A Service-Oriented Priority-Based Resource Scheduling Scheme for Virtualized Utility Computing

Ying Song[1,2,3], Yaqiong Li[1,2,3], Hui Wang[1,2], Yufang Zhang[1,2,3], Binquan Feng[1,2,3], Hongyong Zang[1,2,3], and Yuzhong Sun[1,2]

[1] Key Laboratory of Computer System and Architecture,
Institute of Computing Technology, Chinese Academy of Sciences, China
[2] National Research Center for Intelligent Computing Systems, ICT, China
[3] Graduate University of Chinese Academy of Sciences, China
songying@ncic.ac.cn

Abstract. In order to provide high resource utilization and QoS assurance in utility computing hosting concurrently various services, this paper proposes a service computing framework-RAINBOW for VM(Virtual Machine)-based utility computing. In RAINBOW, we present a priority-based resource scheduling scheme including resource flowing algorithms (RFaVM) to optimize resource allocations amongst services. The principle of RFaVM is preferentially ensuring performance of some critical services by degrading of others to some extent when resource competition arises. Based on our prototype, we evaluate RAINBOW and RFaVM. The experimental results show that RAINBOW without RFaVM provides 28%~324% improvements in service performance, and 26% higher the average CPU utilization than traditional service computing framework (TSF) in typical enterprise environment. RAINBOW with RFaVM further improves performance by 25%~42% for those critical services while only introducing up to 7% performance degradation to others, with 2%~8% more improvements in resource utilization than RAINBOW without RFaVM.

Keywords: Resource scheduling, utility computing, virtualization.

1 Introduction

It's a new trend towards providing heterogeneous services concurrently by enterprise data centers, for example, of Google and Amazon. Google provides services consisting of Google search, Google office, and Youtube. In the past, those services were provided by different platforms. In such a case, QoS guaranteeing of services with the time-varying capacities (including computing, storage, and communication) demands as the result of request arrival distributions resulted in over-provision each service. Such datacenters were often underutilized. One approach to increase the resource utilization is consolidating services in a shared infrastructure-utility computing [2]. In such a shared platform, isolation among the hosted services is crucial. As virtualization is increasingly popular, utility computing is incorporating virtualization technology such as virtual machines (VMs) with effective isolation among services. Many companies envisioning

P. Sadayappan et al. (Eds.): HiPC 2008, LNCS 5374, pp. 220–231, 2008.

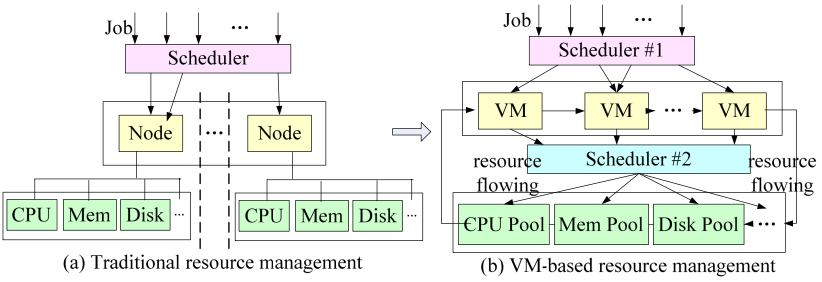


Fig. 1. The evolution of resource management

this popular trend have devoted themselves to developing new utility computing infrastructures based on virtualization technologies such as VMware [17] and XenSource [11].

VM-based utility computing has some obvious advantages like consolidation, isolation, flexibility resource management. Dynamic load changes of services give rise to dynamic capacities demands, which implies it is necessary to control ***resource flowing*** among services on-demand. Resource flowing denotes the process in which resources released by VMs are allocated to others. The VM-based resource management differs from the previous works in the granularity (from nodes to components) and dimensions (from one to two), illustrated in fig.1. Traditional resource management corresponds to the scheduler in fig.1(a), which dispatches jobs onto a set of exclusively servers. As to the VM-based resource management (fig.1(b)), scheduler #1 corresponding to the traditional resource management dispatches jobs onto a set of VMs. It adds a new dimensioned resource scheduler (scheduler #2) to optimize the usage of finer-grained resources via resource flowing among VMs. Any contemporary VMMs (Virtual Machine Monitor, i.e. Xen and VMware) with resource reallocating scheme provide technical support rather than strategy to resource flowing. They need better scheduler#2 to optimize the usage of resources. Optimizing resource flowing among VMs is a challenge in such a platform.

In order to address above challenge, we propose a novel service computing framework --RAINBOW for VM-based utility computing. RAINBOW integrates separate resources into a shared virtual computing platform, ensuring QoS and higher resource utilization. We model the resource flowing using optimization theory. Based on this model, we present a priority-based resource scheduling scheme including a set of algorithms of resource flowing among VMs (RFaVM). The principle of RFaVM is preferentially ensuring performance of some critical services by degrading of others to some extent when resource competition arises. We implement a Xen-based prototype to evaluate RAINBOW and RFaVM. We consider CPU and memory flowing schemes which could be generally applicable to other resources. The experimental results indicate that RAINBOW with RFaVM effectively improves the resource utilizations, and meets QoS goals of services, in the typical enterprise IT environment with inappreciable overheads.

This paper has the following contributions. 1) We propose a novel service computing framework (RAINBOW) for VM-based utility computing. 2) We model

the resource flowing in RAINBOW and present a priority-based resource scheduling scheme including a set of algorithms of resource flowing among VMs (RFaVM).

The rest of this paper is organized as follows: RAINBOW is introduced in Section 2. Section 3 describes resource scheduling scheme, while Section 4 discusses the experimental results. Section 5 presents related work. Section 6 concludes the paper.

2 A Novel Service Computing Framework – RAINBOW

2.1 RAINBOW Statement

We present a novel service computing framework (RAINBOW, illustrated in fig.2(b)) to improve the resource utilization and QoS. Different from the traditional service computing framework (TSF, illustrated in fig.2(a)) in which one service runs on a set of dedicated servers, RAINBOW uses virtualization to isolate concurrent services in a shared physical infrastructure. We observe that diverse services may be various resource-bound, for example, VoD service is I/O-bound, HPC service is CPU-bound. Further, we obtain another observation that diverse services may have various time-varying resource demands as the result of request arrival distributions [4][13]. Those two observations motivate our design of RAINBOW. In order to minimize the interaction among the hosted services due to their competitions for resources, services with the same resource-bound should be distributed onto different physical servers. In RAINBOW, a set of VMs serving a particular service is called a ***group***. The key principle is that VMs belonging to a single group may be split onto multiple physical

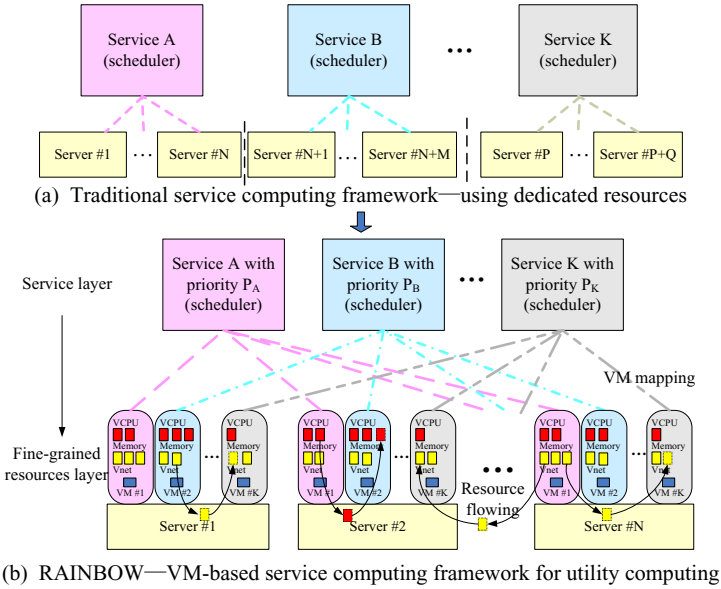


Fig. 2. The evolution of service computing framework

servers, while each server hosts VMs belonging to different groups. Each service dispatches workloads to VMs in its group according to its scheduling algorithms. RAINBOW provides resources to the hosted services on-demand via resource flowing taking the priority of service into account. This allows better resource utilization and QoS at the service level compared to previous proposals [12] in scenarios where there is competition for the same resource by similar service components.

Resource flowing strategy should solve four problems. 1) Which resource will flow? 2) When will such resource flow? 3) Which VMs will be the source and target of flow? 4) How many resources will flow? In order to answer these four problems, we model the resource flowing in RAINBOW first.

2.2 Resource Flowing Model

We consider the problem of resource flowing among VMs in RAINBOW, and model it by optimization theory. This model is a general one which can be respectively used by CPU, memory or other resources. Based on the model, we present a set of algorithms of resource flowing amongst VMs (RFaVM) to provide resources to the hosted services on-demand. First we introduce the following notations and concepts:

- R - The total CPU, memory or other resources in a server.
- K - The number of VMs resided in the server.
- $C_{i\text{-}min}$ - The minimum threshold of resources allocated to VM_i, which is used to avoid huge interaction among VMs when competition for resources arises. It is set by experience in our experiments, and will be justified in the near future.
- R_{it} - Resources allocated to VM_i at time t. It obeys: $R \geq \sum_{i=1}^{K} R_{it}$ and $R_{it} \geq C_{i\text{-}min} > 0$.
- D_{it} - Resources demands by VM_i at time t. It is proportional to the requests arrival rate.
- SP_i - The static priority of service hosted in VM_i. It indicates how critical the requirement for QoS. If $i>j$, $SP_i \geq SP_j$. SP_i is determined by administrator.
- Φ_i - The tolerable QoS threshold of service hosted in VM_i.
- Q_{it} - Quality of service hosted in VM_i at time t. The smaller the Q_{it} is, the better QoS the service gains. As we all know QoS (i.e. response time) is decided by the required and allocated resources, namely, $Q_{it}=f_i(R_{it},D_{it})$. The relationships can be found by studying the typical services, which is our ongoing work. In order to fairly weight various services using their QoS, we use the rate of QoS of service hosted in VM_i at time t and of the tolerable QoS (Q_{it}/Φ_i, ***QoS-rate*** for short).

The goal of resource flowing is to optimize QoS taking the priority 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 resource flowing. In order to provide resource flowing with a utility function, which maps QoS of the target to a benefit value, we define the utility function UF_t[1]. UF_t is related to the static priorities and *QoS-rates* of the hosted services. The resource flowing is how to control resource allocation among VMs with the goal of minimizing the utility function UF_t, giving the limited resources, formulated as formula (1).

We simplify the above model to design our resource flowing algorithms. All the functions $f_i(R_{it},D_{it})/\Phi_i$ are set as: 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 relationship between D_{it} and R_{it} can be directly reflected by the resource utilization of VM_i. Thus, our algorithms control resource flowing according to the resource utilization of each VM and static priority of each service.

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

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

3 Priority-Based Resource Scheduling Scheme

We consider CPU and memory flowing algorithms. We assume that there are *K* VMs hosting various services in a server. The efficacy of our algorithms is intimately dependent on how well it can predict resource utilization. We take a simple low-overhead last-value-like prediction as reference [15] does, in which the resource utilization during the last interval is used as a predictor in the next interval.

3.1 CPU Flowing Amongst VMs Algorithm (CFaVM)

The hypervisor (Xen) we used provides a credit scheduler that can operate in two modes: capped and non-capped. We select the non-capped mode to prevent from the degradation of CPU utilization. In the non-capped mode, if multiple VMs require idle processing units, the scheduler distributes idle processing units to the VMs in proportion to their weights. There is no support on automatically changing weights of VMs according to its workloads and QoS. CFaVM uses a hybrid priority scheme to adjust the dynamic priority (weight) of each VM according to its static priority and resource utilization. Some VMs with high priority are preferentially guaranteed with the rapidly increased dynamic priorities, while others suffer performance degradations via the slowly increased dynamic priorities when competition for CPU arises.

W refers to the weight assigned to a VM, *OW* refers to the weight of a VM in the prior interval, $\triangle W$ means the increased weight when a VM is CPU overload. We fix W_1 to avoid frequent changes of W_i when CPU competitions arise. The changes of other W_i alter the CPU share of VM_1. We define $\triangle W$ to be in proportion to *SP*. Formula (2) gives the relationship of $\triangle W_i$ and $\triangle W_j$. We define the minimum CPU resources allocated to VM_i ($C_{i\text{-}min}$), which could be denoted by the maximum ($W_{i\text{-}max}$) and minimum ($W_{i\text{-}min}$) weight thresholds assigned to VM_i. We define $W_{j\text{-}max}$ to be in proportion to SP_j as formula (3).

$$\triangle W_i = (SP_i/SP_j) * \triangle W_j. \tag{2}$$

$$W_{j\text{-}max} = (SP_j/SP_j) * W_{j\text{-}max}. \tag{3}$$

Input: U_i, OW_i, i=1,…,K; **Output:** W_i, i=1,…,K;
Algorithm:
Initialize OW_i (i=1,…,K) to be 256. SP_i, $\triangle W_i$, $W_{i\text{-}max}$, $W_{i\text{-}min}$ (i=1,…,K) are initialized.
While (1)
{ For (i=1,i<=K,i++)
{
1) If (U_i=100%) and ($OW_i+\triangle W_i<W_{i\text{-}max}$)
Then $W_i=OW_i+\triangle W_i$;
Else if (U_i>=T𝕦) and ($OW_i<W_{i\text{-}max}$) and($OW_i+\triangle W_i>=W_{i\text{-}max}$)
Then $W_i=W_{i\text{-}max}$;
2) If U_i<T𝕕 Then { $W_i = \frac{OW_i \times U_i \times (\sum_{j=1}^{K} OW_j - OW_i)}{T_d \times (\sum_{j=1}^{K} OW_j) - OW_i \times U_i}$
If $W_i<W_{i\text{-}min}$
Then $W_i=W_{i\text{-}min}$;
}
} sleep(1);
} End;

Fig. 3. CFaVM

Based on the CPU utilization of each VM, CFaVM (fig.3) determines whether the CPU is overload in a VM or not. We choose T_u as the threshold of CPU overload and T_d as the desired CPU utilization. If the CPU utilization of VM_i reaches T_u, CPU resources should flow in VM_i (we call it a ***consumer VM***). CFaVM increases $\triangle W_i$ on the weight of VM_i to increase CPU resources. Other VMs will be the ***provider VMs***, which flow out CPU in proportion to their weights. If the CPU utilization of VM_j is lower than T_d, CPU should flow out from VM_j. The weight of VM_j is decreased to decrease CPU resources. In order to let the CPU utilization of VM_j reach T_d, the new weight of VM_j is calculated as formula (4).

$$\because \frac{OW_j}{\sum_{i=1}^{K} OW_i} \times R \times U_j = \frac{W_j}{W_j + \sum_{i=1}^{K} OW_i - OW_j} \times R \times T_d \quad \therefore W_j = \frac{OW_j \times U_j \times (\sum_{i=1}^{K} OW_i - OW_j)}{T_d \times (\sum_{i=1}^{K} OW_i) - OW_j \times U_j} \tag{4}$$

3.2 Lazy Memory Flowing amongst VMs Algorithm (LMFaVM)

In LMFaVM, *M* refers to the total memory which could be used by the *K* VMs. Each VM is initially allocated *M/K* memory. The minimum memory threshold of VM_i ($M_{i\text{-}min}$), corresponding to $C_{i\text{-}min}$ in section 2.2, is defined as formula (5).

$$M_{i-\min} = (SP_i / \sum_{j=1}^{K} SP_j) * M / K \tag{5}$$

$$\triangle M_i = (SP_i/SP_1) * \triangle M_1 \tag{6}$$

Based on the static priority and the collected idle memory of VM_i (IM_i), LMFaVM (fig.4) determines whether there is memory overload in a VM or not. If idle memory of each VM is higher than the threshold Ψ, no memory flows. If IM_i is lower than Ψ, LMFaVM increases $\triangle M_i$ (illustrated as formula (6)) or less memory to VM_i (consumer VM), as long as there is another VM (provider VM) which can give its memory to VM_i. A VM can be a provider when its memory is more than its minimum

```
Input: IMi, Mi, i=1,…,K;  Output: Mi, i=1,…,K;
Algorithm:
Initialize Mi (i=1,…,K) as M/K. SPi,△Mi and Mi-min (i=1,…,K)are initialized.
While (1)
{   For (i=1,i<=K,i++)
      {   If (IMi<Ψ)
          {     1) select VMx with maximum E from VMs which could give their memory to VMi;
                2) control  min{△Mi,Mx-Mx-min} memory flow from VMx  to VMi;
          }
      }  sleep(T);
}; End;
```

Fig. 4. LMFaVM

memory threshold and either of the two conditions is satisfied: 1) it has idle memory; 2) its priority is lower than that of the consumer VM. We define *E* to be the rate of *IM* and *SP* (*E=IM/SP*) to select the provider VM. If memory of a VM is overload, LMFaVM compares *E* of VMs which can be the provider, and selects the VM with maximum *E* to be the final provider VM.

4 Implementation and Performance Evaluation

The implementation of our prototype is based on Xen. In our servers pool, there are four servers each of which has two 2190MHz Dual Core AMD Opteron(tm) processors with 1024KB of cache and 4GB of RAM. We use CentOS4.4, and Xen-3.0.4. We use other four machines to be clients of services. The systems are connected with a Gigabit Ethernet. Using our prototype, we do a set of experiments to evaluate RAINBOW and RFaVM. The experiments are classified into two groups: Group-I evaluates RAINBOW without resource flowing ('***RAINBOW-NF***' for short); Group-II verifies RFaVM in RAINBOW.

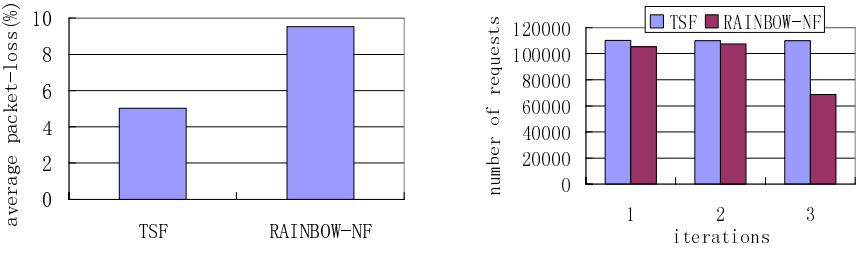
Group-I: We evaluate the strengths and weaknesses of our RAINBOW-NF using the following two comparisons with various scenarios.

Comparison-I: We compare RAINBOW-NF with TSF using the following three typical enterprise services: web, HPC, and office services. On each server, we create 3 VMs. Each VM is allocated 1 VCPU and 1GB memory. VMs devoted to a service are distributed onto the four servers. Apache [14], LVS [19] with Round Robin algorithm, and the e-commerce workloads of SPECWeb2005 [22] are used for the web server. We use Condor [16] to dispatch Linpack [20] jobs to the HPC VMs. The office service is provided by our office applications virtualization product. It separates the display from the running of applications based on virtual desktop, and it distributes applications to servers according to workloads in these servers. We use Xnee [3] to emulate the real-world office applications based on the trace collected by [6].

As to the office service, we collect the starting up time of eight applications (FTP, mozilla, openoffice, etc.) for 4 times as the performance metric which is captured by the modified VNC. We compare the average starting up time of all the applications. The number of requests with good response (respond within 3 seconds) defined by SPECWeb, is the performance metric of the web service. The HPC service is evaluated by throughput (the number of completed linpack jobs during 3 hours). The

experiment results show that RAINBOW-NF provides dramatic improvements both in service performance (28%~324%) and in the CPU utilization (26%) over TSF.

Comparison-II: We evaluate RAINBOW-NF using two I/O-bound services (VoD and web). Helix [21] is used for the VoD server and we make the VoD workloads generator according to ref[4]. Figure 5(a)(b) show the comparisons of TSF and RAINBOW-NF by the two services. From these figures we can see that service performance is degrading by RAINBOW-NF, especially for the web service in its third iteration when the VoD service reaches its peak of workloads.



(a) The VoD service (b) The web service

Fig. 5. Comparisons between TSF and RAINBOW

Analysis and Conclusion on Group-I: In Comparison-I, since the three services are different resource-bound, their resource bottlenecks are different. Distributing the same service onto multiple physical servers reduces the demands on its bottleneck resources on each server. On the other hand, each server hosts concurrently multiple services with diverse resource-bound, which avoids frequent competitions for the same resource. Thus, RAINBOW-NF provides huge improvements. In Comparison-II, the two services are I/O-bound. Frequent competitions for I/O, as well as Xen's I/O overhead, leads to huge interactive impacts between services in RAINBOW.

Group-II: We verify RFaVM using the same experimental scenario and benchmarks as in the Comparison-I of Group-I. On each physical server, we create 3 VMs. We initially allocate various resources to VMs (table 1) to provide different conditions (BN denotes the bottleneck resources when services reach their peaks of workloads).

Table 1. Initial resources allocation to VMs on various conditions

BN	CPU	memory
CPU	1 VCPU (VCPUs of the 3 VMs running on the same server are pinned to the same physical core)	1G
memory	1 VCPU (mapping to all the physical cores)	600M
CPU& mem	1 VCPU (the same as BN=CPU)	600M

The baseline system RAINBOW-NF provides static resources allocation to VMs. 'CFaVM' refers to RAINBOW adopting CPU flowing algorithm. 'LMFaVM' uses memory flowing algorithm in RAINBOW. 'RFaVM' adds CFaVM and LMFaVM.

SP_{office}:SP_{web}:SP_{hpc} is initialized as 4:3:1. We choose 50MB idle memory as the threshold of memory overload, 90% as the threshold of CPU overload (T_u), and 70% as the desired CPU utilization (T_d) for VMs. These parameters are decided by our experience. Table 2 illustrates the comparisons results on various conditions (***A:B*** denotes that *A* compares with *B*). From these results we can see that RFaVM provides great improvements (42%) in performance of the office service and tiny improvements (2%) in performance of the web service, while introducing small impairments (up to 7%) in performance of the HPC service. This is the result that RFaVM preferentially ensures performance of the office service by degrading of others to some extent, and it gives preference to the web service when the web service competes for resources with the HPC service.

Table 2. Various scheduling algorithms in RAINBOW vs. RAINBOW-NF ('NF' for short)

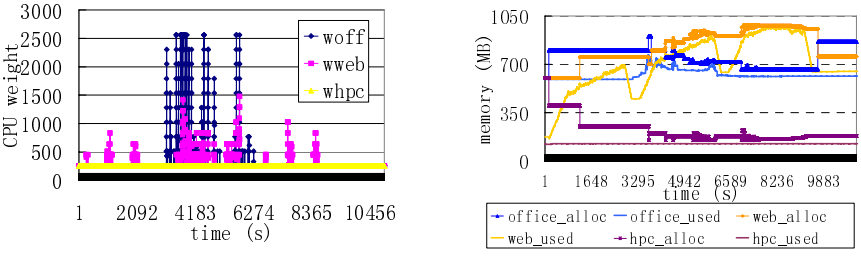
BN	Comparisons	Office	web	HPC	Resource utility
CPU	CFaVM: NF	25%	1%	-7%	CPU:2 %
mem	LMFaVM: NF	37%	1%	1%	mem:8%
CPU&mem	RFaVM: NF	42%	2%	1%	CPU:2%; mem:6%

We compare our resource flowing in RAINBOW ('RAINBOW' for short) with ref[12] in table 3. Ref[12] only focuses on CPU reallocation, while RAINBOW focuses on both CPU and memory flowing. The working intervals of RAINBOW are 1s for CPU and 5s for memory, which are smaller than that of ref[12] (10s). This implies that RAINBOW has faster response to the change of resource requirements by services. We compute the improvement and degradation introduced by ref[12] using its fig.14-15 of response time with and without their controller. The improvement provided by ref[12] (28%) is smaller than that provided by RAINBOW (42%) for the critical service. The degradation introduced by ref[12] (41%) is much larger than that introduced by RAINBOW (7%) for other services. These results imply that our scheme is better than ref[12] in the aspect of assuring QoS.

Table 3. RAINBOW vs. ref[12]

	resources	working interval	Improvement	Degradation
ref[12]	CPU	10s	28%	41%
RAINBOW	CPU&mem	1s(CPU), 5s(mem)	42%	7%

We use CPU flowing as an example of analyzing the reason why RFaVM further improves performance of the critical services by slightly impairing of others using fig.6(a)-(b) and fig.7. Figure 6(a)-(b) illustrate the behaviors of adjusting CPU weights and memory of VMs controlled by RFaVM. In fig.6(a), *woff* denotes weight of office VM, *wweb* and *whpc* are similar. Figure 6(b) illustrates memory allocated to services (denoted by service_alloc) adopting RFaVM and memory used by services (denoted by service_used). Figure 7 shows the CPU requirement by workloads of each service. From fig.7 we can see that in the first 3000 seconds (3000s for short) and after 7000s, competitions for CPU arise between web and HPC services. The



(a) CPU weights of VMs (b) Memory owned&used by VMs

Fig. 6. Behaviors of resources flowing using RFaVM

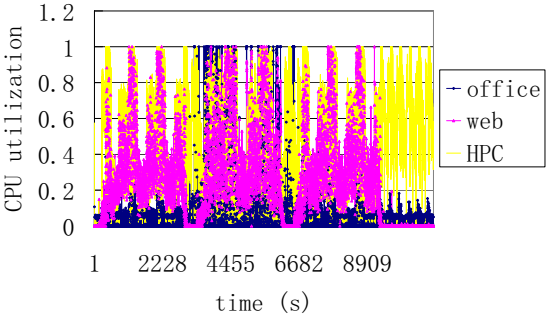


Fig. 7. CPU utilization of VMs

priority of web service is higher than that of HPC service. RFaVM controls CPU flow to the web VM (fig.6(a)). From 3000s to 7000s, office service with the highest priority requires more CPU than in other periods. Thus RFaVM controls CPU flow to the office VM when CPU competitions arise, resulting in more CPU allocated to the office VM in this period than in other periods (fig.6(a)). The priority of HPC service is the lowest one. RFaVM controls CPU flow out from the HPC VM when CPU competitions arise, which results in less CPU allocated to the HPC VM than others (fig.6(a)). As the result, CPU flowing improves the performance of office and web services by impairing that of HPC service.

In our experiments, the interval of resource flowing ranges from 1s to 1118s for CPU, and ranges from 5s to 2555s for memory. Our scheme leads to the overhead of collecting information and controlling resource flow. Collecting information leads to 8M memory overhead. Each resource flow leads to 0%~0.3% CPU overhead. Such overhead can be ignored since it is inappreciable to the system.

5 Related Work

Currently, a large body of papers is on managing data center in a utility computing environment that provides on-demand resources, such as HP's SoftUDC [9], IBM's

PLM [5], and VMware DRS [17]. To the best of our knowledge, no other studies proposed the same service computing framework and scheduling algorithms as ours.

Several studies provide on-demand resources at the granularity of servers in the data center. Oceano [8] dynamically reallocates whole servers for an e-business computing utility. SoftUDC [9] proposes a software-based utility data center. It adopts the strategy of on-the-fly VM migration, which is also implemented by VMware's VMotion [10], to provide load balancing. Ref[18] dynamically allocates resources to applications via adding/removing VMs on physical servers. They are in contrast to our scheme that controls resource allocation at the granularity of CPU time slots and memory.

There is a growing body of work on providing on-demand fine-grained resources in the shared data center. IBM's PLM [5] and VMware DRS [17] dynamically allocate resources to partitions/VMs according to shares specified statically and resource utilization, ignoring QoS. In [7], a two-level resource management system with local controllers at the VM level and a global controller at the server level is proposed. Ref[12] dynamically adjusts CPU shares to individual tiers of multi-tier applications in the VM-based data center. They choose the design where one server hosts multiple VMs which provide the same service. This is different from our RAINBOW that puts various services hosted in VMs onto the same physical server and distributes the same service onto different physical servers. Ref[12] uses the CPU capped mode to provide performance isolation between VMs, which decreases QoS and resource utilization. Based on the resource flowing model, our scheme focuses on both CPU and memory flowing and uses the CPU non-capped mode to achieve better QoS than [12].

6 Conclusion

This paper presents a novel service computing framework (RAINBOW) for VM-based utility computing. In RAINBOW, a priority-based resource scheduling scheme including a set of algorithms of resource flowing amongst VMs (RFaVM) is proposed according to the time-varying resources demands and QoS goals of various services. We implement a Xen-based prototype to evaluate RAINBOW and RFaVM. The results indicate that RAINBOW without RFaVM gains improvements in the typical enterprise IT environment (improves 28%~324% in service performance, as well as 26% in CPU utilization over TSF), while introducing degradation in the case of hosting multiple I/O-bound services, which results from competitions for I/O and Xen's I/O overhead. RFaVM further improves performance by 25%~42% for those critical services while only introducing up to 7% performance impairment to others, with 2%~8% improvements in average physical resource utilization than RAINBOW without RFaVM. Compared with ref[12], our resource scheduling is better in assuring QoS. Such results indicate that RFaVM gains its goals of service differentiation as well as improvements in resource utilization.

Some inviting challenges remain in this research. The parameters in our algorithms will be justified in the near future. In order to ensure RAINBOW work smoothly, admission control in RAINBOW is still open.

Acknowledgments. This work was supported in part by the National High-Tech Research and Development Program (863) of China under grants 2006AA01Z109, 2006AA01A102, and 2007AA01Z119, China's National Basic Research and Development 973 Program (2006CB303106), and the projects of Natural Science Foundation of China (NSFC) under grants 90412013.

References

[1] Menascé, D.A., Bennani, M.N.: Autonomic Virtualized Environments. In: ICAS 2006, p. 28 (2006)
[2] Hewlett-Packard: HP Utility Data Centre – Technical White Paper (October 2001), http://www.hp.com
[3] Sandklef, H.: Testing applications with Xnee. Linux Journal 2004(117), 5 (2004)
[4] Yu, H., Zheng, D., Zhao, B.Y., et al.: Understanding User Behavior in Large-Scale Video-on-Demand Systems. In: EuroSys 2006, pp. 333–344 (2006)
[5] IBM Redbook: Advanced POWER Virtualization on IBM System p5: Introduction and Configuration (January 2007)
[6] Wang, J., Sun, Y., Fan, J.: Analysis on Resource Utilization Patterns of Office Computer. In: PDCS 2005, pp. 626–631 (2005)
[7] Xu, J., Zhao, M., et al.: On the Use of Fuzzy Modeling in Virtualized Data Center Management. In: ICAC 2007, p. 25 (2007)
[8] Appleby, K., et al.: Oceano-SLA based management of a computing utility. In: Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, pp. 855–868 (2001)
[9] Kallahalla, M., Uysal, M., Swaminathan, R., et al.: SoftUDC: A Software-Based Data Center for Utility Computing, November 2004, pp. 38–46. IEEE Computer Society, Los Alamitos (2004)
[10] Nelson, M., Lim, B.H., Hutchins, G.: Fast Transparent Migration for Virutal Machines. In: 2005 USENIX Annual Technical Conference, pp. 391–394 (2005)
[11] Barham, P., Dragovic, B., et al.: Xen and the art of virtualization. In: SOSP, pp. 164–177 (2003)
[12] Padala, P., Zhu, X., Uysal, M., et al.: Adaptive Control of Virtualized Resources in Utility Computing Environments. In: EuroSys 2007, pp. 289–302 (2007)
[13] Wang, Q., Makaroff, D.: Workload Characterization for an E-commerce Web Site. In: Proc. CASCON 2003, pp. 313–327 (2003)
[14] Fielding, R.T., Kaiser, G.: The Apache HTTP Server Project. IEEE Internet Computing 1(4), 88–90 (1997)
[15] Govindan, S., Nath, A.R., Das, A.: Xen and Co.: Communication-aware CPU scheduling for consolidated Xen-based hosting platforms. In: VEE, pp. 126–136 (2007)
[16] Tannenbaum, T., Wright, D., Miller, K., et al.: Condor - A Distributed Job Scheduler. In: Sterling, T. (ed.) Beowulf Cluster Computing with Linux, pp. 307–350. The MIT Press, Cambridge (2002)
[17] VMware Infrastructure: Resource Management with VMware DRS
[18] Wang, X., Lan, D., Wang, G., et al.: Appliance-based Autonomic Provisioning Framework for Virtualized Outsourcing Data Center. In: ICAC 2007, p. 29 (2007)
[19] http://www.linuxvirtualserver.org/
[20] http://www.netlib.ort/benchmark/phl/
[21] http://www.realnetworks.com
[22] http://www.spec/org/web2005/