

Toward scalable Web systems on multicore clusters: making use of virtual machines

Xuanhua Shi · Hai Jin · Hongbo Jiang ·
Xiaodong Pan · Dachuan Huang · Bo Yu

© Springer Science+Business Media, LLC 2011

Abstract Limited by the existing design pattern, a lot of existing softwares have not yet taken full use of multicore processing power, incurring low utilization of hardware, even a bottleneck of the whole system. To address this problem, in this paper, we propose a VM-based Web system on multicore clusters. The VM-based Web system is scheduled by Linux Virtual Server (LVS) and we implement the web server with Tomcat. In the mean time, we develop *VNIX*, a set of VM management toolkit, to facilitate managing VMs on clusters, aiming at improving the usage of multicore CPU power. To reduce resources contention among VMs, we propose to deploy LVS schedulers distributively on different physical nodes. To evaluate our approach, we conduct extensive experiments to compare VM-based Web system with classical physical machine-based Web system. Our experimental results demonstrate that the proposed VM-based Web system can result in throughput improvements of up to three times compared with the same multicore clusters, with an error rate at the server side as low as 20% of that of classic systems.

X. Shi (✉) · H. Jin · X. Pan · D. Huang · B. Yu
Service Computing Technology and Systems Lab, Cluster and Grid Computing Lab, School
of Computer, Huazhong University of Science and Technology, Wuhan, 430074, China
e-mail: xhshi@hust.edu.cn

H. Jin
e-mail: hjin@hust.edu.cn

X. Pan
e-mail: xdongp@gmail.com

D. Huang
e-mail: hdc1112@gmail.com

B. Yu
e-mail: yubo@hust.edu.cn

H. Jiang
EIE Department, Huazhong University of Science and Technology, Wuhan 430074, China
e-mail: hongbojiang2004@gmail.com

1 Introduction

Technology trends in recent years of multicore process have resulted in a single chip's increasing processing power and capacity, while not requiring a complex microarchitecture [3]. We found that most existing works take the advantage of the multithreading programming to utilize a multicore platform [1, 17]. One drawback using multithreading programming is that the programmers have to improve the software performance case by case, which is not feasible for large-scale data centers [19], or the cloud centers [32], exemplified by Amazon EC2.¹ An operating system for multicore gives another solution to improve software performance [5]; one drawback of this is that the coordination among a cluster will be a new challenge.

On the other hand, virtualization techniques are widely used in years, to present the illusion of many small virtual machines (VMs); each of which runs a separate operating system instance [4]. VM technologies have been capitalized on the ease of system management and administration. In addition, via a Virtual Machine Monitor (VMM or hypervisor), which is implemented directly on hardware, VM technologies allow isolation of hardware and software management. VM technologies have already been widely adopted in industry computing environments, especially data-centers [18].

Our work is motivated by the fact that the virtualization technology makes possible to deploy VMs on the multicore platform. We emphasize that this work is not straightforward and we are facing numerous challenges. For example, while high-performance web site design techniques have been extensively studied in the past 10 years [13], there are few efforts on the web site performance in the VM environment with the multicore platform, even less for the web site performance optimization in virtual machine environment with multicore clusters. In addition, in the VM environment, due to the interactions of the underlying VMM and other VMs, an application's performance could be greatly different from a nonvirtualized environment [20].

In this paper, we propose a VM-based Web system on multicore clusters. This VM-based Web system is scheduled by Linux Virtual Server (LVS).² We summarize the main contributions of this paper as follows:

- We propose a scalable VM-based Web system, including front-end load balancers and back-end servers. By exploiting VM technology, a multicore CPU is split into multiple small VCPUs such that LVS scheduler and the Tomcat server could take full use of multicore processing power [17].
- We develop *VNIX*, a set of VM management toolkit, to facilitate managing VMs on clusters, so as to improve the usage of multicore CPU power.
- Our extensive results demonstrate the proposed VM-based Web system can result in throughput improvements of up to three times compared with the same multicore clusters, as well as error rate deterioration of the client requests up to 20% compared with classical Web systems.
- One important observation of our experimental results is that VMs with applications running *at OS kernel level* (say, LVS) on one physical machine are more

¹Amazon EC2. <http://aws.amazon.com/ec2/>.

²LVS. <http://www.linuxvirtualsever.org/>.

prone to influencing each other, compared with VMs with applications (such as Tomcat) *at application levels*. This leads to an implication that it is undesirable to deploy VMs with OS kernel processes on the same physical node. This is the reason why in our algorithm the LVSs are distributedly deployed across physical machines.

The rest of this paper is organized as follows. Section 2 shows the architecture of web systems. We present the experimental setup in Sect. 3, and analyze the experimental data in Sect. 4. Section 5 introduces the related work. The conclusions and future work are presented in Sect. 6.

2 Architecture of Web systems

In this section, we present the architecture of our proposed VM-based Web system. We first give a general overview of the Web System, followed by a description of the VM-based web server and the VM-based load balancer. Finally, the algorithm of VM management is proposed.

Distributed architectures of Web systems could vary, depending on the name virtualization being extended at the IP layer or not. There exist three classes of architectures for a set of server nodes that host a Web site at a single location: *Cluster-based Web system* (or Web cluster) where the server nodes mask their IP addresses to clients. The only client-visible address is a Virtual IP (VIP) address corresponding to one device which is located in front of the Web server layer; *Virtual Web cluster* where the VIP address is the only IP address visible to the clients like in a cluster-based Web system. Unlike the previous architecture, this VIP address is not assigned to a single front-end device but shared by all the server nodes; *Distributed Web system* where the IP addresses of the Web-server nodes are visible to client applications [6]. A large-scale Web system consists of many server nodes, grouped in a local area with one or more mechanisms to spread client request among the nodes. A Web system requires one authoritative Domain Name System (DNS) server for translating the Web site name into one or more IP addresses. The client requests are spread by DNS to load balancers, and responded by the Web servers with different load balancing strategies. Our problem system in this paper falls into *cluster-based web system*. A typical architecture of cluster-based Web system is shown in Fig. 1.

2.1 VM-based Web server

In a classical Web system, a Web server is typically deployed on a physical machine. This Web server answers the requests, or direct these requests to the back-end data servers. In contrast, we propose an VM-based Web servers in this paper, which is shown in Fig. 2. All the Web servers are deployed on virtual machines, and the VMs are deployed on the physical nodes.

To manage the VMs over physical clusters, we developed VNIX, which provides a whole-set of tools for monitoring, deploying, controlling, and configuring virtual machines on physical clusters. In the VM-based Web server, we only employ three components of VNIX to handle the management of VMs: *Monitor*, *Deployer*, and

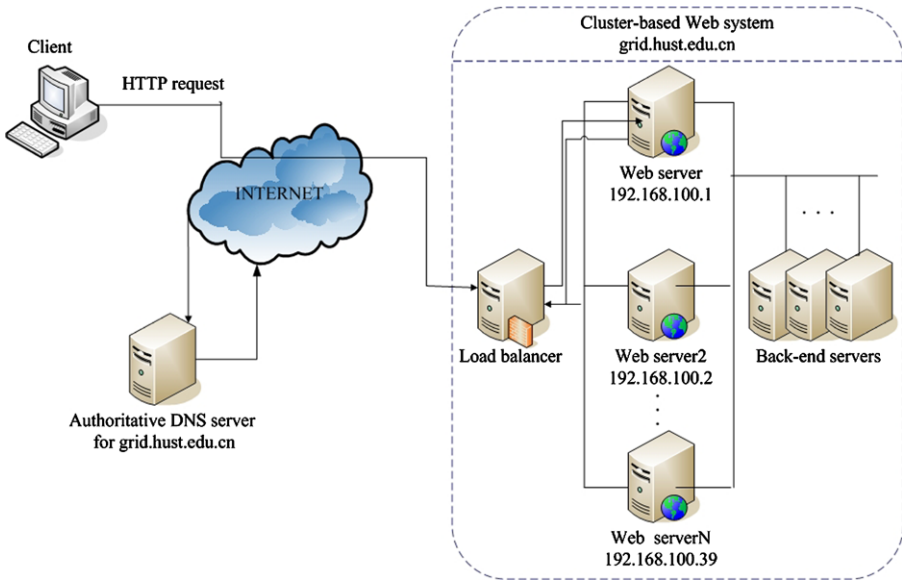
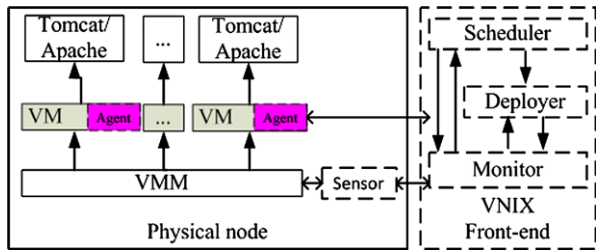


Fig. 1 Web cluster architecture

Fig. 2 VM-based Web server



Scheduler. The *Monitor* monitors the status of VMMs and VMs, such as CPU information, memory information, and I/O information. The *Deployer* handles the deployment/undeployment of VMs, and the *Deployer* supports incremental deployment of VMs, this makes it convenient to install new softwares on a existing VM. Moreover, the *Deployer* handles the network organization of the VMs, such as network IP assignment, gateway setup. The *Scheduler* is a key component in VNIX, which has two functions: to allocate specific resources to create a VM and to change the allocated resources to this VMs when necessary; migrate applications from one VM to another. Besides the VNIX modules, we extend OAR³ to support VM leasing function, a multi-functional cluster can be managed with VNIX and OAR. Due to space limitation, we do not present implementation details of VNIX in this paper, but the VNIX demo video can be downloaded from <http://grid.hust.edu.cn/xhshi/vnix.avi>.

The VM-based Web server consists of several components. Within one physical node, several VMs are deployed using Tomcat or Apache, and managed by the *VMM*.

³OAR. <http://oar.imag.fr/>.

In our proposed VM-base Web server, the operations of the *Deployer* and the *Monitor* of the VNIX are performed by a component, so-called *Sensor*. The *Sensor* component is deployed as shown in Fig. 2. It manages VMs through the *VMM*, performing actions such as deploying, suspending, destroying VMs. Besides, the *Sensor* component can monitor VM, gathering information such as CPU utilization, memory utilization, I/O utilization, etc. Outside the VMs, the *Sensor* cannot get the detailed information of the applications running on the VMs. To solve this problem, we deploy a *Agent* component inside the VMs. The *Agent* component is a daemon deployed in each VM, which monitors the VM status inside the VM. In the VM-based system, the *Agent* collects the workload of each Web server. We emphasize that when the workload of Web server is higher than a given threshold α , a new VM should be deployed to improve the performance of the Web systems in terms of throughput. Later we will discuss the setting of the appropriate threshold. Another function of the *Agent* is that it provides software configurations predefined by administrators. The *Sensor* and the *Agent* gather the information of each VM-base Web server, sent to the VNIX *Scheduler*. The operation of VMs directed by VNIX *Scheduler* is handled by the *Sensor* and the *Agent*.

With VNIX, the VM-based Web server can be managed according to Web system performance issue and power reduction issue. The idea about VM power reduction has been already discussed in [10]. In this paper, we only illustrate scalability issue of Web servers on VMs, to be exact, we strive to build a powerful Web systems with VMs on an existing multicore cluster.

2.2 VM-based load balancer

As illustrated above, load balancer is a key component for a Web cluster. There are lots of cluster-based load balancer, such as LVS, nginx.⁴ Nginx forwards traffic at the layer 7, so it is less scalable than LVS [18]. In this paper, we employ LVS as the load balancer for the Web system. LVS exploits several techniques to distribute IP packets among nodes in the cluster, such as network address translation, direct routing, and IP encapsulation. As mentioned in [25], IP encapsulation has better performance and higher flexibility. Our system chooses IP encapsulation, or IP tunneling, in which a packet intended for the virtual server is enclosed in another packet and re-targeted to one of the real servers. IP Tunneling allows the LVS administrator to put servers on separate network segments, enabling our systems to scale to more cluster nodes and clusters.

The Linux kernel is well known to make insufficient use of CPU power on multicore machines [5]. To address this, we propose two architectures of load balancers for the Web system, as shown in Fig. 3. The *Web* in Fig. 3 refers to a VM with Web server software, in this paper; we choose Tomcat as the Web server. The *LVS* in Fig. 3 refers to the VM with the LVS load scheduler. As shown in Fig. 3(a), each LVS load scheduler is deployed on a VM, and all the VMs are deployed on the same physical node. The management of the LVS VMs are handle by the VNIX modules as illustrated above. The client request is first directed by the DNS server to different LVS

⁴nginx. <http://nginx.net/>.

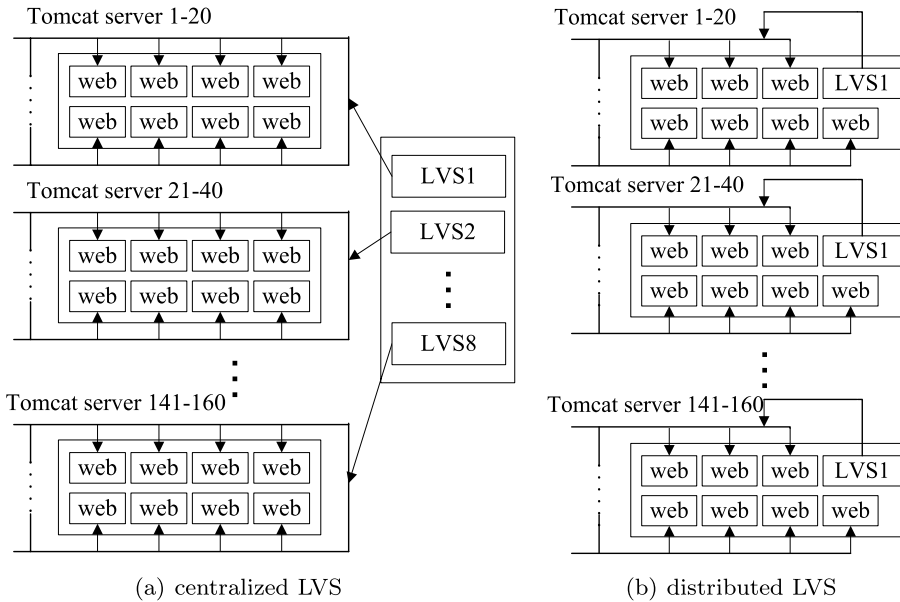


Fig. 3 The architectures of load balancer

schedulers. In Fig. 3(b), every LVS load scheduler is deployed on a VM. Contrast to Fig. 3(a), all the VMs are deployed distributively among the cluster nodes, instead of being deployed on a single physical node. By means of VM techniques, a multicore CPU can be split into multiple small VCPUs such that LVS scheduler could take full use of multicore processing power.

2.3 Algorithm of VM management

We tailor the general Linux OS to small appliances with the least necessary functions that the application requires so as to reduce the storage space and decrease the transferred data volume for VM deployment. We build two VM appliances: one is Linux kernel with LVS, the other is Linux kernel with Tomcat. The VM appliances scheduling is one of the key problem in the VM-based Web system.

The VM management is scheduled by two steps: the first is resource reservation from physical nodes, followed by appliance deployment, as shown in Algorithm 1. In the first step, a physical node is reserved from the physical cluster. The extended OAR system will decide which node will be selected; in the second step, VMs will be deployed on the selected node. As mentioned in Sect. 2.2, the VM-based Web system is composed with two components: a load balancer (LVS in this paper) and a back-end web server (Tomcat in this paper). What kind of VM will be deployed is decided by the scheduling algorithm. To avoid the VCPU overcommitted problem [31], we employ one CPU core to deploy one virtual machine, e.g., if one physical node has 8 CPU cores, one virtual machine with LVS is deployed on it, and there will be 7 cores left for deploying Tomcat servers. For simplicity, we split memory equally to each appliance.

```

Require: ( $NR_t, NC_t, NN_a$ )
1: Initialization;
2: // reserve one physical node, and return node $_i$ ;
3:  $i = \text{reserve\_node}(1)$ ;
4:  $N = 1$ ;
5: // deploy 1 LVS appliance and  $NC_t - 1$  Tomcat appliances;
6:  $\text{deploy}(\text{LVS}, i)$ ;
7:  $\text{deploy}(\text{Tomcat}, NC_t - 1, i)$ ;
8: // Configure Tomcat servers to LVS
9:  $\text{LVS} \leftarrow \text{Tomcat servers}$ ;
10: // Scheduling
11: while  $N \leq NN_a - 1$  do
12:   // If Tomcat overloaded
13:   if  $WL_T \geq TH_T$  then
14:      $i = \text{reserve\_node}(1)$ ;
15:      $N++$ ;
16:      $\text{deploy}(\text{Tomcat}, NC_t - 1, i)$ ;
17:   end if
18:   // If LVS overloaded;
19:   if  $WL_L \geq TH_L$  then
20:      $\text{deploy}(\text{LVS}, i)$ ;
21:      $\text{LVS} \leftarrow \text{Tomcat servers}$ ;
22:   else
23:      $\text{deploy}(\text{Tomcat}, 1, i)$ ;
24:      $\text{LVS} \leftarrow \text{Tomcat servers}$ ;
25:   end if
26: end while

```

Algorithm 1: VM management for web cluster

To describe the algorithm, we first present some notations. NR_t refers to the number of the total Web requests, NC_t refers to the number of total cores on one physical node. The number of the total available physical nodes are denoted by NN_a . WL_L refers to workload of LVS, and TH_L means the threshold of LVS. WL_T refers to workload of Tomcat, and TH_T means the threshold of Tomcat.

At the beginning of building the web system, one physical node is reserved from the multicore cluster, the extended OAR system returns the physical node number (Lines 2–4 in Algorithm 1). A LVS appliance and $NC_t - 1$ Tomcat appliances are deployed on the selected node (Lines 5–6), and all the Tomcat server are configured to be scheduled by the LVS appliance (Line 9);

With the number of the requests increasing, the Tomcat servers will be overloaded (Line 13). One more physical node will be reserved by OAR (Line 14), and $NC_t - 1$ Tomcat appliances will be deployed on the new reserved physical node (Line 16).

Then the workload of the LVS appliances will be checked, if the LVS appliances are overloaded, one LVS appliance will be deployed on the new reserved node (Lines 19–20), and the new deployed Tomcat appliances are configured to be scheduled by the new LVS (Line 21). If the LVS appliances are not overloaded, one Tomcat appliance will be deployed on the reserved node, and all the new deployed Tomcat servers will be configured to the existing LVS (Lines 23–24).

As a result, the web cluster scales well with the increasing of client requests until no further physical resource is available.

3 Experimental setup

In this section, we describe the hardware and software that we use for our experiments, and our experimental methodology.

Hardware and system software We conduct all the experiments on the IBM Blade Cluster in High Performance Computing Center (HPCC),⁵ Service Computing Technologies and Systems Lab/Cluster and Grid Computing Lab (SCTS/CGCL), Huazhong University of Science and Technology (HUST). SCTS/CGCL HPCC is Huazhong Grid Center of ChinaGrid [15], and CNGrid [33]. The IBM Blade Cluster is equipped with three X3650 nodes (one works as the management node and the other two work as I/O nodes), 39 HS21 nodes, and 20 TB SAN based on DS4800. Each HS21 node is equipped with 2-way, 4-core Intel Xeon CPU E5345 2.33 GHz and 8 GB memory. Besides, each HS21 node features 73 SCSI hard disk. All the HS21 nodes are connected with a Gigabit Ethernet network and a 10 Gbps Infiniband, and all HS21 nodes run RedHat Enterprise Linux 5 (Linux 2.6.18-8.el5).

In our experiments, the load balancer for web servers is LVS 1.24, and Web server is running Tomcat 5.5.17-8 with JDK 1.5.0_09. The VNIX, the Sensor and the Agent in Fig. 3 are implemented with Python 2.4.3. We take two paravirtualization implementations to server as VMs: one is Xen 3.1.0, and the other is VM-2. We cannot name the other virtual machine implementation directly because of license reasons, and thus we only call it VM-2 in this work. The operating system on all VMs is Linux 2.6.18-8.el5.

We use httperf 0.9.0 and autobench 2.1.1 to test the Web server performance. Httperf is run on each client machine such that the desired connection rate is distributed evenly among all the clients. We tune Tomcat to allow 65,535 connections per time. Aim at reducing I/O consumption, we only take the default Tomcat static testing file—index.html with the size of 8 KB, for the request target.

Methodology First, in order to maximize the number of concurrent connections a given client can create, it is necessary to increase the system wide limit of file descriptors on physical nodes and VMs.

```
01: * soft nofile 1000000
02: * hard nofile 1000000
03: sysctl -w 'fs.file-max=1500000'
```

Second, the amount of socket memory on each system was increased to allow a large TCP window size (the configurations on VMs are marked with (VM)).

```
01: echo 524288000 > /proc/sys/net/core/rmem_max
02: echo 524288000 > /proc/sys/net/core/rmem_default
03: echo 524288000 > /proc/sys/net/core/wmem_default
04: echo 524288000 > /proc/sys/net/core/wmem_max
05: (VM) echo 52428800 > /proc/sys/net/core/rmem_max
06: (VM) echo 52428800 > /proc/sys/net/core/rmem_default
07: (VM) echo 52428800 > /proc/sys/net/core/wmem_default
08: (VM) echo 52428800 > /proc/sys/net/core/wmem_max
```

⁵<http://grid.hust.edu.cn/hpcc>.

In addition, for all nodes, we need to configure the system such that TCP connection is able to be reconnected and recycled quickly (the configurations on VMs are marked with (VM)).

```
01:echo `1024 65535` > /proc/sys/net/ipv4/ip_local_port_range
02:echo 2147483648 > /proc/sys/kernel/shmmax
03:(VM) echo 214748364 > /proc/sys/kernel/shmmax
04:echo 5 > /proc/sys/net/ipv4/tcp_fin_timeout
05:echo 5 > /proc/sys/net/ipv4/tcp_keepalive_time
06:echo 1 > /proc/sys/net/ipv4/tcp_syncookies
07:echo 1 > /proc/sys/net/ipv4/tcp_tw_reuse
08:echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle
```

We first present some notations. *CPU workload* is the total workload of all CPU cores, and the full use of one core is 100%. We refer to *Reply rate* as the number of successful connections between the client and Web server in 1 second. *Response time* refers to the time between sending requests and getting response from the Web server. *Network throughput* refers to the speed of data transfer between clients and Web servers. *Error rate* refers to the quotient of error connections and total connections. The error connection is defined by `httperf` as follows: Specify the amount of time X that `httperf` is willing to wait for a server reaction. The timeout is specified in seconds and can be a fractional number (e.g., `-timeout 3.5`). This timeout value is used when establishing a TCP connection, when sending a request, when waiting for a reply, and when receiving a reply. If during any of those activities a request fails to make forward progress within the allotted time, `httperf` considers the associated connection as an error connection and increases the error count.

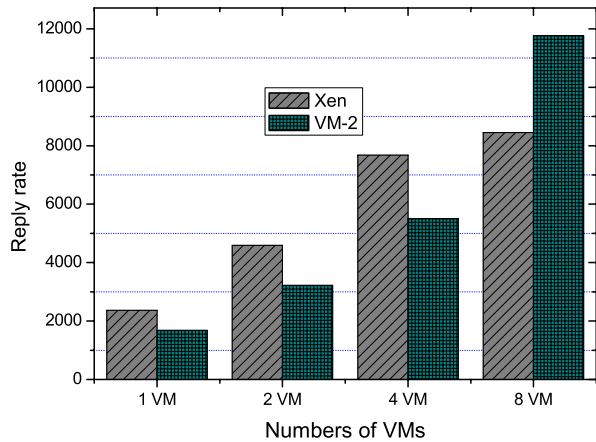
4 Results and analysis

Based on the setup in Sect. 3, we conduct several experiments on IBM Blade Cluster HS21 nodes. All experiments are run in dedicated mode, and each measurement is repeated 5 times, and we present the mean value.

4.1 Web servers on one physical node

First experiment The first experiment compares the Web servers performance of the two types of virtual machine implementation (Xen and *VM-2*). We deploy 1, 2, 4, 8 VMs on one HS21 node. For the case of 1 VM, we set 2 cores of CPU as VCPU and 3.6 GB memory as virtual memory; For 2 VMs, we set 2 cores of CPU as VCPU and 2 GB memory as virtual memory for each VM; For 4 VMs, we set 2 cores of CPU as VCPU and 1 GB memory as virtual memory for each VM; For 8 VMs, we set 1 core of an CPU as VCPU and 512 MB memory as virtual memory for each VM. We do not assign all the memory to virtual machines, because VMM also needs memory for VM management (for simplicity, we do not present experiment data about the memory allocation for VMM in this paper). The largest reply rates of Xen and *VM-2* are shown in Fig. 4. The column labeled “Xen” represents the reply rate of a given number of VMs on one HS21 node. The X axis shows the number of VMs on one HS21 node.

Fig. 4 Reply rate comparison between two kinds of VMs



We have two observations from Fig. 4: (1) The largest reply rate for Xen-based implementation is about 75% of the implementation on VM-2. The reply rate for the implementation on VM-2 grows linearly with the increasing numbers of VMs. (2) The reply rate for Xen-based implementation is larger than VM-2-based implementation when the number of VMs is small. However, the Xen-based implementation cannot improve the reply rate linearly with the increasing numbers of VMs.

We analyze the trace of Xen-base implementation with Oprofile,⁶ and the result shows that the “_do_softirq,” “_spin_lock_irq,” and “evtchn_set_pending” is increasing when the numbers of VM increases, which means more software interrupt, spin lock waiting, and event channel switch occurs. How to reduce software interrupt, spin lock waiting, and event channel switch is out of the scope of this paper. We try to make the maximum utilization of multicore resources, so we choose VM-2 in our implementation. As such, the rest experiments in this paper are conducted with VM-2.

Second experiment We conduct the second experiment to study the system performance in term of reply rate, response time, network throughput, and error rate with the respect to varying number of VMs at one physical node, and we evaluate five cases (deploying 1, 2, 4, 8, and 16 VMs on one HS21 node). The second experiment repeats exactly the same description of the implementation of 1, 2, 4, and 8 VMs as the first experiment. For 16 VMs, we assign 1 core as VCPU for 2 VMs and we assign 256 MB memory as virtual memory for each VM.

The reply rate, response time, network throughput and error rate of the Web servers are shown in Fig. 5. The X axis refers to the request rate of httperf (or the connections per second), the curve marked with “avg_rep_rate_ θ vm” represents the reply rate for θ VMs, the curve marked with “resp_time_ θ vm” represents the response time for θ VMs, the curve marked with “net_io_ θ vm” represents the network throughput for θ VMs, and the curve marked with “errors_ θ vm” represents the error rate for θ VMs.

The maximum CPU workload and memory utilization for different numbers of VMs are shown in Table 1. The HS21 node is equipped with 2-way four-core CPU,

⁶Oprofile, <http://oprofile.sourceforge.net>.

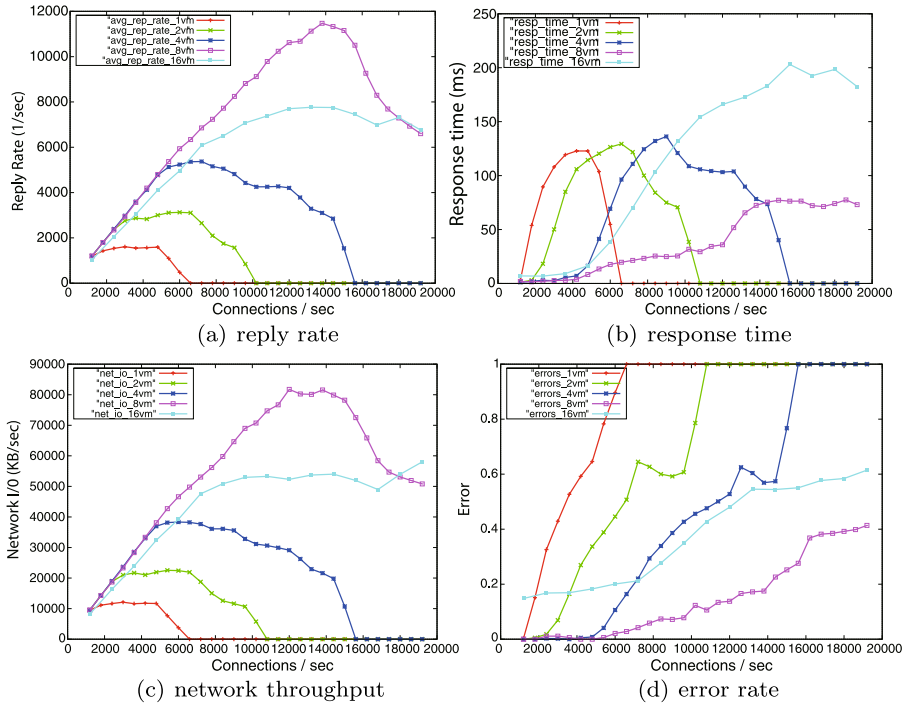


Fig. 5 Tomcat performance for different numbers of VM on HS21 node

Table 1 Workload of VMs

Workload	VM Num				
	1	2	4	8	16
CPU workload (%)	200	350	700	800	700
Memory (MB)	500	800	1600	3200	3700

totally there are 8 cores on one node. The CPU load in Table 1 sum up of the eight cores, e.g., if the load for every core is 50%, the load for the node will be 400% in Table 1; The memory utilization in Table 1 is the sum up of memory utilization of all VMs, e.g., for 2 VM, the memory utilization is about 800 MB; this means that each VM uses about 400 MB memory.

Figure 5(a) shows the reply rates for different numbers of VMs that we deploy on one HS21 node. From Fig. 5(a), we can see that reply rates on 16 VMs and 8 VMs have the largest reply rate when the number of the http requests is high, and when the http requests exceed a certain number, the reply rate on 1 VM, 2 VMs, and 4 VMs decreases sharply. The reasons are, based on the observation of Table 1: (1) For one VM on HS21, the CPU workload is only 200%, most of the CPU power are wasted, and the same reason goes to 2 VMs on HS21. For 4 VMs on HS21, the CPU workload is about 700%; this results in a higher reply rate. (2) Although CPU workload for 4 VMs and for 16 VMs are almost the same, the performance in terms of

reply rate will vary with the different the memory utilizations. (3) The CPU workload on 8 VMs is about 800%, which means that all HS21 CPU power are fully utilized. Memory utilization for 8 VMs is lower than for 16 VMs, but the reply rate on 8 VMs is higher than on 16 VMs; this shows that CPU utilization influences the reply rate more than memory utilization for our test setup. Because we assign one CPU core to two VCPUs, the VCPU scheduling should handle the contention of CPU resource between two VMs; this will incur extra overhead for VMM [31].

Figure 5(b) shows the response time for different numbers of VMs on one HS21 node. From Fig. 5(b), we find that: (1) Generally, the response time increases slowly for Web servers on more VMs, however, the response time on 16 VMs is larger than on 8 VMs; (2) When the response time reaches a peak, the response time decreases sharply for small numbers of VMs on one HS21 node, this shows that the Web servers are almost down at that time. For Web servers on 8 VMs, we did not see the inflection point even the number of request exceeds 18,000. We can conclude that 8 VMs on one HS21 has the best response time for http requests.

Figure 5(c) shows the network throughput of the VMs. Figure 5(c) shows the similar information as Fig. 5(a) and (b). Figure 5(c) also shows that the network throughput for 16 VMs is larger than for 8 VMs when http requests exceed 18,000. Because network throughput consists with successful connections and error connections, when http requests exceed 18,000, the number of error replies for 16 VMs is larger than for 8 VMs.

Figure 5(d) shows the error rate of different numbers of VMs. We can see that the error rate increases sharply for Web servers on 1 VM and 2 VM when the http request increases, as the curve with “errors_1vm” shows. After the sharp increasing, the error rate goes stable at about 100%; this is shows that all the Web servers are crashed after the http requests exceed the specific threshold. From Fig. 5(d), we can see that the error rate for 16 VMs is larger than for 8 VMs, even the number of http requests is smaller than 2,000. If there are a large number of VMs on one physical machine, the management of VMs is a challenge job, which leads to great pressure on the VMM. As analyzed above, Tomcat works better on 8 VMs deployed on one HS21 node.

Overall, we conclude that Tomcat running in 8 VMs works much better than a single Tomcat running on the physical node directly, the CPU load in Table 1 shows that Tomcat running on 8 VMs can improve the multicore CPU utilization greatly.

Third experiment Since the second experiment shows the best number of VMs, we choose to deploy on one HS21 node is 8, we conduct the third experiment to compare the Tomcat performance on a physical machine with that on 8 VMs. The experimental results are shown in Fig. 6. Figure 6 gives the comparison in terms of reply rate, response time, network throughput, and error rate. The curve marked with “xxx_phy” represents the “xxx” metric on physical node, and the curve marked with “xxx_8vm” represents the “xxx” metric on 8 VMs.

From Fig. 6(a), we can see that Tomcat on 8 VMs and on physical machine almost have the same reply rate when the requests are less than 5,000. When the number of requests is larger than 5,000, the reply rate on physical node does not increase any more, while the reply rate on 8 VMs can still increases linearly. When requests are larger than 12,000, Tomcat on physical node cannot reply any requests any more.

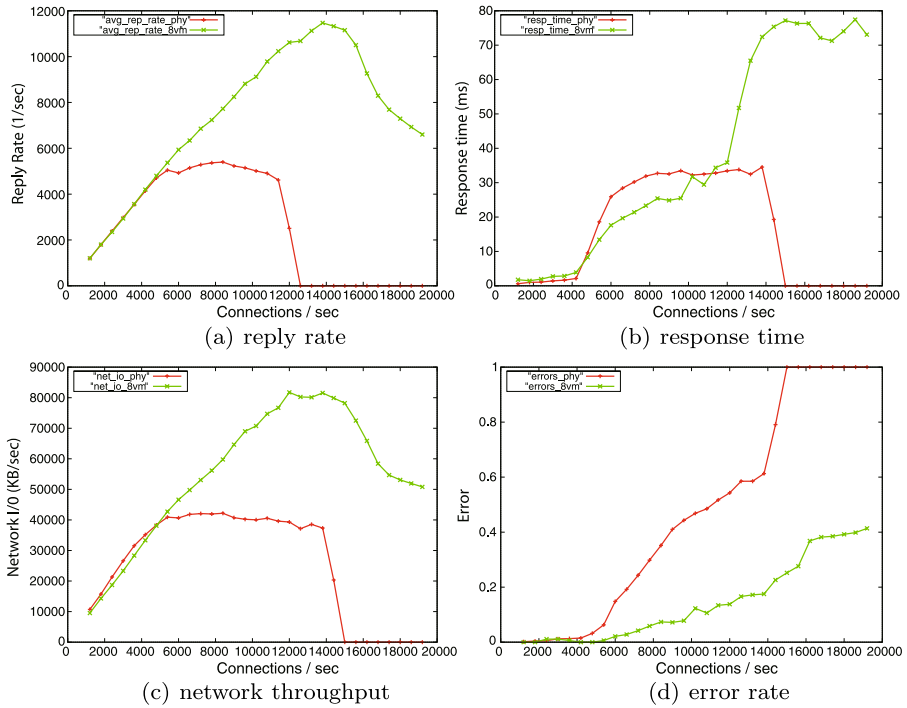


Fig. 6 Tomcat performance for VMs and physical node

This shows that VM-base Tomcat has much better reply rate than Tomcat on physical node directly. The maximum reply rate on the VM-base Web server is about 2.5 times as the maximum reply rate on the physical node.

Figure 6(b) shows that the response time on VMs are larger than on physical node when requests number is lower than 5,000. Generally Tomcat on VM has more software stack to respond to the client requests, if the requests number is not large, Tomcat on physical node can respond more quickly. After, the response time on physical node is smaller than on VMs, because reply rate on VMs are larger; this leads to a small response time. When requests are larger than 12,000, response time on VMs are larger than that on physical node, because the reply rate on physical node is very small, Tomcat can response these requests quickly.

Figure 6(c) shows the network throughput of two implementations, and we find that the maximum network throughput for VM-based Web server is 2 times for Web server on a physical node. Compared with the 2.5 times of reply rate, the network throughput does not increase such a scale, because the network throughput includes successful responses and errors.

From Fig. 6(d), we can see that the error rate on physical node increases sharply with the increasing of requests. When the requests are over 14,000, the error rate on physical node is about ten times on VMs. This shows that deploying proper number of VMs on multicore node cannot only improve the systems performance but also improve the reliability of the system.

4.2 Web servers with load balancers

Next, we strive to evaluate the performance of the whole Web system. The Web system is built with Tomcat and LVS, as shown in Fig. 3. We employ 37 HS21 nodes to run the experiment. Generally, 21 HS21 nodes are working as the web system, and 16 nodes work as http clients. We test three cases as follows:

- Case one: We use 20 HS21 nodes to work as Web servers, and deploy one Tomcat on each physical node. We use 1 HS21 node to deploy LVS scheduler on it. This is a classical Web system which is widely used today. We call case one *phy*, and the data we get are marked with “phy.”
- Case two: We use 20 HS21 nodes to work as Web servers, and deploy 8 VMs with Tomcat on each node. We use 1 HS21 to deploy 8 VMs with LVS scheduler. The architecture is shown in Fig. 3(a). We call case two *same*, and the data we get are marked with “same.”
- Case three: We use 21 HS21 nodes to deploy 168 VMs, each HS21 node is deployed with 8 VMs. To compare with the case two, we deploy the same number of VMs with LVS and some number of VMs with Tomcat. So, we deploy 8 LVS schedulers on 8 VMs, and deploy 160 Tomcat on 160 VMs, the distribution of LVS scheduler and Tomcat are shown in Fig. 3(b). The LVSs of the system distribute the load among 20 VMs with the Web server. We call case three *dist*, and the data we get are marked with “dist.”

With the setup mentioned above, we conduct the experiments, and we get the performance data of the whole Web system, which are shown in Fig. 7. The maximum utilization of CPUs are shown in Table 2. The workload in Table 2 sums up the workload of all cores on one node, and the full utilization of each core is 100%. From Table 2, we can see that the CPU utilization for “phy” is very low, it is only around 1/8 of the total system.

As shown in Fig. 7(a), the reply rate on “*phy*” Web system increases slower than on “*same*” and “*dist*.” This shows that the classical Web system directly deployed on physical node cannot reply all the client requests as quickly as on VM-based Web systems. The maximum reply rate for VM-based Web system is about twice as the maximum reply rate for classical Web system. Figure 7(a) also shows that the VM-based Web system works much better than the classical Web system when client requests exceed 20,000. From Fig. 7(a), we can see that “*dist*” system has higher reply rate than “*same*” system. This can be explained as this: LVS is part of Linux kernel, a lot of kernel operations in VMs are trapped into VMM; this leads to resource contention in the VMM. The VMs with LVS on one physical node influence each other, if lots of operations are trapped into VMM, the influence will be much

Table 2 CPU Workload of Web systems

	LVS server	Tomcat server
<i>phy</i>	100%	110%
<i>same</i>	60% * 8	50% * 8
<i>dist</i>	100% * 8	65% * 8

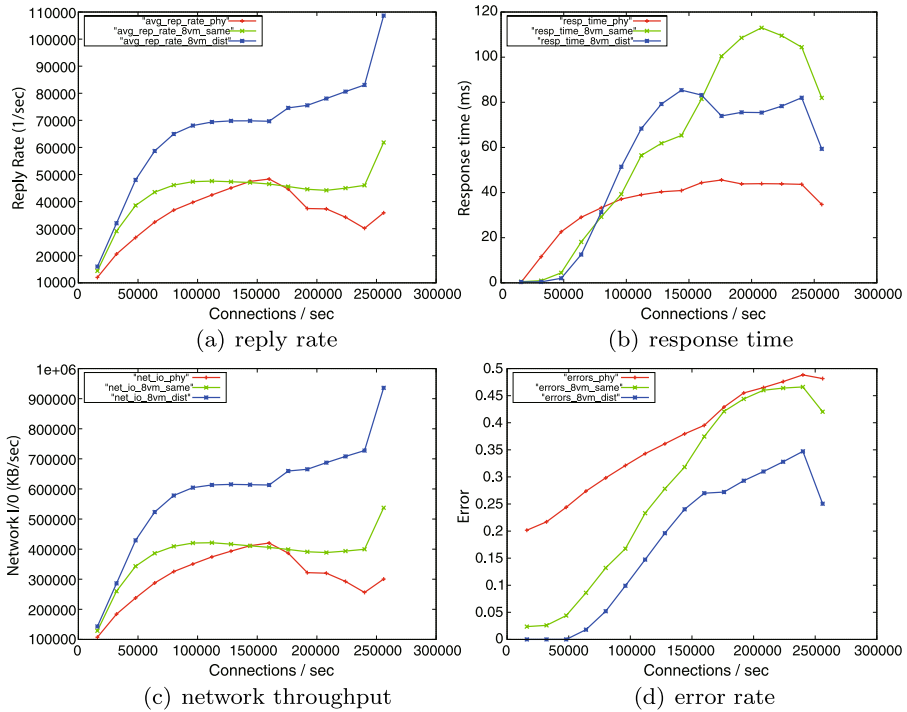


Fig. 7 LVS performance for VMs and one physical node

higher. The CPU workload in Table 2 also gives the explanation of this. In the “same” system, the cpu workload of LVS is only 480%, that means about 2/5 CPU power are not used by LVS. The cpu workload of Tomcat server are not fully used by Tomcat on “same” Web system, either. From the data in Table 2, it is better to create a new VM, when the CPU workload is over 65%. Thus, we set the coefficient α in Sect. 2.1 to 65% in our implementations.

Figure 7(b) shows response time of the Web systems. We can see that the response time on the classical Web system increases faster than the response time on the VM-based Web system with when http requests increase. However, when the http requests exceed 80,000, the response time for VM-based Web system is larger than for the classical Web system. This can be explained as this: When the total requests are less than 8,000, the classical Web system can handle all the Web requests, and the software stacks of the classical Web systems is simpler than VM-based Web system. When the requests exceed 80,000, the classical Web system is overloaded, the response time increases.

Figure 7(c) shows that network throughput on “dist” VM-based Web system can be three times as on classical Web systems. This shows that properly deployed VM-based system on physical nodes can have more http connections between the clients and the Web servers than the classical Web system deploying the Web software on physical node directly.

Figure 7(d) shows the error rate of these three cases. From Fig. 7(d), we can see that the error rate on “*dist*” VM-based Web system is much lower than other implementations. This shows that it is better to deploy VMs with different applications on physical node to improve the system reliability.

To summarize, on a give multicore cluster, VM-base web system has better performance than classical Web systems, especially distributively deploying VMs with load balancers on different physical nodes.

5 Related work

Multicore present many chances to computer science and applications, while multicore brings out many challenges. Some researchers discussed the impact of Multicore on computational science software [9], and the impact of performance asymmetry in multicore architectures [3]. Compared with the related research, our work targets the software performance in services computing areas, and these softwares are not computation-intensive. We employ Tomcat as an example to show how to improve software performance on multicore system without modifying the software itself.

Virtualization presents a new way to use multicore computers [4, 21, 23], e.g., combining batch execution and leasing for clusters using virtual machines [30], improving the agility in utility computing [27]. Some researchers have studied the management of virtual clusters, such as Sandpiper provide black-box and gray-box strategies for virtual machine migration [32], Padala et al. provides adaptive control of virtualized resources in utility computing environments [26], Nishimura et al. [24] system for rapid deployment of virtual clusters can deploy a 190-node cluster in 40 seconds, the Shirako system [11] uses VMs to partition a physical cluster into several virtual clusters. Amazon’s EC2 introduces the notion of cloud computing, where virtual machines are dynamically provisioned immediately with customized software environments and use is charged by the hour. More recently, Sotomayor et al. [29] introduce infrastructure-as-a-service (IaaS) management of private and hybrid clouds, and discussed the integration of Haizea, a resource lease manager, and OpenNebula, an open-source virtual infrastructure manager. The related research tries to provide a multifunctional data center with VMs, while our work maximums the CPU utilization and improves the web server throughput on multicore computers with VMs.

Many related works target the performance and high availability of Web servers, especially Web sites receive large numbers of Web requests concurrently. There are several techniques—including redundant hardware, load balancing, Web server acceleration, and efficient management of dynamic data—that can be used at popular sites to improve performance and availability [12]. Iyengar et al. [12] propose a dynamic data caching method to improve performance of Web systems, and many other Web server improvement techniques are studied in the past. Cardellini et al. [7] reviews the possible solutions for load balancing of Web systems, and there are lots of related research target load balancing issues on parallel and distributed systems, e.g., Zhang et al. [34] propose a workload-aware solution for Web system load balancing, Karger and Ruhl [16] propose two load-balancing protocols on peer-to-peer systems. Several groups target the performance of Web systems or cloud platforms. Mission

Critical Linux, Inc. gives a clear performance evaluation of LVS [25]. The virtual machines overhead in different areas are studied in [2, 8, 20]. Most Web performance are tested with `httperf` [22]. These researches target how to use VM to create data center to provide so-called cloud services or to evaluation performance of these cloud platforms; our work handles how to improve system performance and reliability of existing softwares on multicore platforms. These researches target the performance issue with optimization of Web server software or load balancers, while our work target the deployment of Web server software and load balancers on multicore clusters, and these related researches can be used in our systems.

6 Conclusions and future work

In this paper, we have proposed the design of a VM-based Web system on multicore clusters. The VM-based Web system is scheduled by LVS, and the Web server implementation uses Tomcat. We have developed a set of virtual machine management toolkit, called VNIX that facilitates the administration of virtual machines on clusters. VNIX can be used to deploy VMs with LVS and Tomcat, and to improve the resource utilization of multicore cluster. We have conducted extensive experiments to compare the performance of a VM-based Web system and a classical Web system on physical machines. The results show that VM-based Web systems works about three times faster than classical Web systems on the same multicore clusters, and the error rate of the client requests to the VM-based Web systems is about 20% of that of classical Web systems. In addition, through the experiments we have also observed that VMs with the OS kernel processes on one physical machine influence each other more than VMs with applications at application levels.

In the future, we will do several things: (1) To improve the load balancer's performance with VNIX for SIP servers. The motivation is that the SIP server cannot fully utilize the multicore resources [14]; (2) To improve VNIX scheduling algorithm with the Agent. Most scheduling algorithms in VNIX are based on the information obtained from VMM, that is, outside of the VMs. However, the information is not precise enough, especially when it comes to memory information; (3) To devise a good appliance scheduling method to reduce the influence among the appliances on individual physical node with adaptive components [28].

Acknowledgements This work was supported in part through National 973 Basic Research Program of China (No. 2007CB310900), National Natural Science Foundation of China (No. 60973037), and Wuhan Chenguang Program under Grant No. 201050231075.

References

1. Andrews DL, Niehaus D, Jidin R, Finley M, Peck W, Frisbie M, Ortiz JL, Komp E, Ashenden PJ (2004) Programming models for hybrid fpga-cpu computational components: a missing link. *IEEE MICRO* 24(4):42–53
2. Apparao P, Makineni S, Newell D (2006) Characterization of network processing overheads in xen. In: *Proceedings of the second international workshop on virtualization technology in distributed computing*. IEEE, New York

3. Balakrishnan S, Rajwar R, Upton M, Lai K (2005) The impact of performance asymmetry in emerging multicore architectures. In: Proceedings of the 32nd international symposium on computer architecture. IEEE Computer Society, New York
4. Barham P, Dragovic B, Fraser K, Hand S, Harris T (2003) Xen and the art of virtualization. In: Proceedings of the 19th ACM symposium on operating systems principles. ACM, New York, pp 164–177
5. Boyd-Wickizer S, Chen H, Chen R, Mao Y, Kaashoek F, Morris R, Pesterev A, Stein L, Wu M, Dai Y, Zhang Y, Zhang Z (2008) Corey: an operating system for many cores. In: Proceedings of the 8th USENIX symposium on operating systems design and implementation. USENIX Association
6. Cardellini V, Casalicchio E, Colajanni M, Yu PS (2002) The state of the art in locally distributed web-server systems. *ACM Comput Surv* 34(2):263–311
7. Cardellini V, Colajanni M, Yu PS (1999) Dynamic load balancing on web-server systems. *IEEE Internet Comput* 3(3):28–39
8. Cherkasova L, Gardner R (2005) Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In: Proceedings of 2005 USENIX annual technical conference, pp 387–390. USENIX
9. Dongarra J, Gannon D, Fox G, Kennedy K (2007) The impact of multicore on computational science software. *CTWatch Q* 3:3–10
10. Hu L, Jin H, Liao X, Xiong X, Liu H (2008) Magnet: a novel scheduling policy for power reduction in cluster with virtual machines. In: Proceeding of 2008 IEEE international conference on cluster computing cluster, 2008. IEEE Computer Society, New York, pp 13–22
11. Irwin D, Chase J, Grit L, Yumerefendi A, Becker D, Yocum KG (2006) Sharing networked resources with brokered leases. In: Proceedings of USENIX technical conference. USENIX
12. Iyengar A, Challenger J (1997) Improving web server performance by caching dynamic data. In: Proceedings of 1997 USENIX annual technical conference. USENIX
13. Iyengar A, Challenger J, Dias DM, Dantzig P (2000) High-performance web site design techniques. *IEEE Internet Comput* 4(2):17–26
14. Jiang H, Iyengar A, Nahum E, Segmuller W, Tantawi A, Wright CP (2009) Load balancing for sip server clusters. In: Proceedings of the 28th conference on computer communications. IEEE, New York
15. Jin H (2004) Chinagrid: making grid computing a reality. In: Digital libraries: international collaboration and cross-fertilization. Lecture notes in computer science. Springer, Berlin, pp 13–24
16. Karger DR, Ruhl M (2004) Simple efficient load balancing algorithms for peer-to-peer systems. In: Proceedings of ACM symposium on parallelism in algorithms and architectures. ACM, New York
17. Lim B-H, Bianchini R (1996) Limits on the performance benefits of multithreading and prefetching. In: Proceedings of the ACM SIGMETRICS. ACM, New York, pp 37–46
18. Liu H, Wee S (2009) Web server farm in the cloud: performance evaluation and dynamic architecture. In: Proceedings of CloudCom 2009. LNCS, vol 5931. Springer, Berlin
19. McDowell L, Eggers S, Gribble S (2003) Improving server software support for simultaneous multi-threaded processors. In: ACM SIGPLAN symposium on principles and practice of parallel programming. PPOPP 03 ACM, New York
20. Menon A, Santos JR, Turner Y, Janakiraman GJ, Zwaenepoel W (2005) Diagnosing performance overheads in the xen virtual machine environment. In: Proceedings of the 1st international conference on virtual execution environments. ACM, New York, pp 13–23
21. Moretti C, Steinhäuser K, Thain D, Chawla NV (2008) Scaling up classifiers to cloud computers. In: Proceedings of IEEE international conference on data mining. IEEE, New York
22. Mosberger D, Jin T (1998) httpperf—a tool for measuring web server performance. *ACM SIGMETRICS Perform Eval Rev* 26(3):31–37
23. Nae V, Iosup A, Podlipnig S, Prodan R, Epema D, Fahringer T (2008) Efficient management of data center resources for massively multiplayer online games. In: Proceedings of the 2008 ACM/IEEE conference on supercomputing. IEEE, New York
24. Nishimura H, Maruyama N, Matsuoka S (2007) Virtual clusters on the fly—fast, scalable, and flexible installation. In: Proceedings of the seventh IEEE international symposium on cluster computing and the grid. IEEE, New York
25. O'Rourke P, Keefe M (2001) Performance evaluation of Linux virtual server. In: Proceedings of 15th USENIX large installation system administration conference (LISA), pp 79–92. USENIX
26. Padala P, Shin PG, Zhu P, Uysal P, Wang Z, Singhal P, Merchant A, Salem P (2007) Adaptive control of virtualized resources in utility computing environments. In: Proceedings of Eurosys07. ACM, New York, pp 289–302

27. Qian H, Miller E, Zhang W, Rabinovich M, Wills CE (2007) Agility in virtualized utility computing. In: Proceedings of the 3rd international workshop on virtualization technology in distributed computing. ACM, New York
28. Shi X, Pazat J-L, Rodriguez E, Jin H, Jiang H (2008) Dynasa: adapting grid applications to safety using fault-tolerant methods. In: Proceedings of the 17th international symposium on high-performance distributed computing. HPDC2008, ACM, New York, pp 237–238
29. Sotomayor B, Montero RS, Llorente IM, Foster I (2009) Virtual infrastructure management in private and hybrid clouds. In: Internet computing. IEEE, New York, pp 14–22
30. Sotomayor B, Keahey K, Foster I (2008) Combining batch execution and leasing using virtual machines. In: Proceedings of IEEE/ACM international symposium on high performance distributed computing (HPDC). ACM, New York, pp 87–96
31. Wells PM, Chakraborty K, Sohi GS (2006) Hardware support for spin management in overcommitted virtual machines. In: Proceedings of the 15th PACT. IEEE Computer Society, New York
32. Wood T, Shenoy PJ, Venkataramani A, Yousif MS (2007) Black-box and gray-box strategies for virtual machine migration. In: Proceedings of the 4th symposium on networked systems design and implementation. NSDI 2007. pp 164–177. USENIX
33. Yang G, Jin H, Li M, Xiao N, Li W, Wu Z, Wu Y, Tang F (2004) Grid computing in china. *J Grid Comput* 2(2):193–206
34. Zhang Q, Riska A, Sun W, Smirni E, Ciardo G (2005) Workload-aware load balancing for clustered web servers. *IEEE Trans Parallel Distrib Syst* 16(3):219–233