (illustrated in Fig. 3) for our two-tiered on-demand resource allocation mechanism. Controller-L and Controller-G correspond to the local and global resource scheduler, respectively.

Controller-L controls resource allocation in a server according to the resource utilization ($U_i(t)$) of each VM, the static priority (SP_i), and the activity ($A_i(t)$) of each application. The resource utilization refers to the CPU utilization and the idle memory. As the output, $C_i(t+1)$ refers to the resource assigned to VM_i at time $t+1$, where $C_i(t+1)$ should be a value between the maximum and minimum available resource thresholds of VM_i to avoid the huge interaction among applications. As the actuator, the local resource allocation algorithm controls the resource allocation to VMs in a server.

Controller-G adjusts activities of applications. Resources allocated to each application ($C_i(t-1)$), static priority (SP_i), workloads ($W_i(t-1)$), and the feedback activity ($A_i(t-1)$) of each application are the inputs. As the output, activity ($A_i(t)$) of each application is the key parameter to control resource allocation in Controller-L. A feedback control loop requires an actuator to implement the changes indicated by the control knobs. The global resource allocation algorithm is its actuator.

4 The On-Demand Resource Allocation Model

We consider the problem of on-demand resource allocation to VMs and model it using optimization theory. We model the K-VM-1-PM (PM denotes physical machine) problem and the K-VM-N-PM problem to depict the resource allocation to K VMs within a server and to K VMs residing in N servers, respectively. These two models are general ones, and CPU or other resource allocations can, respectively, use them. Based on the K-VM-1-PM model, we propose a set of priority-based resource allocation algorithms to provide on-demand resources to the hosted applications.

In the on-demand resource allocation model (illustrated in Fig. 4), the number of request queues equals the number of VMs, and the granularity of allocation is resource components (e.g., memory). The resource allocation should make decisions on which resource will be allocated, how many resources will be allocated, and so on to improve the quality of applications.

4.1 The *K*-VM-1-PM Model

The K-VM-1-PM problem depicts the on-demand resource allocation to K VMs within a server. Thus, we model it to guide the algorithm design for the on-demand resource allocation to VMs within a server. First, we introduce the following notations:

- K is the number of VMs residing in the server.
- C_{i-min} is the minimum threshold of resources allocated to VM_i. We use C_{i-min} to avoid huge interaction among the VMs when the competition for resources arises.
- R is the total CPU or other resources that are available to all VMs in a server. R_{out} is the amount of resources allocated to all the hosted VMs. R_{it} is the amount of resources allocated to VM_i at time t, where $R \geq \sum_{i=1}^{K} R_{it}$ and $R_{it} \geq C_{i-min} > 0$.
- D_{it} is the resource demand of VM_i at time t, and it is proportional to the request arrival rate.
- SP_i is the static priority of an application hosted in VM_i. It indicates how critical the requirement for quality of this application. The smaller the SP_i is, the more preferential the application gains. The administrator determines SP_i before running the applications. It does not change during the runtime.
- Φ_i is the tolerable quality threshold of the application hosted in VM_i.
- Q_{it} is the quality of application hosted in VM_i at time t, which is the performance metric. The smaller the Q_{it} is, the better quality the application gains. As we all know, the resources which it demands for (D_{it}) and the resources allocated to it (R_{it}) determine the quality of an application (Q_{it}), for example, the response time. In other words, Q_{it} is a function of D_{it} and of R_{it}, namely, $Q_{it} = f_i(R_{it}, D_{it})$.

To fairly weight different applications using their respective qualities, we use the quality rate of application hosted in VM_i at time t and of the tolerable quality of such application (Q_{it}/Φ_i, "Q-rate" for short) to normalize the qualities of applications. C_{i-min} is used to guarantee the tolerable quality of application i using the relationship among Q_{it}, D_{it}, and R_{it}, as well as taking SP_i into account,

which is set by experience in our experiments, and we will justify it in the near future.

The goal of the resource allocation is to optimize the qualities of the hosted applications taking their priorities into account, giving the limited resources. It is an optimization problem with limiting conditions. Thus, we select the programming model of optimization theory to model the on-demand resource allocation. To provide the resource allocation with a utility function [26] that maps application quality of the target to a benefit value, we define the utility function UF_t. UF_t is related to the static priorities and Q-rates of the hosted applications

$$UF_t = \sum_{i=1}^{K} \frac{Q_{it}}{\Phi_i} \times SP_i = \sum_{i=1}^{K} \frac{f_i(R_{it}, D_{it})}{\Phi_i} \times SP_i. \quad (1)$$

The on-demand resource allocation problem is how to control the resource allocation to VMs with the goal of minimizing the utility function UF_t, giving the limited resources, formulated as follows:

$$min\ UF_t = \sum_{i=1}^{K} \frac{f_i(R_{it}, D_{it})}{\Phi_i} \times SP_i$$

$$s.t. \begin{cases} \sum_{i=1}^{K} R_{it} \le R \\ R_{it} \ge C_{i-min} \quad (i = 1, 2, \ldots, K). \end{cases} \quad (2)$$

Our former work [34] gave the functions $Q_{it} = f_i(R_{it}, D_{it})$ of several typical enterprise applications including the web, database, and office applications, and of different resources such as CPU and memory with plentiful experiments as follows: $f_{ij}(R_{it}, D_{it}) = a_{ij}D_{it} + b_{ij}R_{it} + g_{ij}$, $U_i(j-1) < D_{it} \le U_{ij}$, where $f_{ij}(R_{it}, D_{it})$ denotes the jth stage of function $f_i(R_{it}, D_{it})$, a_{ij}, b_{ij}, and g_{ij} refers to the coefficients in function $f_{ij}(R_{it}, D_{it})$, and U_{ij} denotes the upper threshold of resources for application i at stage j. Putting these functions into our model, we get the following formulation:

$$minUF_t = \sum_{i=1}^{K} \frac{\sum_{j=1}^{m} a_{ij} \times D_{it} + b_{ij} \times R_{it} + g_{ij}}{\Phi_i} \times SP_i$$

$$s.t. \begin{cases} \sum_{i=1}^{K} R_{it} \le R \\ R_{it} \ge C_{i-min} \quad (i = 1, 2, \ldots, K), \end{cases} \quad (3)$$

where m denotes the maximum number of stages for $f_i(R_{it}, D_{it})$. As for the on-demand resource allocation problem, we may resolve the above model to get the close-to-optimal resource allocation R_{it} at time t using the Simplex Method.

4.2 The *K*-VM-N-PM Model

The K-VM-N-PM problem depicts on-demand resource allocation to K VMs residing in N servers, where each VM may use resources in more than one server concurrently. Thus, we model it to guide the design of algorithm of the on-demand resource allocation among VMs each of which may use resource in various servers concurrently, which is similar to the K-VM-1-PM model. First, we introduce the following notations:

- R is the total CPU or other resources that are available to all VMs in N servers. R_i is the total CPU or other resources that are available to the hosted VMs in server i, where $1 \le i \le N$ and $R = \sum_{i=1}^{N} R_i$.
- K is the number of VMs residing in the N servers. K_i is the number of VMs residing in server i, where $1 \le i \le N$ and $K = \sum_{i=1}^{N} K_i$.
- RN_{ijt} is the native resources allocated to VM_{ij} in server i (VM_{ij} denotes the jth VM in server i). RO^x_{ijt} is the amount of resources allocated to VM_{ij} in server x, namely, the amount of remote resources (located in server x) allocated to VM_{ij}, where $1 \le x \le N$.
- C_{ij-min} is the minimum threshold of resources allocated to VM_{ij}. We use C_{ij-min} to avoid huge interaction among the VMs hosting applications when the competition for resources arises.
- R_{ijt} is the amount of resources allocated to VM_{ij} at time t, where $R_{ijt} = RN_{ijt} + \sum_{x=1}^{N} RO^x_{ijt}$. R_{ijt} obeys the rules as follows:

$$R \ge \sum_{i=1}^{N} \sum_{j=1}^{K_N} R_{ijt}$$

  and $R_{ijt} \ge C_{ij-min} \ge 0$ ($i = 1, \ldots, N$; $j = 1, \ldots, KN$).
- D_{ijt} is the resource demand of VM_{ij} at time t, and it is proportional to the request arrival rate.
- SP_{ij} is the priority of application hosted in VM_{ij}. It indicates how critical the requirement for quality of this application. The administrator determines SP_{ij}.
- Φ_{ij} is the tolerable quality threshold of the application hosted in VM_{ij}.
- Q_{ijt} is the quality of application hosted in VM_{ij} at time t, which is the performance metric. The smaller the Q_{ijt} is, the better quality the application gains. As we all know, the quality of an application (Q_{ijt}), such as the response time, is decided by the resources it demands for (D_{ijt}) and the resources allocated to it (R_{ijt}). In other words, $Q_{ijt} = f_{ij}(R_{ijt}, D_{ijt})$.

Similar to the K-VM-1-PM model, we use Q-rate (Q_{ijt}/Φ_{ij}) to normalize the quality of different applications, and we get the following model using the same method in Section 4.1:

$$min\ UF_t = \sum_{i=1}^{N} \sum_{j=1}^{K_i} \frac{f_{ij}(RN_{ijt}, \sum_{x=1}^{K} RO^x_{ijt}, D_{ijt})}{\Phi_{ij}} \times SP_{ij}$$

$$s.t. \begin{cases} \sum_{i=1}^{N} \sum_{j=1}^{K_i} R_{ijt} \le R \\ R_{ijt} \ge C_{ij-min} \\ \qquad (i = 1, 2, \ldots, N; j = 1, 2, \ldots, K_i) \\ \sum_{j=1}^{K_i} RN_{ijt} + \sum_{j=1}^{K_i} RO^i_{ijt} \le R_i \\ R_{ijt} \ge 0 \\ \qquad (i = 1, 2, \ldots, N; j = 1, 2, \ldots, K_i), \end{cases} \quad (4)$$

where

$$Q_{ijt} = f_{ij}(R_{ijt}, D_{ijt}) = f_{ij}\left(RN_{ijt}, \sum_{x=1}^{K} RO_{ijt}^{x}, D_{ijt}\right).$$

The third condition in formulation 4 denotes that the total resources provided by a server are no less than the sum of the resources allocated to the native VMs and the resources allocated to the remote VMs. When a VM on one server uses the remote resources, some performance impairment may come out compared with using the same amount of local resources. We can measure such performance impairment offline based on our ongoing distributed VMM (DVMM). Using such performance impairment, the conversion from RO_{ijt}^{x} to RN_{ijt} can be calculated. Thus, Q_{ijt} in the K-VM-N-PM model is converted to Q_{it} in the K-VM-1-PM model.

We may use the same method to solve the K-VM-N-PM model as the K-VM-1-PM model. Now, we can only verify the K-VM-1-PM model using the following algorithms. However, now we can neither determine the function $f_{ij}(R_{ijt}, D_{ijt})$ nor verify the K-VM-N-PM model, because current VMMs have no technical support on using remote resources by a VM (namely, one VM using resources in more than one server concurrently). Developing a DVMM to support such remote resource access is one of the trends in virtualization field, for example, HP's SoftUDC [18], which is also our on-going project. Thus, in the future, we will evaluate the K-VM-N-PM model and the corresponding algorithms based on our DVMM project.

5 On-Demand Resource Allocation Algorithms

We propose local and global on-demand resource allocation algorithms based on the K-VM-1-PM model. From the local perspective, we simplify the K-VM-1-PM model to design a priority-based local on-demand resource allocation algorithm within a server. Based on the K-VM-1-PM model, we propose a global on-demand resource allocation algorithm to optimize the resource allocation among applications. The efficacy of our algorithms intimately depends on how well it can predict the resource utilization and the number of arrival requests. While it is certainly tempting to try sophisticated prediction techniques, we take a simple low-overhead last-value-like prediction as reference [10] does, in which we use the resource utilization of VMs and the number of arrival requests during the last interval as a predictor at the next interval.

5.1 The Local Resource Allocation Algorithm

We propose a set of local on-demand resource allocation algorithms ("ResourceFlow-L" for short), based on the above control model and the simplified K-VM-1-PM model. In the simplified K-VM-1-PM model, all functions $f_i(R_{it}, D_{it})/\Phi_i$ are set as follows: If $D_{it} > R_{it}$, $f_i(R_{it}, D_{it})/\Phi_i = D_{it} - R_{it}$; else, $f_i(R_{it}, D_{it})/\Phi_i = 0$. Applying the Simplex Method, we get the resolution of this simplified model. If $D \leq R$, $R_{it} = D_{it}$; else, we give priority to allocating resources to VMs with higher priority. The resource utilization of VM_i directly reflects the relationship between D_{it} and R_{it}. Thus, ResourceFlow-L control resource allocation according to the resource utilization of each VM and the static priority of each application. ResourceFlow-L includes the local on-demand CPU allocation algorithm (CpuFlow-L) and the local and lazy on-demand memory allocation algorithm (MemFlow-L). We simply introduce these algorithms as follows.

5.1.1 The Local CPU Allocation Algorithm

The hypervisor (Xen) we used provides a credit scheduler [6], which is a general CPU scheduler in current VMMs, to allocate CPU to VMs in proportion to their weights. The local on-demand CPU allocation algorithm (CpuFlow-L) dynamically adjusts the weights of VMs according to their static priorities, resource utilizations, and activities of the hosted applications. It preferentially guarantees some critical applications with the rapidly increased weights, while other applications may suffer performance degradation via the slowly increased weights when the competition for CPU arises. The increased weights are in proportion to their static priorities. To avoid the huge negative effect among the hosted applications, the amount of resources allocated to a VM should be a value between the maximum and minimum available resource thresholds of the VM.

CpuFlow-L controls CPU allocation, which answers the above four problems about the on-demand resource allocation. Based on the periodically collected CPU utilization of each VM, CpuFlow-L determines if there is CPU overload in a VM. We choose T_u as the threshold of CPU overload (activity of the hosted application) and T_d as the desired CPU utilization level. If the CPU utilization of VM_i reaches T_u, some more CPU resources should be allocated to VM_i. CpuFlow-L increases the weight of VM_i to increase the CPU allocated to this VM. If the CPU utilization of VM_j is lower than T_d, CpuFlow-L decreases the weight of VM_j to decrease the CPU allocated to this VM.

5.1.2 The Local and Lazy Memory Allocation Algorithm

The local and lazy on-demand memory allocation algorithm (MemFlow-L) dynamically controls memory allocation, which answers the above four problems about the on-demand resource allocation. Based on the static priority and the periodically collected idle memory ("IM" for short) of each VM_i, MemFlow-L determines whether there is memory overload in a VM or not. The activity Θ refers to the threshold of idle memory for memory overload. If idle memory (IM) of each VM is higher than Θ, no memory need to be reallocated. If IM_i is lower than Θ, MemFlow-L increases memory for VM_i, as long as there is another VM that can give some of its memory to VM_i.

5.2 The Global Resource Allocation Algorithm

When guiding the design of the global on-demand resource allocation algorithm (ResourceFlow-G), the K-VM-1-PM model evolves to optimize resource allocation among applications in the entire system. In our scenario, we may distribute the VMs hosting the same application onto all servers, so that each application can indiscriminatingly use resources from these servers. It resembles that each VM can indiscriminatingly use resources of the server in the K-VM-1-PM model. After replacing VM by application, as

well as replacing one server by the entire system, the K-VM-1-PM model and its solution could be used by ResourceFlow-G. Using the K-VM-1-PM model, ResourceFlow-G calculates the optimized resource allocation scheme for each application, namely, the resources which should be allocated to each application (R_{it}).

To control the resource allocation systematically, ResourceFlow-G adjusts the activity of each application according to the monitored resources that should really be allocated to each application (R^r_{it}) and the resources that should be allocated to the application (R_{it}). In most cases, R^r_{it} does not equal R_{it}. Thus, we define a resource threshold (Ψ) that actuates the activity adjustment of VMs, to avoid frequent activity adjustment. Only when the difference between R^r_{it} and R_{it} exceeds Ψ, ResourceFlow-G will adjust the activity. If R^r_{it} is larger than R_{it}, ResourceFlow-G decreases the activity of VM_i at time t. Conversely, ResourceFlow-G increases the activity of VM_i at time t.

To address the single-point failure problem of the global scheduler running the global resource allocation algorithm, we dynamically select a server to run it. If the server running the global scheduler fails (monitored by heartbeat), we randomly select another server to run this scheduler. Even if the global scheduler fails and no other global scheduler replaces it to work, all the local schedulers could continue working to allocation resource to the hosted VMs without the optimization at the macro level by the global scheduler. In a word, the global scheduler's failure could not lead to the failure of the resource allocation in the system. As to the scalability problem, the computation scale of the global scheduler is in direct proportion to the number of applications corresponding to i in the K-VM-1-PM model, because the proposed constrained optimization in the K-VM-1-PM model is linear. Thus, the global scheduler is not the bottleneck even in a large-scale computing environment. From the above analysis, we can see that our two-tiered on-demand resource allocation addresses the problems of availability and scalability faced by the previous work [45].

6 Performance Evaluation

In this section, we first introduce the prototype, Rainbow, we implemented based on Xen. Then, we evaluate our VM-based shared platform as well as on-demand resource allocation algorithms in Rainbow.

6.1 Rainbow Prototype

The implementation of the Rainbow prototype is based on Xen. In the server pool of our prototype, there are four physical servers, each of which has two 2,190 MHz Dual Core AMD Opteron(tm) processors with 1,024 KB of cache and 4 GB of RAM. We use CentOS4.4 and Xen-3.0.4. We use other four machines to generate workloads of applications, for example, web application. The systems are connected with a Gigabit Ethernet. Rainbow is responsible for creating, monitoring, and managing VMs in a distributed computing environment, as well as allocating resources to these VMs.

6.2 Performance Evaluation

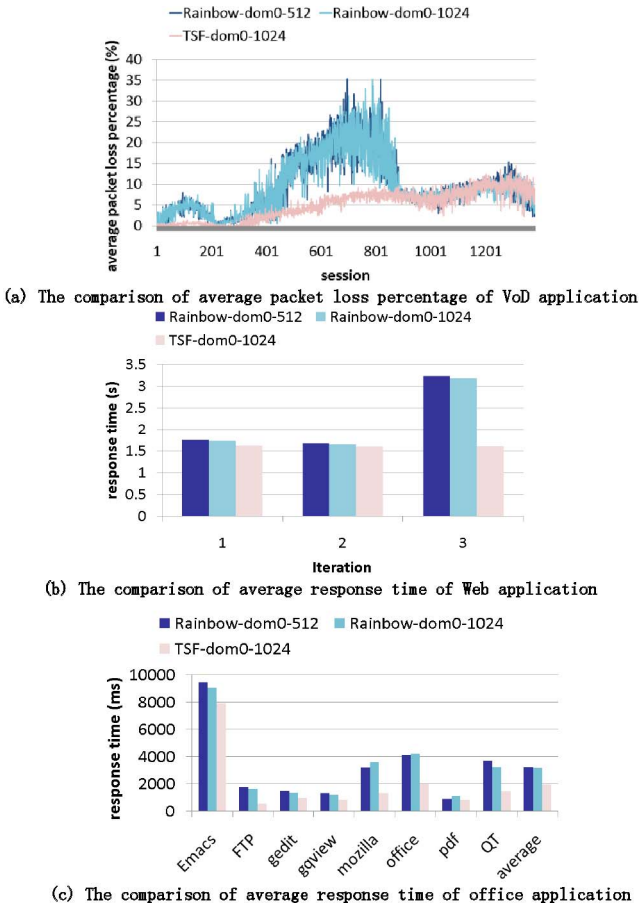
We evaluate the Rainbow in the three groups of experiments in the two cases, the one is without dynamic resource allocation called Rainbow-NDA, another one is with two-tiered on-demand resource allocation called Rainbow-DA. In the first group of experiments, we compare Rainbow-NDA with TSF to evaluate the server consolidation. We evaluate our on-demand resource allocation algorithms in the last two groups of experiments, we compare Rainbow-DA with Rainbow-NDA in the second group of experiment. In the last group of experiment, we compare Rainbow-DA with [30]. All experiments use some of the following applications.

- Web application—The web server is Apache [9]. LVS [25] dispatches requests among the web VMs (xxx VM denotes VM hosting xxx application in this paper.) adopting round robin algorithm. We use the e-commerce workloads of SPECWeb2005 [37] for the web application. The average response time is its performance metric.
- HPC application—We use Condor [38] to dispatch Linpack [24] jobs to the HPC VMs, and we evaluate the HPC application by the throughput (the number of completed Linpack jobs during 3 hours).
- Office application—Another one of our products provides the office application, which distributes office application workloads to VMs according to the resource utilization of these VMs. We use Xnee [32] to emulate the real-world office application workloads based on the trace collected by our work [42]. As to the office application, the response time, including the starting up time of an application and the time of executing an operation in application, is the key metric to evaluate its performance. The starting up time of an application is sensitive to both CPU and memory. Thus, we collect the starting up time of eight typical applications such as FTP, Mozilla, and OpenOffice for four times as the performance metric, captured by the modified VNC. We compare the average starting up time of all the applications to avoid randomicity.
- VoD application—The VoD server uses Helix [11]. We generate the VoD workloads according to Yu et al.'s work [50]. The average packet loss percentage is the performance metric.
- Database ("DB" for short) application—We use MySQL [8] and Apache Tomcat [39] for the DB server, and LVS dispatches requests among the DB VMs adopting round-robin algorithm. TPC-W [4] is the DB workloads generator. The total size of the data base files is 2.7 G. We evaluate the DB application in terms of the average WIPS (the number of web interactions per second).

We introduce the three groups of experiments, respectively, in the following sections.

6.2.1 Rainbow-NDA versus TSF

We evaluate the strengths and weaknesses of Rainbow-NDA in the following two comparisons with various experimental scenarios. Comparison-I compares Rainbow-NDA with TSF using three applications with different

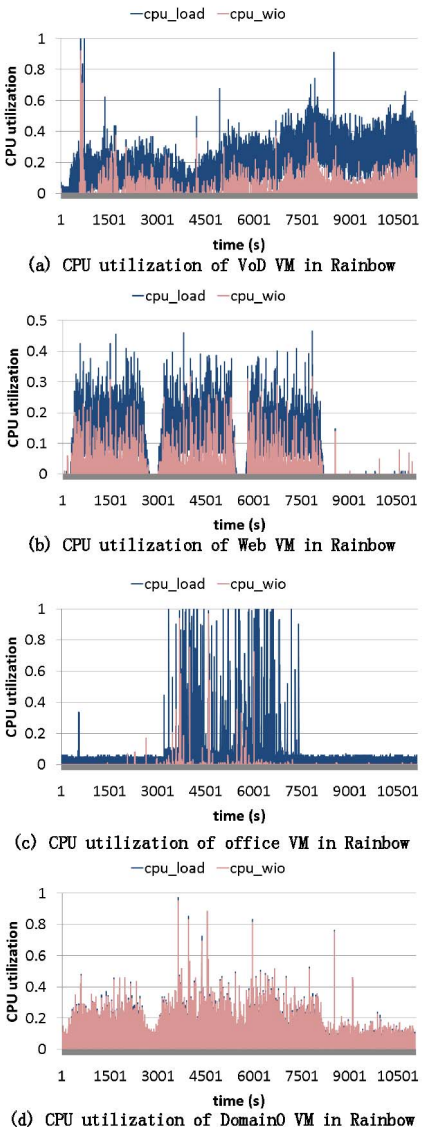


(a) The comparison of average packet loss percentage of VoD application

(b) The comparison of average response time of Web application

(c) The comparison of average response time of office application

Fig. 5. The comparison between TSF and Rainbow-NDA.



(a) CPU utilization of VoD VM in Rainbow

(b) CPU utilization of Web VM in Rainbow

(c) CPU utilization of office VM in Rainbow

(d) CPU utilization of Domain0 VM in Rainbow

Fig. 6. CPU utilization of VMs and Domain0 in Rainbow.

resource intensities, whereas Comparison-II evaluates the influence of the size of memory in Domain0 on Rainbow-NDA, and compares Rainbow-NDA with TSF using two I/O-intensive applications and one CPU&memory-intensive application. First, we introduce the experiments and results of these comparisons. Then, we give the analysis on them.

Comparison-I. We compare Rainbow-NDA with traditional service computing framework (TSF, namely, one application per set of dedicated servers) in the same physical server environment hosting three typical enterprise applications with diverse resource intensities, namely, a web application with CPU and I/O intensity, an HPC application with CPU intensity, and an office application with CPU AND memory intensity. On each server, we create three VMs, each of which has one vCPU and 1 GB memory. We distribute VMs devoted to each application onto the four servers. The experimental results show that Rainbow-NDA provides 28 to 324 percent improvements in the application performance and 26 percent improvement in the CPU utilization over TSF.

Comparison-II. We evaluate Rainbow-NDA by hosting office application with CPU and memory intensity and two I/O-intensive applications, namely, VoD and web applications. We first evaluate the performance effect caused by memory in Domain0 via comparing the performance of these applications in the cases of Domain0 with 512 and 1,024 MB memory denoted by Rainbow-dom0-512 and Rainbow-dom0-1024, respectively, in Figs. 5a, 5b, and 5c. These figures illustrate that the difference in memory of Domain0 leads to tiny or no effect on the application performance. Thus, Domain0's memory is not the bottleneck in this experiment.

Then, we compare TSF and Rainbow-NDA in the case of Domain0 with 1,024 MB memory denoted by Rainbow-dom0-1024 and TSF-dom0-1024, respectively in Figs. 5a, 5b, and 5c. Fig. 5a illustrates the comparison of the average packet loss percentage of VoD. It shows that TSF-dom0-1024 is much better than others, while other cases are similar in the performance of VoD. This figure also implies that the VoD performance degrades up to 46 percent between the 600th and 900th sessions of the VoD when it shares the platform with the web and office applications. After the 900th session of VoD, such impairment disappears.

We analyze the reason why Rainbow-NDA degrades the performance of VoD so remarkably with Figs. 5b, 6, 7, and 8. Figs. 6a and 6b conclude CPU is not the bottleneck resource of the VoD and web applications. Fig. 5b illustrates the throughput comparison of web application between Rainbow-NDA and TSF. This figure implies that the performance

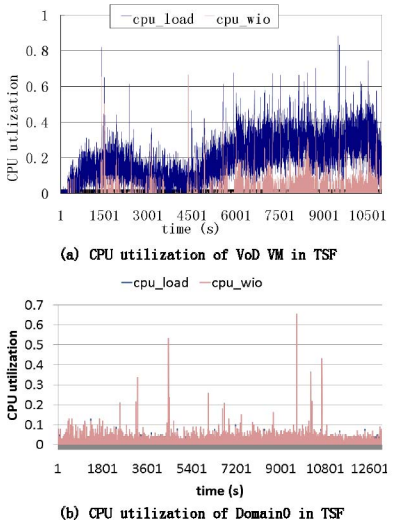


Fig. 7. CPU utilization of VoD VM and Domain0 in TSF.

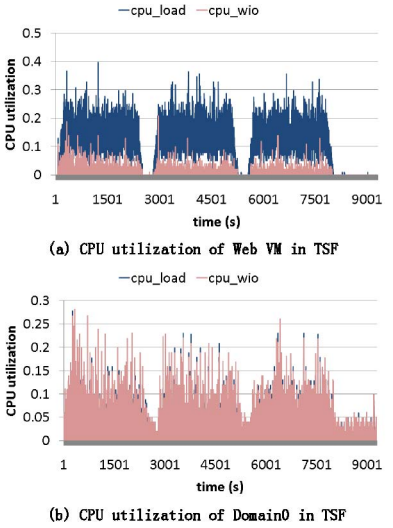


Fig. 8. CPU utilization of web VM and Domain0 in TSF.

of web application in the first two iterations degrades minutely (1 to 5 percent), while it impairs hugely (30 to 60 percent) in the last iteration, resulting in 36 percent degradation in average when Rainbow-NDA compares with TSF. Figs. 6a and 6b show that the workloads of VoD enter their peaks when the workloads of the web application enter the third iteration. In this period, the effect between these two applications is huge, which results in the huge performance degradation of VoD when it shares resources with the other applications.

What resources do they compete for in case of the such performance degradation? Figs. 6a, 6b, 6c, 7, and 8 help to answer this question. In these figures, blue (dark color) lines denote the CPU utilization, and red (light color) lines denote the utilization of CPU used to wait I/O. Fig. 6 illustrates the CPU utilization of VMs and Domain0 in Rainbow-NDA during the test. Figs. 6 and 8 illustrate the CPU utilization in TSF which exclusive hosts the VoD and web applications, respectively. These figures show that the requirement on I/O resource by VoD (Figs. 6a and 7a) and web applications (Figs. 6b and 8a) is remarkable, and most CPU time slots of Domain0 are used to wait for I/O operations (Figs. 6d, 7b, and 8b), which means that I/O is the bottleneck resource of the VoD and web applications. After comparing Figs. 6d, 7b, and 8b, we find that the concurrency of VoD and web applications results in 5.3 times and 1.4 times more CPU slots used to wait I/O in Domain0. Thus, frequent competitions for I/O lead to huge negative effect between the VoD and web applications in Rainbow-NDA.

Fig. 5c illustrates the comparison of the average response time of opening office applications in the same cases as above. Rainbow-NDA increases less than 1 second in the average response time resulting from the competitions for CPU resource. Such delayed response can be ignored by the clients. Fig. 6c shows that the office application is low on I/O intensity, which results in no significant performance effect caused by the VoD and web applications.

Analysis and conclusion. The applications in Comparison-I are with different resource intensities, their resource bottlenecks are various. Distributing copies of the same application onto multiple servers reduces the demands on its bottleneck resource on each server. On the other hand, each server hosts concurrently multiple applications with different resource intensities, which avoids frequent competitions for the same resource. Thus, Rainbow-NDA provides huge improvements when the hosted applications are with different resource intensities. The two I/O-intensive applications in Comparison-II frequently compete for the I/O resource when they share a server. Frequent competitions for I/O resource and Xen's I/O overhead lead to huge negative effects between these applications in Rainbow-NDA.

6.2.2 Rainbow-DA versus Rainbow-NDA

We evaluate our resource allocation algorithms including ResourceFlow-L and ResourceFlow-G via the comparison between Rainbow-DA and Rainbow-NDA.

Rainbow with ResourceFlow-L versus Rainbow-NDA. We compare Rainbow-NDA with Rainbow with ResourceFlow-L in the same experimental scenario as comparison-I of Section 6.2.1. On each physical server, we create three VMs. We initially allocate different resources to VMs (illustrated in Table 1, where "$BN:A$" denotes that resource A is the bottleneck resource when applications reach their peaks of workloads) to provide different test conditions.

ResourceFlow-L optimizes the resource allocation to VMs within a server. In order to compare the system performance without manual control, the baseline system Rainbow-NDA provides static resource allocation to each VM. CpuFlow-L refers to Rainbow adopting the local on-demand CPU

TABLE 1
Initial Resource Allocation to VMs on Various Conditions

conditions	CPU	memory
BN:CPU	1 vCPU (vCPUs of VMs running on the same server are pinned to the same core)	1G
BN:mem	1 vCPU (may map to any core)	600M
BN:CPU & mem	1 vCPU (the same as that in BN:CPU)	600M

TABLE 2
Local Resource Allocation Algorithms versus Rainbow-NDA

condition	comparison	office	web	hpc	resource utilization
BN:CPU	CpuFlow-L	25%	1%	-7%	CPU:2%
BN:mem	MemFlow-L	37%	1%	0%	mem:8%
BN:CPU & mem	ResourceFlow-L	42%	2%	0%	CPU:2%; mem:6%

allocation algorithm. MemFlow-L adopts the local on-demand memory allocation algorithm in Rainbow. ResourceFlow-L denotes adding CpuFlow-L and MemFlow-L in Rainbow. According to characteristics of the applications, we initialize SP_{off}:SP_{web}:SP_{hpc} to be 4:3:1 (suffixes off, web, and HPC denote office, web, and HPC applications, respectively). Under the similar time-varying workloads we execute the comparisons illustrated in Table 2. We collect the CPU utilization and idle memory of VMs as the feedback every 1 second and 5 seconds, respectively. The response time of the CPU and memory allocation is within 1 and 5 seconds.

Table 2 illustrates the comparison results between various resource allocation algorithms and Rainbow-NDA on various conditions. The results show that ResouceFlow-L provides up to 25 to 42 percent improvements in the performance of the office application and up to 2 percent improvements in the performance of the web application, while introducing up to 7 percent impairments in the performance of the HPC application. This is the result that ResourceFlow-L preferentially ensures the performance of the office application by degrading the performance of the web and HPC applications to some extent, and it gives preference to the web application when the web application competes for resources with the HPC application. Table 2 also shows that ResourceFlow-L causes 2 to 8 percent improvements in the average resource utilization. These results imply that ResourceFlow-L in Rainbow can utilize the hardware rationally based on the application differentiation.

Rainbow with ResourceFlow-G versus Rainbow-NDA. We compare *Rainbow with ResourceFlow-G* (adding ResourceFlow-G in Rainbow with ResourceFlow-L) with Rainbow-NDA adopting three typical enterprise applications, namely, the web, database, and office applications. On each physical server, we create three VMs. We initially allocate 700 M memory and one vCPU to each VM. To ensure that the CPU is the bottleneck resource when the applications reach their peaks of workloads, we pin all the vCPU of the three VMs running on one server to the same physical core. Domain 0 of Xen uses the other one core, while the rest cores are idle.

According to the characteristics of these applications, SP_{DB}:SP_{office}:SP_{web} is initialized to be 4:2:1 (suffixes DB, office, and web refer to, respectively, DB, office, and web applications). The global scheduler is invoked every 30 seconds. "ResourceFlow-G" denotes adding ResourceFlow-G in Rainbow with ResourceFlow-L.

Table 3 gives the performance comparisons of adding different resource allocation algorithms in Rainbow against Rainbow-NDA. This table illustrates that ResourceFlow-L provides up to 6 and 16 percent improvements in the performance of the DB and office applications, while introducing 1 percent degradation in the performance of the web application compared with Rainbow-NDA. Compared with ResourceFlow-L, ResourceFlow-G further achieves 3 percent improvement, namely achieves 9 percent improvement compared with Rainbow-NDA, in the performance of DB application by slightly reducing the rate of the performance improvement for the office application (10 percent improvement) as well as degrading the performance of the web application by up to 2 percent. Compared with Rainbow-NDA, the maximum improvement space for DB application is 12 percent in the case that all resources shared by all applications are exclusively allocated to this application. Thus, compared with Rainbow-NDA, the performance improvement for the DB application introduced by ResourceFlow-G is up to 75 percent of the maximum performance improvement. In Comparison-I of Section 6.2.1, ResourceFlow-L provides 42 percent improvement in the performance of the most critical application, while introducing up to 7 percent degradation to the performance of other applications. Thus, the hosted applications and their workloads affect the improvement provided by the on-demand resource allocation algorithms. Table 3 also shows that Rainbow-DA causes up to 5 percent improvements in the average resource utilization. These results imply that Rainbow-DA utilizes the hardware more rationally based on the application differentiation.

TABLE 3
The Results of Two-Tiered On-Demand Resource Allocation Mechanism Compared with Rainbow-NDA

Cases	DB	office	web	CPU utilization	Mem utilization
Resource Flow-L	6%	16%	-1%	0.3%	1%
Resource Flow-G	9%(75% of the max improvement)	10%	-2%	1%	5%

We analyze the reason why Rainbow-DA outperforms Rainbow-NDA in qualities of applications and resource utilization by collecting of the allocated and used resources in each VM in the cases of Rainbow-DA and Rainbow-NDA. When the resource demands of some application increase, Rainbow-DA increase the resource allocated to VMs hosting the application in the case that no resource competition arises. On the other hand, when the resource competition arises, Rainbow-DA controls much resource allocating to the VM with higher priority, which results in more resources allocated to the VM with higher priority than other VMs. In Rainbow-DA, ResourceFlow-G dynamically adjusts activity of each application in each ResourceFlow-L to optimize the resource allocation among applications. In the above experiment, compared with ResourceFlow-L, ResourceFlow-G further achieves improvement in the performance of DB application by increasing the activity of DB application during the test.

6.2.3 Rainbow-DA versus Reference [30]

We compare Padala's work [30] with our Rainbow-DA in Table 4. Both works focus on the resource allocation taking quality of the hosted applications into account based on the

TABLE 4
Comparison between Rainbow-DA and [30]

	resources	working interval	improvement	degradation
Ref. [30]	CPU	10s	28%	41%
Rainbow-DA	CPU&mem	1s(CPU), 5s(mem)	20%	5%

application differentiation. We compute the average improvement and degradation introduced by Padala et al.'s work [30] according to its figures of the response time with and without their controller [30, Figs. 14 and 15]. Their work [30] provides about 28 percent improvement in the performance of the critical application while introducing about 41 percent performance degradation to another one. Although the total improvement provided by their work (28 percent) is slightly larger than that provided by Rainbow-DA (20 percent), the total degradation introduced by their work (41 percent) is much larger than that introduced by Rainbow-DA (5 percent). We cannot evaluate such algorithms of application differentiation only using the performance improvements of some applications. The performance degradation of other applications should be considered. Thus, in order to fairly compare Padala's work and Rainbow-DA in terms of guaranteeing quality of the concurrent applications, we take both the total performance improvement and the total performance degradation provided by these algorithms into account. The comparison of the performance improvement and degradation illustrated in Table 4 implies that Rainbow-DA, greatly improving the performance of critical applications at the cost of slightly degrading the performance of others, is better than Padala et al.'s work, which greatly improves the performance of critical applications at the cost of even more hugely degrading the performance of others, in the aspect of assuring quality of applications.

Let's analyze the reason. Padala et al.'s work [30] only focuses on the CPU allocation among VMs within a server and uses the fixed allocation threshold according to the experience. On the other hand, Rainbow-DA focuses on both CPU and memory dynamic allocation not only among VMs within a server but also among applications in the entire platform. Rainbow-DA automatically adjusts the allocation threshold (activity) according to the time varying workloads of hosted applications. The working intervals of Rainbow-DA are 1 second for CPU and 5 seconds for memory, which are much smaller than that of Padala's work (10 seconds). This implies that Rainbow-DA has quicker response to the change of the resource requirements by applications. As we all know, the Internet-based applications are interactive with burst workloads. As to such applications, quicker response leads to better quality.

The interval of the resource reallocation depends on workload distributions of the concurrent applications. In our experiments, it ranges from 1 to 1,118 seconds for CPU, and ranges from 5 to 2,555 seconds for memory. Our method leads to the overhead of learning from the feedback and controlling resource reallocation. Learning from the feedback leads to 0.2 percent ($0.2\% * 4\ \mathrm{G} = 8\ \mathrm{M}$) memory and 0 percent CPU overhead. Controlling resource reallocation leads to 0 to 0.3 percent CPU and 0 percent memory overhead per resource reallocation. We can ignore such overhead because it is inappreciable to the system.

7 Discussions

In order to ensure Rainbow works smoothly in the case that workloads are beyond the limits of the available system resources, we may add an admission controller in the application level to drop some new incoming requests. Thus, we propose a global admission control algorithm (AdmisCon-G) in combination with our on-demand resource allocation algorithms. The admission control algorithm drops some new incoming requests for the applications with lower priority to guarantee quality of the accepted requests in our Rainbow, when the total arrival requests surpass the total acceptable requests.

The main idea of AdmisCon-G is as follows: According to the quantity of the resources allocated to each application, AdmisCon-G computes the number of acceptable requests for such application and drops extra requests based on the functions of $Q_{it} = f_i(R_{it}, D_{it})$ gained in Section 5 of our previous work [36].

We evaluate the global admission algorithm using the same experimental scenario as Section 6.2.3. After adding AdmisCon-G in Rainbow-DA, dropping some requests of the web application degrades its performance by up to 3 percent, while the DB and office applications gain improvements in performance by 1 percent in return. It reflects the objective of admission control, that is, dropping some noncritical requests to ensure the quality of the accepted requests when resource requirement by workloads surpasses the resource provided by the platform. In our experiment, the percentage of dropping requests is roughly 1 percent. These experimental results show that the global admission algorithm can be a complement to the on-demand resource allocation algorithms, helping to guarantee the platform work smoothly.

8 Conclusions

This paper proposes a novel two-tiered on-demand resource allocation mechanism with feedback for VM-based data centers. We model the on-demand resource allocation via the K-VM-1-PM problem and the K-VM-N-PM problem. Based on the K-VM-1-PM model, we design a set of algorithms to control the dynamic resource allocation among VMs according to the time-varying resource demands and quality goals of the hosted applications. We evaluate the VM-based shared platform as well as the on-demand resource allocation algorithms using a Xen-based prototype, Rainbow. The experimental results indicate that Rainbow-NDA improves 26 to 324 percent in the application performance as well as 26 percent in the CPU utilization over TSF in the typical enterprise IT environment, while introducing degradation in the case of hosting multiple I/O-intensive applications with high workloads. Such degradation results from their competition for I/O and Xen's I/O overhead. Compared with Rainbow-NDA, the two-tiered on-demand resource allocation in Rainbow further improves in the performance of the critical applications by up to 9 to 16 percent, which is

up to 75 percent of the maximum performance improvement, while introducing at most 5 percent degradations in the performance of others, with up to 5 percent improvements in the resource utilization. These results confirm that the two-tiered on-demand resource allocation gains its goal of improving the resource utilization as well as ensuring quality of the hosted applications.

We only focus on the two-tiered on-demand resource allocation in this paper. However, the application workloads scheduling is also important, in which we simply apply round robin policy to dispatch requests to VMs hosting the application. Our local and global resource schedulers reallocate resources by evaluating the arriving workloads but unaware of the request dispatch of each application, which may result in the mismatch between the on-demand resource allocation and the workloads dispatch. For example, just after some resources of a VM allocating to other VMs, workloads of applications hosted by this VM may arrive continuously and hugely, and vice versa. It cannot achieve the load balance if the workload switch is unaware of the on-demand resources allocation, and this may result in an inefficient usage of the resources and may degrade the quality of applications. Thus, the workload switch and the on-demand resource allocation should cooperate together to manage the workloads and resources in the virtualized computing environment, which is our future work.

In the future, we will propose and evaluate distributed resource allocation algorithms to control on-demand resource allocation among VMs each of which may use resources in different servers concurrently based on the K-VM-N-PM model and our on-going DVMM project. Analyzing the potential of each tier in the resource allocation mechanism is also our future work.

ACKNOWLEDGMENTS

This work was supported in part by the project of NSFC under grants 61202060, 912183001 and 60921002, the National High-Tech Research and Development Program (863) of China under grants 2012AA011003 and 2013AA01A212, and the National Science and Technology Major Project under grant 2011ZX03002-001-01.

REFERENCES

[1] P. Apparao et al., "Characterization & Analysis of a Server Consolidation Benchmark," *Proc. Fourth ACM SIGPLAN/SIGOPS Int'l Conf. Virtual Execution Environments (VEE '08)*, pp. 21-29, Mar. 2008.

[2] K. Appleby et al., "Oceano-SLA Based Management of a Computing Utility," *Proc. IFIP/IEEE Int'l Symp. Integrated Network Management*, pp. 855-868, 2001.

[3] P. Barham et al., "Xen and the Art of Virtualization," *Proc. 19th ACM Symp. Operating Systems Principles (SOSP)*, pp. 164-177, 2003.

[4] H.W. Cain et al., "An Architectural Evaluation of Java TPC-W," *Proc. Seventh Int'l Symp. High-Performance Computer Architecture (HPCA)*, pp. 229-240, 2001.

[5] G. Chen et al., "Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services," *Proc. Fifth USENIX Symp. Networked Systems Design and Implementation (NSDI '08)*, pp. 337-350, 2008.

[6] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the Three CPU Schedulers in Xen," *ACM SIGMETRICS Performance Evaluation Rev.*, vol. 35, no. 2, pp. 42-51, 2007.

[7] I. Cunha et al., "Self-Adaptive Capacity Management for Multi-Tier Virtualized Environments," *Proc. IFIP/IEEE Int'l Symp. Integrated Network Management (IM '07)*, pp. 129-138, 2007.

[8] P. Dubois, "MySQL," *NewRiders*, Dec. 1999.

[9] R.T. Fielding and G. Kaiser, "The Apache HTTP Server Project," *IEEE Internet Computing*, vol. 1, no. 4, pp. 88-90, July 1997.

[10] S. Govindan, A.R. Nath, and A. Das, "Xen and Co.: Communication-Aware CPU Scheduling for Consolidated Xen-Based Hosting Platforms," *Proc. ACM Third Int'l Conf. Virtual Execution Environments (VEE)*, pp. 126-136, 2007.

[11] Helix, http://www.realnetworks.com, 2012.

[12] F. Hermenier et al., "Entropy: A Consolidation Manager for Clusters," *Proc. ACM Int'l Conf. Virtual Execution Environments (VEE '09)*, pp. 41-50, 2009.

[13] M. Hines and K. Gopalan, "Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning," *Proc. ACM Int'l Conf. Virtual Execution Environments (VEE '09)*, pp. 51-60, 2009.

[14] "HP Utility Data Center - Technical White Paper," white paper, Hewlett-Packard, 2001.

[15] Httperf, http://www.hpl.hp.com/research/linux/httperf/, 2012.

[16] IBM Redbook, "Advanced POWER Virtualization on IBM System p5: Introduction and Configuration," Jan. 2007.

[17] G. Jung et al., "Generating Adaptation Policies for Multi-Tier Applications in Consolidated Server Environments," *Proc. Int'l Conf. Autonomic Computing (ICAC '08)*, pp. 23-32, 2008.

[18] M. Kallahalla et al., "SoftUDC: A Software-Based Data Center for Utility Computing," *Computer*, vol. 37, no. 11, pp. 38-46, Nov. 2004.

[19] N. Kandasamy et al., "A Hierarchical Optimization Framework for Autonomic Performance Management of Distributed Computing Systems," *Proc. IEEE 26th Int'l Conf. Distributed Computing Systems (ICDCS)*, p. 9, 2006.

[20] S. Kumar and K. Schwan, "Netchannel: A VMM-Level Mechanism for Continuous, Transparent Device Access during VM Migration," *Proc. ACM Fourth Int'l Conf. Virtual Execution Environments (VEE '08)*, 2008.

[21] D. Kusic et al., "Power and Performance Management of Virtualized Computing Environments via Lookahead Control," *Proc. Int'l Conf. Autonomic Computing (ICAC)*, pp. 3-12, 2008.

[22] H.A. Lagar-Cavilla et al., "SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing," *Proc. Fourth ACM European Conf. Computer Systems (Eurosys)*, pp. 1-12, 2009.

[23] L.S. Lasdon, *Optimization Theory for Large Systems*. Courier Dover, 2002.

[24] Linpack, http://www.netlib.org/benchmark/hpl/, 2012.

[25] LVS, http://www.linuxvirtualserver.org/, 2012.

[26] D.A. Menasc and M.N. Bennani, "Autonomic Virtualized Environments," *Proc. Int'l Conf. Autonomic and Autonomous Systems (ICAS)*, p. 28, 2006.

[27] Microsoft, http://www.microsoft.com/management/, 2012.

[28] M. Nelson et al., "Fast Transparent Migration for Virutal Machines," *Proc. Ann. Conf. USENIX Ann. Technical Conf. (ATC)*, pp. 391-394, 2005.

[29] P. Padala et al., "Automated Control of Multiple Virtual Resources," *Proc. Fourth ACM European Conf. Computer Systems (Eurosys)*, pp. 13-26, 2009.

[30] P. Padala et al., "Adaptive Control of Virtualized Resources in Utility Computing Environments," *Proc. Second ACM European Conf. Computer Systems (Eurosys '07)*, pp. 289-302, 2007.

[31] M. Rosenblum, "VMware's Virtual Platform: A Virtual Machine Monitor for Commodity PCs," *Proc. 11th Hot Chips Conf. (Hot Chips '11)*, 1999.

[32] H. Sandklef, "Testing Applications with Xnee," *Linux J.*, vol. 2004, no. 117, p. 5, Jan. 2004.

[33] A. Shoykhet et al., "Virtuoso: A System for Virtual Machine Marketplaces," Technical Report NWU-CS-04-39, Dept. of Computer Science, Northwestern Univ., July 2004.

[34] Y. Song et al., "A Service-Oriented Priority-Based Resource Scheduling Scheme for Virtualized Utility Computing," *Proc. Int'l Conf. High Performance Computing (HiPC)*, pp. 220-231, 2008.

[35] Y. Song, Y. Zhang, Y. Sun, and W. Shi, "Utility Analysis for Internet-Oriented Server Consolidation in VM-Based Data Centers," *Proc. IEEE Int'l Conf. Cluster Computing and Workshops*, pp. 1-10, 2009.

[36] Y. Song, H. Wang, Y. Li, B. Feng, and Y. Sun, "Multi-Tiered On-Demand Resource Scheduling for VM-Based Data Center," *Proc. IEEE Ninth Int'l Symp. Cluster Computing and the Grid (CCGrid 09)*, pp. 148-155, May 2009.
[37] SPECweb2005, http://www.spec.org/web2005/, 2012.
[38] T. Tannenbaum et al., "Condor - A Distributed Job Scheduler," *Beowulf Cluster Computing with Linux*, Thomas Sterling, ed., pp. 307-350, MIT Press, 2002.
[39] Tomcat, http://tomcat.apache.org/, 2012.
[40] VMware, "Resource Management with VMware DRS," technical paper, 2006.
[41] C.A. Waldspurger, "Memory Resource Management in VMware ESX Server," *Proc. Fifth Symp. Operating Systems Design and Implementation (OSDI '02)*, pp. 181-194, 2002.
[42] J. Wang, Y. Sun, and J. Fan, "Analysis on Resource Utilization Patterns of Office Computer," *Proc. IASTED Int'l Conf. Parallel and Distributed Computing and Systems*, pp. 626-631, 2005.
[43] Q. Wang and D. Makaroff, "Workload Characterization for an E-Commerce Web Site," *Proc. Conf. Centre for Advanced Studies Conf. Collaborative Research (CASCON '03)*, pp. 313-327, 2003.
[44] X. Wang et al., "Appliance-Based Autonomic Provisioning Framework for Virtualized Outsourcing Data Center," *Proc. IEEE Fourth Int'l Conf. Autonomic Computing (ICAC '07)*, p. 29, 2007.
[45] X. Wang et al., "A Resource Management Framework for Multi-Tier Service Delivery in Autonomic Virtualized Environments," *Proc. IEEE Network Operations and Management Symp.*, pp. 310-316, 2008.
[46] Z. Wang et al., "Capacity and Performance Overhead in Dynamic Resource Allocation to Virtual Containers," *Proc. IFIP/IEEE Int'l Symp. Integrated Network Management (IM '07)*, pp. 149-158, 2007.
[47] C. Weng, Z. Wang, M. Li, and X. Lu, "The Hybrid Scheduling Framework for Virtual Machine Systems," *Proc. ACM SIGPLAN/SIGOPS Int'l Conf. Virtual Execution Environments (VEE '09)*, pp. 111-120, 2009.
[48] T. Wood, "Black-Box and Gray-Box Strategies for Virtual Machine Migration," *Proc. Fourth USENIX Conf. Networked Systems Design and Implementation (NSDI)*, 2007.
[49] J. Xu et al., "On the Use of Fuzzy Modeling in Virtualized Data Center Management," *Proc. Int'l Conf. Autonomic Computing (ICAC '07)*, p. 25, 2007.
[50] H. Yu et al., "Understanding User Behavior in Large-Scale Video-on-Demand Systems," *Proc. First ACM SIGOPS/EuroSys European Conf. Computer Systems (EuroSys '06)*, pp. 333-344, 2006.
[51] W. Zhao and Z. Wang, "Dynamic Memory Balancing for Virtual Machines," *Proc. Int'l Conf. Virtual Execution Environments (VEE '09)*, pp. 21-30, 2009.
[52] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster Reserves: A Mechanism for Resource Management in Cluster-Based Network Servers," *Proc. ACM Joint Int'l Conf. Measurement and Modeling of Computing Systems (SIGMETRICS '00)*, pp. 90-101, June 2000.
[53] K. Shen, H. Tang, T. Yang, and L. Chu, "Integrated Resource Management for Cluster-Based Internet Servcies," *Proc. Fifth Symp. Operating Systems Design and Implementation (OSDI '02)*, Dec. 2002.
[54] Y. Song et al., "An Adaptive Resource Flowing Scheme amongst VMs in a VM-Based Utility Computing," *Proc. IEEE Seventh Int'l Conf. Computer and Information Technology (CIT '07)*, pp. 1053-1058, 2007.
[55] X. Fan, W.D. Weber, and L.A. Barroso, "Power Provisioning for a Warehouse-Sized Computer," *Proc. 34th Ann. Int'l Symp. Computer Architecture (ISCA '07)*, pp. 13-23, 2007.
[56] R.P. Doyle, J.S. Chase, and W. Jin, "Model-Based Resource Provisioning in a Web Service Utility," *Proc. Fourth Conf. USENIX Symp. Internet Technologies and Systems (USITS '03)*, 2003.

Ying Song received the PhD degree in computer engineering from the Institute of Computing Technology (ICT), Chinese Academy of Sciences. She is an assistant professor at the State Key Laboratory of Computer Architecture at ICT. Her work thus far has covered topics such as performance modeling, resource management, and cloud computing. Her main research interests include computer architecture, parallel and distributed computing, and virtualization technology. She has authored or coauthored more than 10 publications in these areas since 2007, and she served in various academic conferences.

Yuzhong Sun received the PhD degree in computer engineering from the Institute of Computing Technology (ICT), Chinese Academy of Sciences. He is a professor in the State Key Laboratory of Computer Architecture at ICT in Beijing, China. His research interests focus on distributed system software and computing/programming models. He has authored and coauthored more than 50 publications, and he has served in various academic conferences and journals. He is a member of the IEEE and the IEEE Computer Society.

Weisong Shi received the PhD degree in computer engineering from the Chinese Academy of Sciences in 2000. He is an associate professor of computer science at Wayne State University. His current research focuses on computer systems, mobile computing, and high-performance computing. He has published more than 100 peer-reviewed journal and conference papers in these areas. He has also served on technical program committees of several international conferences, including WWW, ICDCS, and MASS. He was a recipient of the Microsoft Fellowship in 1999, the President Outstanding Award of the Chinese Academy of Sciences in 2000, one of 100 outstanding PhD dissertations (China) in 2002, the Faculty Research Award of Wayne State University in 2004 and 2005, and the Best Paper Award of ICWE '04 and IPDPS '05. He is a recipient of the US National Science Foundation (NSF) CAREER award and Wayne State University Career Development Chair award. He is a senior member of the IEEE.