Dynamic Processor Resource Configuration in Virtualized Environments

Hai Jin, Li Deng, Song Wu, Xuanhua Shi
Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, 430074, China
Email: hjin@hust.edu.cn

Abstract—**Virtualization can provide significant benefits in data centers, such as dynamic resource configuration, live virtual machine migration. Services are deployed in virtual machines (VMs) and resource utilization can be greatly improved. In this paper, we present VScheduler, a system that dynamically adjusts processor resource configuration of virtual machines, including the amount of virtual resource and a new mapping of virtual machines and physical nodes. VScheduler implements a two-level resource configuration scheme – local resource configuration (LRC) for an individual virtual machine and global resource configuration (GRC) for a whole cluster or data center. GRC especially takes variation tendency of workloads into account when remapping virtual machines to physical nodes. We implement our techniques in Xen and conduct a detailed evaluation using RUBiS and dbench. The experimental results show that VScheduler not only satisfies resource demands of services, but also reduces the number of virtual machines migration, which can provide a stable VM distribution on physical nodes in data centers.**

Keywords-**Service; Virtualization; Virtual Machine; Live Migration; Resource Configuration; Data Center**

I. Introduction

The workloads of services usually vary with time, while traditional resource allocation is only done statically. Thus, execution environments are often forced to be over-provisioned based on anticipated peak demands, inevitably resulting in substantial wasted resources besides additional consumed power. The average resource utilization is typically below 15% - 20% [1].

In recent years, system virtualization technologies bring a dramatic change to data centers by offering the benefits of resource-efficiency, security, and power-saving [2][3]. Services can be deployed in virtual machines (VMs) [4][5]. These virtual machines are isolated from each other but can multiplex the same underlying hardware resources. Because virtualization decouples the dependencies between application software platform and hardware, the ability of dynamic resource management in data centers becomes strong.

Sandpiper [6] was proposed to automatically monitor system resource usage, detect hotspots, and tackle resource deficits on physical servers using live migration of virtual machine. Entropy resource manager for homogeneous clusters [3] was presented to perform dynamic VM consolidation based on constraint programming to make the number of active physical nodes minimum. A management framework for a virtualized cluster system [7] implemented performance tuning to achieve minimum costs in a virtual cluster. However, these systems did not take variation tendency of dynamic workloads into account especially when redistributing VMs on nodes, which might lead to frequent VM migration.

In this paper, we present VScheduler to implement dynamic resource configuration for services in virtualized environments. VScheduler complements the capacity planning and virtual machine migration technologies to mitigate processor resource contentions both inside and outside of virtual machines. The main contributions of the paper are listed as follows: 1) we present a two-level resource configuration scheme to effectively mitigate resource deficits of services in virtualized environments – one level *LRC* for an individual virtual machine and the other *GRC* for the remapping of virtual machines to physical nodes; 2) the global resource configuration method takes variation tendency of workloads into account and finds a stable distribution of VMs on physical nodes.

We have implemented VScheduler prototype system based on Xen [8]. To test our system, we have built a testbed for a virtual data center using Xen. A detailed experimental evaluation is conducted on the testbed in various scenarios using RUBiS benchmarks [9]. Experimental results show that VScheduler not only satisfies resource demands of services, but also reduces the number of virtual machines migration, which provides a stable VM distribution on physical nodes in data centers.

The remainder of paper is organized as follows. Virtualized resource management is discussed in section II. Section III describes the two-level processor resource configuration scheme in detail. In section IV we present the implementations of VScheduler prototype system. In section V we describe our evaluation methodology and present the experimental results. Finally, section VI summarizes our contributions and outlines our future work.

II. Virtualized Resource Management

System virtualization provides to the upper layer the abstraction of the underlying hardware — a complete system
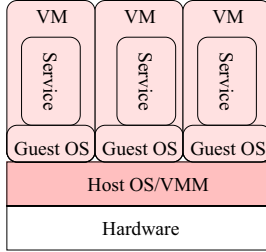
32

Figure 1: System virtualization model

platform which an operating system can run on. The software layer providing resource virtualization is called virtual machine monitor (VMM). The VMM runs on bare hardware or on top of an operating system, while virtual machines (VMs) run on the VMM. A system virtualization model is depicted in Fig. 1.

Guest operating systems in virtual machines use virtualized resource, while VMM is responsible for mapping virtualized resource to physical resource. Usually, there is a privileged domain named *domain 0* [8] on VMM which is responsible for managing other VMs and their virtual devices. In a cluster or data center, each physical node runs a VMM and one or more VMs. Services are deployed in virtual machines.

Virtualization weakens the inter-dependence between operating systems and hardware. It has been an effective way to dynamic resource configuration. Usually, virtual resource management can be done in three levels:

Application-Level Resource Management: This is a distributed resource management method. It is virtual machines themselves who are responsible for the efficiency and fairness of resource allocation across VMs. A "friendly" VM (FVM) [10] is proposed to adjust its demands for system resources. Virtual machine itself should know its resource usage status best. However, because multiple VMs share the same underlying hardware, they should consult with each other about the share of resource in a friendly manner.

Host-Level Resource Management: This is a centralized resource management approach. It is central authority – host kernel or VMM who allocates resources to virtual machines residing on the host. For processor resource, multiple schedulers have been used in Xen [11], such as borrowed virtual time (BVT) [12], simple earliest deadline first (SEDF) [13] and credit scheduler [14]. For memory resource, MEmory Balancer (MEB) [15] is proposed to dynamically monitor the memory usage of each virtual machine, accurately predict its memory needs and periodically reallocate host memory.

Cluster-Level Resource Management: This is a centralized resource management method too. There is a control center that is responsible for resource management in a cluster. As virtualized clusters or data centers enable the sharing of resources among all the hosted applications, control center

can dynamically adjust resource allocation of virtualized nodes on demand. Virtual machines can be migrated freely and smoothly from one node to another, which facilitates system maintenance, load balancing, fault tolerance, and power-saving. Live migration of virtual machines has been a strong resource management tool in clusters or data centers.

Sandpiper [6] is proposed to balance the workloads of physical nodes by using live VM migration. It employs a First-Fit-Decreasing (FFD) heuristic to relocate VMs from overloaded nodes to under-utilized ones. Based on control theory, resource control systems are designed and implemented [16][17] to adapt to dynamic workload changes to achieve application-level quality of service or service-level objectives of applications. A management framework is presented [7] to automatically tune performance for a virtualized cluster.

III. PROCESSOR RESOURCE CONFIGURATION SCHEME

Resource demands of services change with the fluctuation of their workloads. Dynamic configuration is an effective way to improve resource utilization while satifying service-level objectives (SLOs).

After system virtualization is introduced, applications are deployed in guest operating systems (OSes). Guest OSes usually use virtualized resource and virtual machine monitors (VMMs) are responsible for the mapping of virtualized resource to physical one. Thus, resource contentions would occur in two layers: one inside VMs and the other in the layer of VMM.

Resource contentions inside VMs mainly happen among multiple processes or threads of applications. When scarce resource is time-shared by these processes or threads, corresponding management overhead on resource allocation, scheduling and process context switching becomes high, which directly induces performance degradation of services. If much more resource is added, the contention can be removed. So, dynamic configuration for VMs with varying workloads is an effective method to mitigate resource contentions inside VMs.

Resource contentions in the layer of VMM primarily occur among multiple VMs residing on a same physical node. Contentions would become intense especially when total demands of all resident VMs are much larger than the amount of real physical resource that can be provided by the node. Decreasing workloads of physical nodes is the key way to mitigate resource contentions in the layer VMM. Virtual machines can be migrated lively from overloaded physical nodes to light-loaded ones.

The details about mitigating the two kinds of resource contentions are described in the following part.

A. Local Resource Configuration Strategy (LRC) for a Virtual Machine

VScheduler intends to adjust processor resource configuration for virtual machines when their loads vary with time.

We employ *Additive Increase/On-demand Related Decrease* (AIORD) strategy to configure processor resource for virtual machines. The AIORD rules may be expressed as follows:

$$I : l_{t+1} \longleftarrow l_t + \alpha; \alpha > 0$$
$$D : l_{t+1} \longleftarrow f(l_t, u_t)$$

where I refers to the increase of processor resource because of ascending processor demands, D refers to the decrease of processor resource based on current resource demands of applications, l_t the amount of processor resource configured for VMs at time t, u_t the amount of processor resource actually used by VMs at time t, and α is constant.

Here, α is set as 1. We define $f(l_t, u_t)$ as following:

$$f(l_t, u_t) = \frac{l_t + u_t}{2}$$

Linearly increasing processor resource is to prevent virtualized cluster from thrashing due to frequent virtual machine migrations induced by fast growth of the amount of processor resource, while decreasing processor resource based on current resource usage status is to avoid the waste of resource.

B. Global Resource Configuration Strategy (GRC) in a Cluster

When resource demands of a VM go up, the local resource configuration strategy would configure the VM with more or less processor resource. But the performance of the VM can not be assured to be improved. Local resource configuration just makes possible the VM using more processor resource. It is virtualized resource that VMs use and VMM is responsible for mapping virtualized resource to physical one. Whether the VM can really use more resource depends on the real usage status of physical resource on the node that the VM resides on.

If the total amount of virtualized resource employed by all resident VMs is much larger than the amount of physical resource provided by the node, resource contention must be existing. When a physical node has scarce resource for all resident VMs, we call the node an overloaded one. On the contrary, if a node has excessive resource for current workloads, the node is called a lightloaded one. A moderate-loaded node means that the node has the right amount of resource just matching the demands of workloads.

A feasible method to mitigate the contention on VMM is to decrease workloads of the node. Live migration of virtual machines is a good choice. Virtual machines can be migrated lively and smoothly from an overloaded node to a lightloaded one. Global resource configuration is to balance the workloads of nodes in a virtualized cluster.

Global resource configuration intends to redistribute VMs on physical nodes. In fact, the remapping of virtual machines to physical nodes is a NP-hard problem [18]. We only find a good solution close to the optimal one. Here, we

Table I: Symbols and Definitions

Symbol	Definition
N	the total number of physical nodes
M	the total number of virtual machines
$\mathcal{H}$[N][M]	the distribution of VMs on physical nodes
$\mathcal{C}_p$[N]	processing unit capacity associated with each node
$\mathcal{C}_m$[N]	memory capacity associated with each node
$\mathcal{R}_p$[M](t)	processing unit demand of each VM at time t
$\mathcal{R}_m$[M](t)	memory demand of each VM at time t
$\mathcal{T}_{vm}$[M](t)	workloads' variation tendency of each VM at time t
$\mathcal{T}_{pm}$[N](t)	workloads' variation tendency of each node at time t

employ a greedy heuristic to dynamically redistribute virtual machines on a cluster of physical nodes because of its fast performance.

First, some notations are defined in Table I.

$\mathcal{H}$ is defined as follows:

$$\mathcal{H}[i][j] = \begin{cases} 0 & \text{if node } n_i \text{ is NOT hosting the VM } v_j, \\ 1 & \text{if node } n_i \text{ is hosting the VM } v_j. \end{cases}$$

We also define the workloads' variation tendency of VM v_j and node n_i at time t as follows:

$$\mathcal{T}_{vm}[v_j](t) = \mathcal{R}_p[v_j](t) - \mathcal{R}_p[v_j](t-1)$$
$$\mathcal{T}_{pm}[n_i](t) = \sum_{v_j, \mathcal{H}[i][j]=1} \mathcal{T}_{vm}[v_j](t)$$

There are two main issues that should be solved when we design a global resource configuration strategy:

1) Which Virtual Machines Should be Migrated: an intention to migrate VMs is to mitigate resource contentions among VMs on physical nodes. Some VMs should be moved from an overloaded node to others, thus the remaining VMs have adequate resource to use. How to select VMs on overloaded nodes to be moved out? There are two factors that should be considered. One is effectiveness, which means that after the migration of these VMs, an overloaded node becomes a moderate-loaded one. The other is migration cost. The cost should be small.

Total migration time of VMs is usually an important metric for migration cost [3]. We define the migration cost function g_{mc} as follow. The estimated cost $g_{mc}(v_j)$ of migrated VM v_j is the amount of memory allocated to v_j.

$$g_{mc}(v_j) = \mathcal{R}_m(v_j)$$

Furthermore, for light-loaded nodes, server consolidation should be done for the sake of power-saving. All VMs on one node are migrated to others and the node becomes inactive. Power is saved because of less active physical nodes. In the case, how to select VMs to be migrated is also based on minimal migration cost.

2) Which Node Should a Virtual Machines be Migrated to: light-loaded nodes are the best candidates. When we choose a destination node for a migrated VM v_j to be moved to at time t_1, workloads' variation tendency is an important

Algorithm 1 GRC algorithm

```
 1:  for each over-loaded node n_i do
 2:      compute a VM set that just makes n_i un-overloaded
              and has the smallest estimated migration cost;
 3:  end for
 4:  for each VM v_j to be migrated do
 5:      workloadsTdSum ← MAXSUM;
 6:      for each light-loaded node n_i do
 7:          if (T_pm[n_i] + T_vm[v_j]) < workloadsTdSum then
 8:              workloadsTdSum = T_pm[n_i] + T_vm[v_j];
 8:              destNode = n_i;
 9:          end if
10:      end for
11:      add (v_j, destNode) into List migrationTaskList;
12:  end for
```


Figure 2: The system architecture

factor that we care about. GRC prefers the node n_i that the sum of $T_{pm}[n_i](t_1)$ and $T_{vm}[v_j](t_1)$ is the closest to 0. Thus, after VM v_j is migrated to node n_i, the probability that resource contention occurs on the node in the future would be small based on the analysis of history information. The reason is that, workloads' variation tendency of VM v_j is contrary to that of node n_i.

If there is no available light-loaded node to host the migrated VM, VM swap is our second choice to decrease workloads on overloaded nodes. By VM swap, more free resource on the destination node is available for hosting the migrated VM. A scratch node must be ready for VM swap to temporarily host VMs. However, if VM swap fails to mitigate resource contentions on overloaded nodes, an inactive physical node can be activated to host migrated VMs.

IV. SYSTEM DESIGN AND IMPLEMENTATION

VScheduler aims at processor resource configuration for services' QoS and high resource utilization in a large virtualized cluster.

A. System Architecture

VScheduler is designed to be a three-layer, centralized architecture, including a set of VMAgents inside VMs, a group of PMAgents residing on physical computing nodes and a control center – manager on a management node. The system architecture of VScheduler is shown in Fig. 2.

VScheduler manager consists of two main modules: a resource configuration scheduler and a command executor. The scheduler makes decisions to find out a good resource reconfiguration solution for virtualized nodes. The goal of dynamic configuration is to mitigate resource deficits when resource demands of services change. The scheduler includes two parts: local resource configuration (LRC) and global resource configuration (GRC). Command executor implements the new resource configuration generated by the scheduler. Configuration overhead should be as small as possible.

VScheduler is implemented based on Xen 3.1.0. By invoking interfaces provided by XenStore [19], PMAgent in
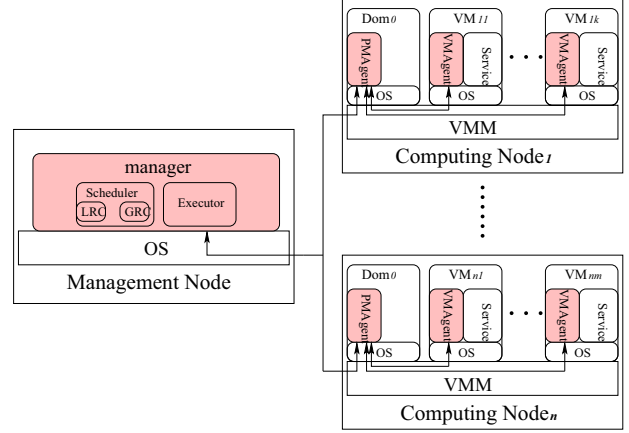
domain0 collects resource usage information about processor, memory and network for each resident virtual machine. VMAgent periodically gathers process-related or thread-related information from */proc* file system.

B. Parallel Migration Mode of Multiple Virtual Machines

As the overhead of dynamic configuration for an individual VM is neglectable, the resource configuration overhead of VScheduler is mainly incurred by global resource configuration – live migration of virtual machines. Total migration time of VMs is usually an important metric for migration cost [3]. To reduce the cost, we parallelize migration processes of some unrelated virtual machines to get shorter total migration time (TMT).

Table II shows total migration time of two virtual machines in different scenarios. The two VMs are configured with four VCPUs and 1GB RAM. They both run memory database applications.

Serial mode means that only after one VM finishes the complete migration process, the other starts to migrate.

Parallel mode denotes that migration processes of two VMs overlap each other, that is, one VM starts to migrate just before the other finishes the migration process. There are four situations in parallel mode.

DSDD: Two VMs migrate from *Different Source* nodes to *Different Destination* nodes (DSDD);

DSSD: Two VMs migrate from *Different Source* nodes to a *Same Destination* node (DSSD);

SSDD: Two VMs migrate from a *Same Source* node to *Different Destination* nodes (SSDD);

SSSD: Two VMs migrate from a *Same Source* node to a *Same Destination* node (SSSD).

In Table II, *TMT-VM1* means total migration time of VM1 and *TMT-VM2* total migration time of VM2. *TMT-VM1&VM2* denotes total migration time of VM1 and VM2. From Table II we can conclude that parallelizing the migration process of two VMs with different source nodes and

Table II: Total Migration Time of VMs in Diverse Modes

VM migration mode		TMT-VM1 (s)	TMT-VM2 (s)	TMT-VM1&VM2 (s)
serial-mode		15.878	16.908	32.786
parallel-mode	DSDD	15.636	16.425	16.425
	DSSD	28.967	26.913	28.967
	SSDD	27.163	29.365	29.365
	SSSD	31.932	32.208	32.208

different target nodes would greatly shorten total migration time of mulitple VMs.

V. PERFORMANCE EVALUATION

In this section, VScheduler has been tested on a wide variety of workloads. The experimental results demonstrate that LRC and GRC strategies in VScheduler both can effectively mitigate resource deficits in a virtualized cluster or data center. They complement each other and none of them can be dispensed with.

A. Experiment Setup

We conduct our experiments on a cluster consisting of 16 server-class nodes. Each node is configured with 2-way quad-core Xeon E5405 2GHz CPUs and 8GB DDR RAM. These nodes have Intel 80003ES2LAN gigabit network interface card (NIC) and are connected via switched gigabit Ethernet. Storage is accessed via iSCSI protocol from a NetApp F840 network attached storage server. We use Redhat Enterprise Linux 5.0 as the guest OS and the privileged domain OS (domain 0). The host kernel is the modified version of Xen 3.1.0.

In the following experiments, Apache servers, LAMP servers and *tbench* run in virtual machines. RUBiS [9], an open-source auction site benchmark, is run on LAMP server. *dbench* [20] is an open source benchmark producing the filesystem load. As one part of *dbench*, *tbench* produces the TCP and process load. Apache 2.2.3-11, PHP 5.1.6, MySQL 5.1.39, RUBiS 1.4.2 [21], and *dbenth* 4.0 are respectively used in the experiments. *httperf* 0.9.0 [22] is used to generate continuous access requests for servers.

B. Effectiveness of LRC Strategy

To evaluate the effectiveness of our LRC strategy, we conduct a set of experiments using RUBiS and *tbench*. Each benchmark respectively runs in three settings: baseline, best case, and LRC.

The baseline and the best case both use static resource configuration. The difference is that, the VM is configured with four virtual CPUs (VCPUs) in the baseline case and eight VCPUs (the number of real physical CPUs) in the best case. For LRC, dynamic resource configuration is used and the VM is configured with four VCPUs initially.

Each experiment is repeated five times and every test result comes from the arithmatic average of five values.

From Fig. 3 we observe that LRC has performance very close to the best case. Although LRC has a little performance
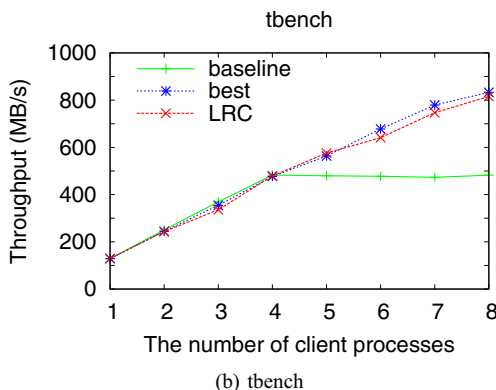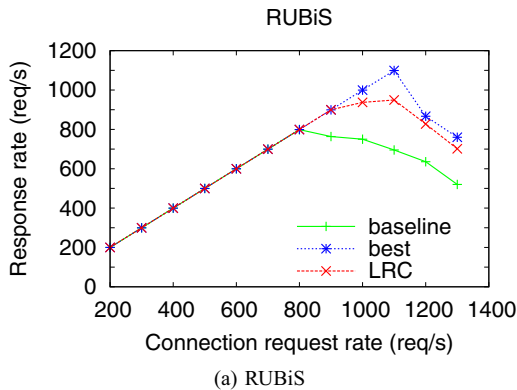

(a) RUBiS

(b) tbench

Figure 3: Performance comparison

loss as opposed to the best case, it provides a more flexible way to configure processor resource than traditional static configuration methods.

C. Virtual Machine Migration

This part of experiments demonstrate the effectiveness of virtual machine migration in the GRC strategy.

To evaluate the effects of VM migration, the experiment uses three physical nodes and eight VMs. The initial configurations of these VMs are listed in Table III. All VMs run Apache serving dynamic PHP web pages. The PHP scripts are designed to be CPU intensive. *httperf* is used to inject workloads of these VMs.

Fig. 4 shows a series of processor load on each VM along with the triggered migrations. Each block represents a VM. At about thirty seconds, a large load is placed on

Table III: The Initial Configuration of Eight VMs

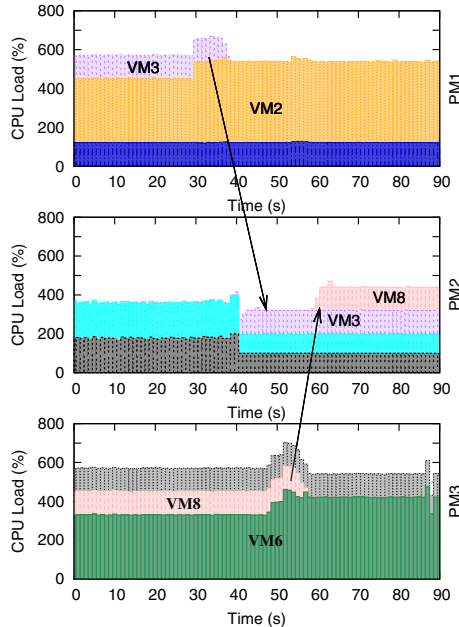| VM	Number of VCPUs	RAM (MB)	Start PM
1	2	512	1
2	4	512	1
3	2	512	1
4	3	512	2
5	3	512	2
6	4	512	3
7	2	512	3
8	2	512	3

Figure 4: A series of migrations to mitigate resource contention

VM2, causing the CPU utilization on PM1 to exceed the busy threshold. Our GRC strategy examines candidate for migration. VM3 has the lowest migration cost and is selected as a candidate. Because PM2 has sufficient free resource to host VM3, it is migrated there, thereby eliminating the hotspot. Similarly, at about 50 seconds, VM6 has increasing workloads to make processor resource on PM3 in busy state, causing VM8 to migrate from PM3 to PM2. Thus, processor resource on PM3 is not busy again.

D. LRC and GRC

In this part, we demonstrate that LRC and GRC strategies in VScheduler both can effectively mitigate resource deficits in a virtualized cluster or data center. They complement each other and none of them can be dispensed with.

The set of experiments use one physical node and three VMs. Each VM is configured with three VCPUs and 1GB RAM. All VMs run RUBiS benchmark. Same connection requests are sent to the three VMs simultaneously. We test the average response rates of VMs using *httperf* with
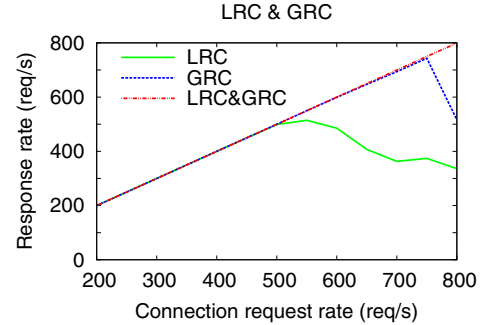

Figure 5: The effects of LRC and GRC

different connection request rates. The test is repeated in three settings: only LRC, only GRC, and LRC&GRC. LRC means that only dynamic resource configuration inside VMs can be done, while GRC denotes that only VM migration is used to mitigate resource deficits. LRC&GRC means that VMs can be dynamically configured by changing the amount of virtual resource and their resident nodes.

Fig. 5 shows the effects of three settings. When connection request rate reaches about 500 req/s, average response rates of VMs in setting LRC begin to drop down. Because the three VMs share eight physical processors, there must be resource contentions among resident VMs even if VMs are configured with more virtual resource. In setting GRC, since live VM migration mitigates resource deficits on the physical node, average response rates go up with the increase of connection request rates. But when connection request rate arrives at about 750 req/s, the performance of VMs starts to decrease. The reason is that resoure deficits occur inside VMs and VM migration could not do anything for this.

E. Data Center Prototype

We conduct an experiment to demonstrate how VScheduler performs under realistic data center conditions. We deploy a prototype data center on a cluster of ten servers that run a total of twenty-eight VMs. An additional node runs VScheduler manager as the management node and the other several nodes send requests to these servers using *httperf* as clients.

The virtual machines run a mix of data center applications including Apache servers, LAMP servers, *tbench*. We run RUBiS on LAMP servers. Of the twenty-eight deployed VMs, five run the RUBiS application, six run *tbench*, seven run Apache serving CPU-intensive PHP scripts, two run main memory database applications, and the remaining eight serve a mix of PHP scripts and large HTML files.

The virtual machine distribution on physical nodes is described in Fig. 6. Fig. 6(a) shows the 28 VMs are deployed on physical nodes in a random mode initially. According to changing application loads, the global scheduling algorithm
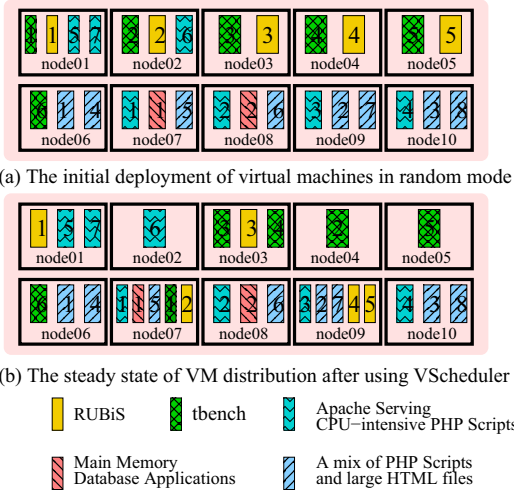
(a) The initial deployment of virtual machines in random mode

(b) The steady state of VM distribution after using VScheduler

Figure 6: The VM distribution on physical nodes in initial status and steady status

Figure 7: Total migration time of virtual machines varying with memory size

Figure 8: GRC performance and the number of VMs

redistributes these virtual machines. Some CPU-intensive applications monopolize a physical node due to their high processor loads, while other applications share a node. After using VScheduler, smaller numbers of virtual machine migration occurs. The whole data center tends to a stable status as shown in Fig. 6(b).

F. System Overhead and Scalability

Overhead in VScheduler is mainly used for resource configuration in a cluster. Its value is dependent on the number of physical nodes and virtual machines in the cluster.

Resource configuration overhead is primarily spent on migrating VMs. Total migration time of VMs is usually considered as a key metric, while the duration of VM migration is related to its memory size. Fig. 7 shows the relationship of migration time and VM memory size. To shorten the total migration time of all migrated VMs, we introduce the parallel mode of VM migration. Parallel mode effectively reduces the total migration time by overlapping migration processes of some migrated VMs.

The scalability of VScheduler is based on computational complexity of resource configuration scheme, especially global resource configuration algorithm. Although a remapping of VMs to physical nodes is a NP-hard problem, our heuristics can find a proper solution with acceptable time overhead. Fig. 8 describes the relationship of the scale of problem and algorithm performance. For very large data centers with thousands of VMs, the computation could be split up across multiple nodes, or applications can be broken up into farms, each controlled independently by its own local control center.
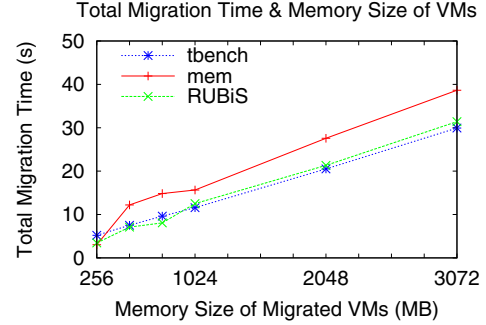
VI. CONCLUSION AND FUTURE WORK

In this paper, we present VScheduler, a system that dynamically adjusts resource configuration of virtual machines, including the amount of virtual resource and a new mapping of virtual machines and physical nodes. A two-level resource configuration scheme is proposed – local resource configuration (LRC) for an individual virtual machine and global resource configuration (GRC) for a whole cluster or data center. GRC especially takes variation tendency of workloads into account when remapping virtual machines to physical nodes. We implement our techniques in Xen and conduct a detailed evaluation using RUBiS and *dbench*. The experimental results show that VScheduler not only satisfies resource demands of services, but also reduces the number of virtual machines migration, which provides a stable VM distribution on physical nodes in data centers.

Currently, our system just dynamically configures processor resource in virtualized environments. We would like to extend our work to consider other computing resource, such as memory, disk I/O, and network resources.

VII. ACKNOWLEDGMENT

REFERENCES

[1] M. Welsh and D. Culler, "Adaptive overload control for busy internet servers," in *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems*, 2003.

[2] R. Nathuji and K. Schwan, "Virtual power: Coordinated power management in virtualized enterprise systems," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP'07)*, 2007, pp. 265–278.

[3] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proceedings of the ACM/Usenix International Conference on Virtual Execution Environments (VEE'09)*, 2009, pp. 41–50.

[4] C. Sun, L. He, Q. Wang, and R. Willenborg, "Simplifying service depolyment with virtual appliances," in *Proceedings of IEEE International Conference on Service Computing (SCC'08)*, 2008.

[5] G. Li and Y. Liang, "Constructing service oriented dynamic virtual enterprise in chinese apparel industry," in *Proceedings of IEEE International Conference on Service Computing (SCC'10)*, 2010.

[6] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI'07)*, 2007, pp. 229–242.

[7] C. Weng, M. Li, Z. Wang, and X. Lu, "Automatic performance tuning for the virtualized cluster system," in *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS'09)*, 2009, pp. 183–190.

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM symposium on Operating Systems Principles (SOSP'03)*, 2003, pp. 164–177.

[9] C. Amza, A. Chanda, A. L.Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite, "Specification and implementation of dynamic web site benchmarks," in *Proceedings of the 5th IEEE Annual Workshop on Workload Characterization*, 2002, pp. 3–13.

[10] Y. Zhang, A. Bestavros, M. Guirguis, I. Matta, and R. West, "Friendly virtual machines: Leveraging a feedback-control model for application adaptation," in *Proceedings of the 1st International Conference on Virtual Execution Environment (VEE'05)*, 2005, pp. 2–12.

[11] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three cpu schedulers in xen," *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, pp. 42–51, Sep. 2007.

[12] K. J. Duda and D. R. Cheriton, "Borrowed-virtual-time (bvt) scheduling: Supporting latency-sensitive threads in a general-purpose scheduler," in *Proceedings of the 17th ACM symposium on Operating Systems Principles (SOSP'99)*, 1999, pp. 261–276.

[13] I. Leslie, D. Mcauley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The design and implementation of an operating system to support distributed multimedia applications," *IEEE Journal of Selected Areas in Communications*, vol. 14, pp. 1280–1297, Sep. 1996.

[14] Credit scheduler. [Online]. Available: http://wiki.xensource.com/xenwiki/CreditScheduler

[15] W. Zhao and Z. Wang, "Dynamic memory balancing for virtual machines," in *Proceedings of the ACM/Usenix International Conference on Virtual Execution Environments (VEE'09)*, 2009, pp. 21–30.

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

[17] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proceedings of the 4th ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys'09)*, 2009.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1st ed. W. H. Freeman, 1979.

[19] D. Chisnall, *The Definitive Guide to the Xen Hypervisor*, 1st ed. Prentice Hall PTR, 2007.

[20] Open source benchmark producing the filesystem load. [Online]. Available: http://samba.org/ftp/tridge/dbench

[21] Rice university bidding system. [Online]. Available: http://rubis.ow2.org/

[22] httperf homepage. [Online]. Available: http://www.hpl.hp.com/research/linux/httperf/