2009 29th IEEE International Conference on Distributed Computing Systems

Automatic Performance Tuning for the Virtualized Cluster System

Chuliang Weng, Minglu Li, Zhigang Wang, and Xinda Lu
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
{*clweng, mlli, felixwang, xdlu*}*@sjtu.edu.cn*

Abstract

System virtualization can aggregate the functionality of multiple standalone computer systems into a single hardware computer. It is significant to virtualize the computing nodes with multi-core processors in the cluster system, in order to promote the usage of the hardware while decrease the cost of the power. In the virtualized cluster system, multiple virtual machines are running on a computing node. However, it is a challenging issue to automatically balance the workload in virtual machines on each physical computing node, which is different from the traditional cluster system's load balance. In this paper, we propose a management framework for the virtualized cluster system, and present an automatic performance tuning strategy to balance the workload in the virtualized cluster system. We implement a working prototype of the management framework (VEMan) based on Xen, and test the performance of the tuning strategy on a virtualized heterogeneous cluster system. The experimental result indicates that the management framework and tuning strategy are feasible to improve the performance of the virtualized cluster system.

1. Introduction

With the development of the computer technology, the multi-core processor gradually becomes popular in the computer system. Virtualization technology [1][2] is a good way to aggregate the functionality of multiple standalone computer systems into a single hardware computer, in order to promote the usage of the multi-core processor. Differing from the traditional system software stack, a virtual machine monitor (VMM) is inserted between the operating system level and the hardware level in the virtualized system. In the virtual machine system, multiple virtual machines (VMs) with a specified individual instance of the operating system are running simultaneously on the top of the VMM. Currently, examples of system virtualization include VMWare [3], Xen [4], Denali [5], etc.

Virtualization technology can be applied to the cluster computing system in order to improve the usage of computing nodes with multi-core processors. Multiple VMs are running on a computing node, which are treated as standalone virtual computing nodes. Besides the potential performance benefit, virtualization makes it easier to migrate workloads between computing nodes in the cluster computing system [6][7]. When a VM being migrated, its entire OS and all of its applications are transferred as one unit from one computing node to the other computing node. This method can avoid many of the difficulties faced by process-level migration approaches. In particular, the interface between a guest operating system and the VMM is related narrow so that it is easy to avoid the problem of residual dependencies [8]. When a computing node is overloaded, a VM on it will be chosen and be migrated to the other underloaded computing node in the cluster system conveniently and seamlessly. The combination of virtualization and migration significantly improves manageability for the cluster computing system.

Virtualization may potentially bring benefit to the cluster system in the aspect of performance and management as discussed above. The challenging issue is how to effectively manage the virtualized cluster system based on virtualization technology for achieving the high performance. The motivation of this paper is to achieve the goal of the automatic performance tuning for the virtualized cluster system. The main contribution of this paper includes as follows. We propose a management framework for the virtualized cluster system to manage all physical computing nodes and VMs on these nodes. Based on the framework, we present an automatic performance tuning strategy to balance the workload in the virtualized cluster system, which can not only balance the resource allocation between VMs in a physical node, also balance the workload between different nodes in the cluster system. In this paper, we assume that applications running on VMs are the web servers.

The rest of this paper is organized as follows. The next section proposes a management framework for the virtualized cluster system, and discusses its implementation (VEMan). Section 3 presents a two-level performance tuning strategy for the proposed management framework. Section 4 presents the local tuning algorithm and the global tuning algorithm for the two-level performance tuning strategy. Section 5 discusses the experimental results. Section 6 provides a brief overview to the related works, and Section 7 concludes the paper.

1063-6927/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDCS.2009.45

2. Management Framework

In this section, we will propose a management framework for the virtualized cluster system, and introduce its implementation (VEMan).

2.1. Virtualization

As depicted in Figure 1, it is a typical system level virtualization architecture [2]. Virtualization provides an additional layer (VMM) between the running operating system and the underlying hardware. The VMM manages the hardware resources and exports them to the operating systems running on them. As a result, the VMM is in full control of allocating the physical resources to the guest operating system. For example, it is the VMM that maps the virtual CPUs (VCPUs) of VMs to the physical cores.

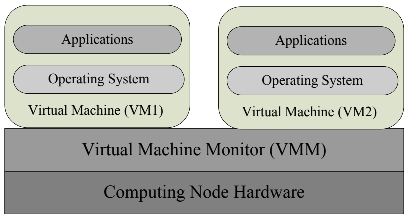

Figure 1. System virtualization architecture

Usually, there are multiple VMs on a single physical computing node in the cluster system. There is a manager VM called as VM0, which is the interface to the system administrator, and is responsible for creating, destroying, modifying the User VMs, which are called as VMUs and are the tuning objects.

2.2. System framework

As depicted in Figure 2, each computing node is virtualized and multiple VMs are running on computing nodes. In each computing node, there are two kinds of *component*s. The *NodeAgent* running in VM0 is responsible for managing the local resources in the computing node and providing the interface to the management node, so that the management node can remotely control the computing node. The *VMAgent* running in each VMU is responsible for gathering the system information on the specified VMU, which is used by the NodeAgent to balance the resource allocation among VMs on the computing node. The third *component* is *ClusterAgent* with Graphic User Interface (GUI), which manages the multiple computing nodes in the cluster system with the help of remote NodeAgents.

Specifically, according to the workload information provided by VMAgents, the NodeAgent dynamically allocates the CPU time, the memory capacity, and the network bandwidth among multiple VMs on the computing node, which is implemented by invoking the system call provided by the VMM. The ClusterAgent dynamically balances the workload among computing nodes by migrating VMUs from the overloaded computing nodes to the underloaded computing nodes.

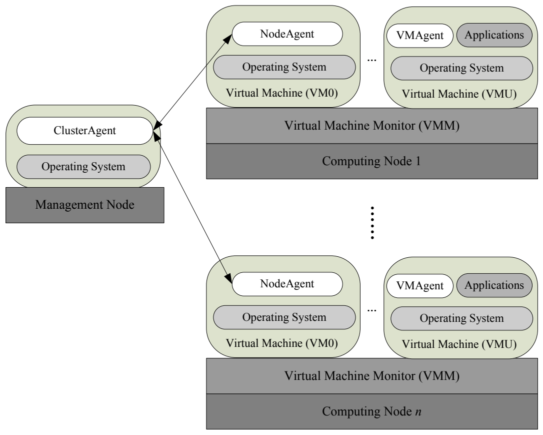

Figure 2. Management framework

The VMAgent running on each VM is adopted for avoiding the problem of the semantic gap [9]. That is, the VMM is full control of allocating local resources while it is the lack of knowledge of the higher-level operating system and its applications. With the VMAgent, the running information about the VM can be obtained intuitively and comprehensively.

2.3. Implementation

We have implemented a working prototype of the proposed management framework, which is called as VEMan (**V**irtual **E**nvironment **Man**ager). In this subsection, we will discuss the implementation briefly.

Xen [4] is adopted as the VMM in VEMan, and Linux is the guest operating system. We choose Linux and Xen because of their broad acceptance and the availability of their open-source codes.

Global management. The design of the ClusterAgent emphasizes convenience and effectiveness. The ClusterAgent's GUI lists information about the computing nodes in the cluster and the VMs on each computing node. The information includes the running information such as the status (running, booted or paused), the usage of CPU and memory, etc, and also includes the hardware information such as the number of CPUs, the capacity of memory, the MAC and IP address of the virtual network. Additionally, some hardware configurations can also be adjusted on the fly. In order to make the ClusterAgent platform-independent, it is implemented in language Python while the GUI is constructed with PyGTK [10].

In order to transfer messages between the ClusterAgent and NodeAgent, we choose a lightweight but powerful remote procedure calls (RPC) protocol XML-RPC as the communication method. XML-RPC works in the client/server mode. In our implementation, the client is the ClusterAgent while the server is the NodeAgent. We have defined and implemented 33 XML-RPC interfaces provided by the NodeAgent to the ClusterAgent. The number will increase in the future if necessary.

For balancing workloads on the computing nodes, the ClusterAgent will determine the migrated VM (see section 4.2) and invoke the XML-RPC interface `virt_vm_migrate()` to accomplish the migration, which parameters include the ID of the migrated VM, its destination computing node, and the transfer port.

System Monitoring. The NodeAgent runs as a daemon in VM0 on each computing node. It is also responsible for gathering the usage information of CPU, memory, and network for each VM on the same computing node. Currently, Xen provides a monitoring application called as XenMon [11] to gather the CPU usage information of VMs. We modified it and integrated it into VEMan, so that the NodeAgent can obtain the CPU usage information of each VM on the computing node. Based on the memory capacity allocated to a VM and the page swap rate in the VM, the NodeAgent may possibly evaluate the memory usage of the VM. Moreover, the virtual firewall-router (VFR) interface is adopted in Xen, and each VM attaches its virtual network interface to the VRF. Therefore, the NodeAgent running in VM0 can monitor the number of bytes transferred on each interface by using `/proc/net/dev` in the system, in order to obtain the network usage information of VMs.

Application monitoring. The VMAgent runs as a daemon in each VMU. It is responsible for gathering the information about applications running on the VM. In this paper, the application on a VMU is the web server. The VMAgent counts the access request amount and the actual access response amount based on the history log. According to the information, the VMAgent estimates the access request amount as the workload on this VM in the near future (see section 4.1). As the VMU is not fixed in a computing node, the estimated workload information of a VM with its MAC address will be sent by multicasting to the NodeAgent, which is on the same computing node.

Local management. The NodeAgent adjusts the CPU and memory allocation among VMs by the `xm` tool provided by Xen (specifically, `xm sched-credit`). However, the network bandwidth control is not provided by the open-source edition of Xen, this function can be implemented by invoking the Linux kernel interfaces in the VM.

3. Performance Tuning Strategy

This section presents a performance tuning strategy for the virtualized cluster system. Specifics regarding the tuning algorithms are detailed in the next section.

As depicted in Figure 3, we adopt a two-level tuning strategy in the virtualized cluster system. After the system initialization, the number of VMs (specifically they are VMUs, hereafter the same) created on each computing node may be customized by the system administrator or be in proportion to its physical computing power.

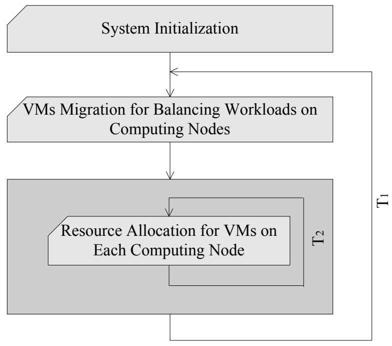
System Initialization
VMs Migration for Balancing Workloads on Computing Nodes
Resource Allocation for VMs on Each Computing Node
T2
T1

Figure 3. Performance tuning

The *local tuning event* occurs in the range of an individual computing node. The application in each VM is a web server, and the workload of a VM can be measured by the number of access requests of the web server. Then the difference between the request amount and the throughput can be used to determine whether the VM is underloaded or overloaded. It is the NodeAgent that performs the resource allocation operation in a computing node. According to the workload information of each VM on the computing node, the NodeAgent will calculate the weight of each VM, and assign the share of the CPU time and the other physical resources to VMs in proportion to their weights.

The ClusterAgent performs the tuning operation at the *global tuning event* in the range of the cluster system. Based on the usage percent information of the physical resources in a computing node, the ClusterAgent determines whether the computing node is overloaded or underloaded. According to this workload information, the ClusterAgent will balance the workloads of computing nodes by migrating the VM from an overloaded node to an underloaded node. For avoiding too frequent migration and reducing the migration overhead, we constrain that the number of the migrated VMs is no more than one at each global tuning event.

The interval of the global tuning event is denoted by T_1, and the interval of the local tuning event is denoted by T_2. As the global tuning overhead is larger than the local tuning overhead, we set that T_1 is an integral multiple of T_2.

It is noted that the resource allocation among VMs determined by the NodeAgent should be based on the number of access requests in the next interval in the ideal condition. However, the access request arrived in the future is not known in advance. The ClusterAgent also encounters the similar problem when determining the balancing operation. Therefore, we will present the specific tuning algorithm based on the proposed tuning strategy in the following section.

4. Tuning Algorithm

In this section, we will describe the tuning algorithms for the ClusterAgent and the NodeAgent, respectively.

The cluster system is a heterogeneous system, and each node is treated as a symmetric multiprocessing (SMP) system, although it is a multi-core system. The number of the computing nodes is denoted by N. The number of physical cores in node i is denoted by $C[i]$, and the CPU speed is denoted by $S[i]$, and the memory capacity is denoted by $RAM[i]$, and the network bandwidth is denoted by $NB[i]$.

4.1. Local tuning algorithm

When a VM is dedicated to running a web server, for a better workload balance, the number of access requests arrived in the VM in the next T_2 will determine the weight of the VM. However, this information is not known in advance.

The access request for a web server has the property of locality and periodicity [12][13]. For estimating the near future workload of a web server, then we adapt a typical learning algorithm to demonstrate the significance of the local tuning. Considering the characteristic of the web access, the modified Roth-Erev learning algorithm is proposed to estimate the amount of access requests arrived in a VM in the next interval. The original version of the Roth-Erev reinforcement-learning algorithm is described in [14], and the modified Roth-Erev algorithm is calibrated with four parameters, a scaling parameter $s(1)$, a recency parameter r, an experimentation parameter e, and a parameter K denoting the number of possible access request amounts.

The modified Roth-Erev algorithm (Algorithm 1) is executed by the VMAgent on each VM, and it is summarized as follows. At the beginning, the VMAgent on VM j assigns an equal propensity $q_{jk}(1) = s(1)X/K$ for each k of all possible request amounts, which total is K, and X is the statistical average value of request amounts in a web server. In addition, an equal choice probability $p_{jk}(1) = 1/K$ is assigned to each of its feasible request amount k. At the first local tuning event, the VMAgent probabilistically selects a feasible request amount k' as the request amount estimation in the following interval. At the next local tuning event, the VMAgent on VM j updates the corresponding parameters according to its estimation accuracy for the last interval, which is determined by the difference between the estimated request amount and the actual request amount.

Algorithm 1 The local tuning algorithm

```
1: At local tuning event $i+1$
2: for VM $j$ on a computing node do
3:     for each propensity $k$ do
4:         $q_{jk}(i+1) \leftarrow (1-r)q_{jk}(i) + E(j,k,k',i,K,e)$;
5:     end for
6:     for each propensity $k$ do
7:         $p_{jk}(i+1) \leftarrow q_{jk}(i+1) \Big/ \sum_{m=1}^{K} q_{jm}(i+1)$;
8:     end for
9:     $p_{jk'}(i+1) = \max_k p_{jk}(i+1)$.
10:    the request amount in the next interval is estimated as $k'$.
11:    $ERA[j] \leftarrow k'$.
12: end for
13: for VM $j$ on a computing node do
14:    $W[j] \leftarrow ERA[j] \Big/ \sum_l ERA[l]$;
15: end for
```

$A(j,k',i)$ is the estimation accuracy for VM j at the local tuning event i, and its estimation of the request amount is k'. $ERA[j]$ is the estimated request amount arrived in the next interval. And E is an updating function reflecting the experience gained from the past estimation activity, which takes the form:

$$E(j,k,k',i,K,e) = \begin{cases} A(j,k',i)(1-e), k = k' \\ q_{jk}(i)\frac{e}{K-1}, k \neq k' \end{cases} \quad (1)$$

Consequently, the NodeAgent will adjust the allocation of the physical CPU, memory, and bandwidth to VM j on the computing node according to the new weight $W[j]$.

4.2. Global tuning algorithm

At the global tuning event, we adopt the concept of cost to select a VM to be migrated from one node to the other node, for balancing the workloads among nodes in the cluster system. The key of the method is to convert the usage of different kinds of resources, such as CPU, memory and bandwidth into a homogeneous cost. We adopt an exponential function for the cost of a VM with a given load [15]. The merit of the exponential function is that the cost of migrating a VM is not only influenced by the workload of the VM itself, but also influenced by the workload of the computing node.

For VM j on node i, the CPU usage percent in the last T_1 is denoted by $CPU_u[i][j]$, and specifically $CPU_u[i][j] = \sum_k T_{VCPU[i][j][k]}/(T_1 \times C[i])$, where $T_{VCPU[i][j][k]}$ denotes the time length of VCPU k of VM j running on the physical CPU in node i in the last T_1. The average used memory capacity of VM j in the last T_1 is denoted by $RAM_u[i][j]$, and the average used network bandwidth is denoted by $NB_u[i][j]$.

The cost of VM j running on node i consists of the CPU cost $cost_c[i][j]$, the memory cost $cost_m[i][j]$, and the network cost $cost_n[i][j]$, and they are defined as follows, respectively.

$$cost_c[i][j] = N^{\sum_k CPU_u[i][k]} - N^{\sum_{k,k \neq j} CPU_u[i][k]} \tag{2}$$

$$cost_m[i][j] = N^{\frac{\sum_k RAM_u[i][k]}{RAM[i]}} - N^{\frac{\sum_{k,k \neq j} RAM_u[i][k]}{RAM[i]}} \tag{3}$$

$$cost_n[i][j] = N^{\frac{\sum_k NB_u[i][k]}{NB[i]}} - N^{\frac{\sum_{k,k \neq j} NB_u[i][k]}{NB[i]}} \tag{4}$$

Then, the cost of VM j running on node i is

$$\mathrm{cost}[i][j] = \mathrm{cost}_c[i][j] + \mathrm{cost}_m[i][j] + \mathrm{cost}_n[i][j] \tag{5}$$

After VM j on node i is migrated to node i', its memory cost $newc_m$ and network cost $newc_n$ can be calculated with above equations, while its CPU cost is calculated as follows because the CPU speed of nodes may be different from each other.

$$newc_c[i'][j] = (N^{\frac{S[i]}{S[i']} CPU_u[i][j] + \sum_k CPU_u[i'][k]} - N^{\sum_k CPU_u[i'][k]}) \tag{6}$$

Then we have the new cost of VM j running on node i' after it is migrated from node i.

$$newcost[i'][j] = newc_c[i'][j] + newc_m[i'][j] + newc_n[i'][j] \tag{7}$$

At the global tuning event, a VM may be chosen to be migrated to balance the workloads on computing nodes in the cluster system. The goal is to reduce the workload unbalance among these computing nodes. The global tuning algorithm is shown as Algorithm 2, where δ is the migration threshold. According to the global tuning algorithm, at most one VM may be chosen to be migrated at each global tuning event.

Algorithm 2 The global tuning algorithm

```
1:  for each global tuning event do
2:      The ClusterAgent getting all information;
3:      $max\_cost\_diff \leftarrow 0$;
4:      $\{fromNode, toNode, peakVM\} \leftarrow 0$;
5:      for each node $i$ do
6:          for each VM $j$ on node $i$ do
7:              $current\_cost \leftarrow cost[i][j]$;
8:              for each node $i'$, $i' \neq i$ do
9:                  $target\_cost \leftarrow new\_cost[i'][j]$;
10:                 $cost\_diff \leftarrow current\_cost - target\_cost$;
11:                 if $cost\_diff > max\_cost\_diff$ then
12:                     $max\_cost\_diff \leftarrow cost\_diff$;
13:                     $fromNode \leftarrow i, toNode \leftarrow i'$;
14:                     $peakVM \leftarrow j$;
15:                 end if
16:             end for
17:         end for
18:     end for
19:     if $max\_cost\_diff > \delta$ then
20:         Migrating VM $peakVM$ from node $fromNode$ to node
            $toNode$;
21:     end if
22: end for
```

Then, we analyze the global tuning algorithm theoretically as follows.

Lemma 1. Minimizing the total cost of all of the computing nodes in the cluster is an optimal solution for reducing the workload unbalance.

Proof. For simplifying the process of proof, only the CPU cost is considered, and the computing node is homogeneous. Then the cost of a computing node is $cost[i][j] = N^{\sum_k CPU_u[i][k]}$, and the total cost of all of the computing nodes is $cost_{all} = \sum_i N^{\sum_k CPU_u[i][k]}$. At the tuning event, the workloads of VMs are fixed, then we have $load_{all} = \sum_i \sum_k CPU_u[i][k]$, which is also a fixed value. According to the property of the average inequality [16], when the workload on each computing node is $load_{all}/N$, the total cost $cost_{all}$ has the minimal value. In this condition each computing node has the equal workload, so the cluster system has an optimal workload balance. As it is a sufficient and necessary condition, therefore minimizing the total cost of all of the computing nodes in the cluster is an optimal solution for reducing the workload unbalance. The conclusion can be easily extended to the heterogeneous situation with more resources' costs considered. ∎

Theorem 1. Algorithm 2 is an optimal solution for balancing the workload under the condition that at most one VM is migrated at each global tuning event.

Proof. With Algorithm 2, the chosen VM $peakVM$ has the maximal cost difference max_cost_diff among all VMs in the cluster. That is to say, the total cost has the maximal decrease after VM $peakVM$ is migrated from its current computing node $fromNode$ to the new computing node $toNode$. Therefore, after this migration, the total cost is the minimal value among all possible migrations at this global tuning event, under the condition that at most one VM is migrated. According to Lemma 1, the migration determined by Algorithm 2 is an optimal solution for reducing the workload unbalance. ∎

According to Theorem 1, we find that the presented global tuning algorithm is an effective method to balance the workloads in the cluster when at most one VM is migrated at each global tuning event. As the overhead of migration is not unneglectable, we argue that this solution has a better tradeoff among the cost and the benefit of the VM migration in the cluster.

5. Performance Evaluation

In this section, we perform some experiments on the virtualized cluster system with workloads.

5.1. Experimental methodology

For studying comprehensively the performance, we firstly test the total throughput of web servers in the virtualized cluster, which is a performance metric for the global workload balance. We also test the throughput of the web server in a single VM, which is a performance metric for the local workload balance.

Workload. To evaluate system performance improvement introduced by the automatic tuning, we perform a series of experiments. The workload in each VM (VMU) is a web server, which provides the web access service. Each VM runs Apache 2.2.3 and PHP 5.2.0, serving dynamic PHP web pages. The PHP scripts are designed to be a mixed physical resources (CPU, memory, and network) intensive. We develop a web client, which will generate continuous web access requests for each VM based on *httperf* [17].

Experimental system. All experiments are executed on a heterogenous cluster connected over gigabit ethernet, which includes 3 Dell servers with dual quad-core Xeon X5310 CPUs and 2GB of RAM, and 3 Dell servers with one dual-core E6550 CPU and 1GB of RAM. Each computing node runs Xen 3.2.1, and all VMs run the Ubuntu 8.04 Linux distribution with the Linux 2.6.18.8 kernel. For implementing the VM migration, iSCSI and NFS (Network File System) are used in the cluster, and a server with dual quad-core CPUs acts as the iSCSI target, and the others act as iSCSI initiators. The iSCSI target is implemented by IET (iSCSI Enterprise Target), and the iSCSI initiator is implemented by open-iSCSI 2.0-870.

5.2. Experimental result

In the experiment, we mainly consider the four scenarios, and there are 18 VMs (VMUs) in the virtualized cluster. The first scenario is that all VMs are equally distributed among all nodes in the cluster, and each VM is deployed and fixed on a computing node all the time. As any automatic tuning is not adopted in this scenario, we call the scenario as `NonTuning`. The second scenario is that only the automatic global tuning is adopted in the cluster system, and the ClusterAgent automatically migrates VMs according to the global tuning algorithm (Algorithm 2), and we call this scenario as `GlobalTuning`, where the global tuning event interval $T_1 = 120s$. The third scenario is that only the automatic local tuning is adopted in the cluster system, and each NodeAgent automatically adjusts the resource allocation among VMs on its computing node according to the local tuning algorithm (Algorithm 1), and we call this scenario as `LocalTuning`, where the local tuning event interval $T_2 = 20s$. The last scenario is that the automatic global and local tunings are simultaneously adopted in the cluster system, and we call this scenario as `G&LTuning`.

Firstly, we test the total throughput of 18 web servers in the cluster system when the total of access requests is increasing in the fixed time length (40 minutes). For testing the influence of the unbalanced workload on the performance, the rate of access requests fluctuates with the time, and initially the average rate of access requests is relatively higher for VMs on nodes with the slower speed. The experimental result is shown as Figure 4.

With the total of access requests increases in a fixed period, the workload of the cluster becomes heavier. The physical resources are less idle in the scenario with the better workload balance, as a result, the total of the access responses (throughput) will be larger if the system has a better workload balance. According to Figure 4, the performance of `G&LTuning` is the best among all four scenarios, while the performance of `NonTuning` is the worst. When the workload is not very heavy, `LocalTuning` outperforms `GlobalTuning`, because the overhead of the local resource reallocation is relatively less. Otherwise, `GlobalTuning` outperforms `LocalTuning`, because `GlobalTuning` balances workloads in the global range of the cluster system, while `LocalTuning` performs only in the local range of individual computing nodes.

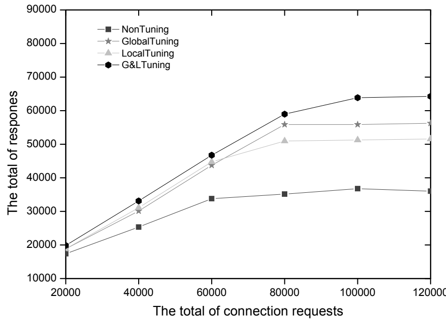

Figure 4. The total throughput

Now we study the performance from the aspect of a single computing node. We monitor the CPU usage percent of a fixed computing node in the same period in the four scenarios, which lasts for 40 minutes. The experimental result is shown in Figure 5, where the value of a point is the average value of the CPU usage in a minute. The CPU usage percent fluctuates acutely when there is no automatic tuning in the system. The automatic tuning may smooth the fluctuation of the resource usage to some degree, which is caused by the application in the system. This result is beneficial to improve the quality of service (QoS) of applications such as web servers running in the system.

Then we turn our attention to the performance of an

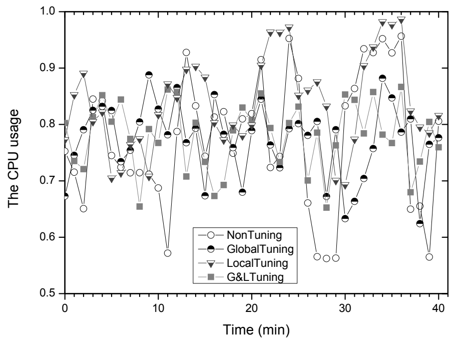

Figure 5. The CPU usage of a computing node

individual VM in the system. Therefore, we also record the number of access requests and the throughput of a fixed VM in the cluster in the four scenarios, when we perform the above experiments to test the total throughput of the cluster. The rate of response is the value of the throughput divided by the request amount, and is adopted to evaluate the performance of a single VM. It is depicted as Figure 6. As the automatic tuning can dynamically adjust the computing capacity of a VM according to the information of the realtime workload, the rate of response is relatively higher in the tuning scenarios. Especially, `G&LTuning` has the best rate of response for the specified VM. Moreover, `GlobalTuning` outperforms `LocalTuning` when the system's workload is relatively heavy.

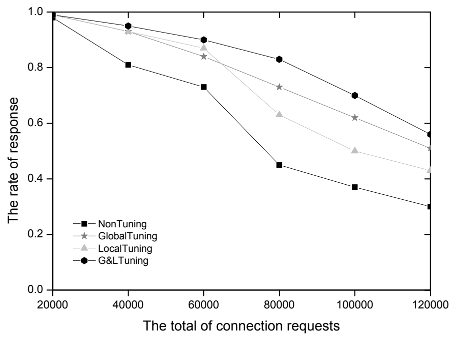

Figure 6. The throughput of a VM

In summary, `G&LTuning` has the best performance among the four scenarios, not only from the aspect of the whole performance, but also from the aspect of the local performance. Also it is helpful to improve the QoS of applications running in the virtualized cluster system. It is noted that `GlobalTuning` can also be beneficial to improve the performance of the system, especially when the workload of the system is relatively heavy. As a result, it is significant to perform the global workload balance by the VM migration with the global tuning algorithm, even if there is no enough information about applications to perform the local tuning.

6. Related Work

The performance tuning issue in the virtualized cluster had close relationship with the task scheduling and process migration in the traditional cluster.

The cluster system can offer high performance, high throughput and high availability at a relatively low cost for the corresponding application scenario. The Condor system [18] is a high throughput cluster system, who aims to maximize the utilization of workstations with the usage of the job migration. Condor adopts the checkpoint technology to save enough information for the migrated job before transferring it to another workstation. Along with Condor, there are a wide variety of powerful batch execution systems such as LoadLeveler (a descendant of Condor), LSF and PBS. Currently, an interesting project is Tashi [19], which is a cluster management system based on virtualization for the emerging cloud computing.

For improving the performance of the virtualized cluster system, VMware's Distributed Resource Scheduler (DRS) [20] uses migration to perform automated load balancing in response to CPU and memory pressure. A userspace application is used to monitor memory usage similar to VEMan's VMAgent, but unlike VEMan, it cannot utilize application logs to adjust the resource allocation among VMs on the computing node. In the other interesting related work [21], the black-box and gray-box strategies are presented in Sandpiper. These strategies are used to detect hotspot, that is, the usage of one resource exceeds a threshold. Differing from estimating a single hotspot, our tuning strategy implemented the global and local workload balance in the virtualized cluster. For the improving the MapReduce performance in the virtualized heterogeneous environments, the LATE scheduling algorithm [22] is presented, and it can improve the response times of Hadoop (an open-source implementation of MapReduce). Another interesting work is STEP [23], where an effective way is presented and implemented to improve the utilization of storage server resources through prefetching in storage servers for clients.

7. Conclusion

Virtualization provides a potential method to effectively manage the multi-core cluster system and promote its usage performance. Automatic performance tuning may provide significant benefits in the virtualized cluster system by adjusting the physical resource allocation among VMs and enabling the VM migration to achieve workload balance.

In this paper, we propose a management framework for the virtualized cluster system, in which the three types of modules: ClusterAgent, NodeAgent, VMAgent are used to monitor the running status of the virtualized system, and determine the resource allocation and the VM migration. For guiding the automatic performance tuning, we present a two-level tuning strategy with the two corresponding algorithms. We have implemented a working prototype of the management framework as VEMan based on the VMM Xen. Experiments indicate that the automatic performance tuning is a viable technique for reducing the workload unbalance between VMs and between computing nodes in the cluster system.

Although the proposed automatic performance tuning framework and strategy discussed in this paper may improve the virtualized cluster system to some degree, there is still much room for the further improvement. Specifically, the local tuning algorithm is focused on the application of web servers, although there are a variety of applications in the cluster system. In the future, we will optimize further the tuning method for the web server application while find the tuning method for more types of applications such as the high performance computing application in the cluster system.

Acknowledgment

This work was supported partly by National Key Basic Research and Development Plan (973 Plan) of China (No. 2007CB310900), and Huawei Science and Technology Foundation, and National Natural Science Foundation of China (No. 90715030, 90612018, and 60503043). We would like to thank the anonymous reviewers for their thoughtful comments and suggestions.

References

[1] P. H. Gum, "System/370 extended architecture: Facilities for virtual machines," *IBM Journal of Research and Development*, vol. 27, no. 6, pp. 530–544, 1983.

[2] J. E. Smith and R. Nair, *Virtual Machines: Versatile platforms for systems and processes*. Elsevier, 2005.

[3] C. A.Waldspurger, "Memory resource management in VMware ESX server," in *Proceedings of the 5th symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[4] 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)*, 2003.

[5] A. Whitaker, M. Shaw, and S. Gribble, "Denali: Lightweight virtual machines for distributed and networked applications," in *Proceedings of the USENIX Annual Technical Conference*, October 2002.

[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (NSDI)*, 2005.

[7] VMWARE, "Migrate virtual machines with zero downtime," http://www.vmware.com/products/vi/vc/vmotion.html.

[8] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process migration," *ACM Computing Surveys*, vol. 32, no. 3, pp. 241–299, 2000.

[9] P. M. Chen and B. D. Noble, "When virtual is better than real," in *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS)*, 2001.

[10] PyGTK, "PyGTK: GTK+ for Python," http://www.pygtk.org/.

[11] G. Diwaker, G. Rob, and C. Ludmila, "Xenmon: Qos monitoring and performance profiling tool," HP Lab, Tech. Rep. HPL-2005-187, 2005.

[12] M. Arlitt and C. Williamson, "Web server workload characterization: the search for invariants," *SIGMETRICS Perform. Eval. Rev.*, vol. 24, no. 1, pp. 126–137, 1996.

[13] J. Dilley, "Web server workload characterization," HP Lab, Tech. Rep. HPL-96-160, 1996.

[14] A. Roth and I. Erev, "Learning in extensive form games: Experimental data and simple dynamic models in the intermediate term," *Games and econmic behavior*, no. 8, pp. 164–212, 1995.

[15] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, "Online machine scheduling with applications to load balancing and virtual circuit routing," in *Proceedings of the ACM Symposium on Theory Of Computing (STOC)*, 1993.

[16] G. Chrystal, *Algebra, An Elementary Textbook.* AMS Bookstore, 1999, vol. II.

[17] httperf, "http://www.hpl.hp.com/research/linux/httperf/."

[18] M. Litzkow, M. Livny, and M. Mutka, "Condor - a hunter of idle workstations," in *Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS)*, June 1988.

[19] Tashi, "http://incubator.apache.org/tashi/."

[20] VMWare, "Dynamic resource scheduler," http://www.vmware.com/products/vi/vc/drs.html.

[21] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proceedings of the Fourth Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.

[22] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008, pp. 29–42.

[23] S. Liang, S. Jiang, and X. Zhang, "STEP: Sequentiality and thrashing detection based prefetching to improve performance of networked storage servers," in *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS)*, 2007.