SLA-Driven Adaptive Resource Allocation for Virtualized Servers

Wei ZHANG,†††† Li RUAN,†††† Mingfa ZHU,†††† Limin XIAO,†††† Jiajun LIU,†††† Nonmembers, Xiaolan TANG,†††† Member, Yiduo MEI,†††† Ying SONG,††††, and Yuzhong SUN,†††† Nonmembers

SUMMARY In order to reduce cost and improve efficiency, many data centers adopt virtualization solutions. The advent of virtualization allows multiple virtual machines hosted on a single physical server. However, this poses new challenges for resource management. Web workloads which are dominant in data centers are known to vary dynamically with time. In order to meet application’s service level agreement (SLA), how to allocate resources for virtual machines has become an important challenge in virtualized server environments, especially when dealing with fluctuating workloads and complex server applications. User experience is an important manifestation of SLA and attracts more attention. In this paper, the SLA is defined by server-side response time. Traditional resource allocation based on resource utilization has some drawbacks. We argue that dynamic resource allocation directly based on real-time user experience is more reasonable and also has practical significance. To address the problem, we propose a system architecture that combines response time measurements and analysis of user experience for resource allocation. An optimization model is introduced to dynamically allocate the resources among virtual machines. When resources are insufficient, we provide service differentiation and firstly guarantee resource requirements of applications that have higher priorities. We evaluate our proposal using TPC-W and Webbench. The experimental results show that our system can judiciously allocate system resources. The system helps stabilize applications’ user experience. It can reduce the mean deviation of user experience from desired targets.

key words: resource allocation, virtualized servers, user experience, optimization theory

1. Introduction

To reduce their management cost, more and more companies rent resources from data centers. The servers in data centers are commonly used for running business-critical applications such as enterprise resource planning, database, customer relationship management and e-commerce applications. Recent advances in virtualizing commodity hardware are changing the structure of the data centers, like Xen [1], VMware [2], KVM [3], OpenVZ [4] and Linux VServer [5]. A physical server is transformed into one or more virtual machines that dynamically share the underlying hardware resources, and applications run within these isolated environments. Server virtualization supports fine-grained resource allocation. Each virtual machine can be given a partition of the underlying resource capacity (CPU and Memory) and each virtual machine is subject to runtime resource allocation.

However, a key challenge that comes with virtualization is adaptively provision virtualized applications with resource commensurate with their workload demands, especially in the context of the time-varying workloads. Indeed, if each application is properly provisioned, additional resources can be used otherwise, e.g., to run additional applications [6]. This issue attracts great research interests from industry [7]–[9] and academia [10]–[18]. But most of these works are based on the concept of resource utilization. They always define desired resource utilization and decide the physical resource partition through evaluating the difference between the desired resource load and the monitored load.

Resource allocation based on resource utilization has several disadvantages. Firstly, in cloud computing environment, more and more applications emerge and applications become much larger and more complex. How to determine desired resource utilization is very difficult. If the desired resource utilization that is defined is higher, it may result in performance violations for applications. If the desired resource utilization that is defined is lower, it may waste resources. Secondly, in virtualized environments, accurate utilization sampling is more difficult, since it needs to filter hypervisor overheads [19]. Thirdly, when multiple VMs or processes co-reside on a physical machine, the measured resource utilization may provide a poor estimate of the actual requirement [20]. Lastly, virtual machine monitor does not guarantee performance isolation among virtual machines. Performance interference makes that resource utilization is a poor indicator of application performance [21]. This is because the resource requirements of virtual machine is dependent not only on its own demand, but also on the demand of the co-located virtual machines along the time.

In [22], the authors find that if system high response time beyond the user’s tolerance, the service providers face with the risk of promoting users’ frustration. Furthermore, this will bring profit and business losses. Poor user experience results in the likelihood that users turn to competitors. What’s more, this will bring a negative impact on the company’s products and the company itself. Therefore, a failure to meet users’ on-line SLA requirements may affect the competitiveness of the company, commercial status and...
Based on the above analysis, triggering resource reallocation directly based on the information of user experience not only is reasonable, but also has practical significance. User experience attracts more and more attention. It is an important manifestation of SLA. This paper presents a new resource allocation scheme directly based on user experience. Response time is a key user-perceived metric [23] and maintaining reasonable response time is imperative. In this paper, we define server-side response time as user experience. We rely on control theory and optimization theory to decide how to divide physical resources among virtual machines. Not all applications are treated equally. When resource competition occurs, we firstly guarantee resource requirements of applications that have higher priorities. In this paper, we mainly consider CPU allocation, which could be generally extended to other resources such as I/O, memory. We deploy our system on Xen-based environment. The experimental results show that our method can help stabilize user experience.

This paper has the following major contributions. 1) We propose a new resource allocation scheme directly based on user experience which is obtained with low overhead, also it is transparent to application and operating system. 2) We design a resource allocation control architecture that relies on optimization and control theory as a theoretical foundation. In our model, we consider service differentiation and firstly guarantee resource requirements of applications that have higher priorities, when resource competition occurs. 3) We design a prototype system and deploy it on Xen-based environment. The experimental results show that the validity of the proposed approach.

The rest of the paper is organized as follows. In Sect. 2, we discuss related work. Section 3 introduces our system architecture, and Sect. 4 we give detailed specification on each component. We evaluate our system performance in Sect. 5. The discussion and ongoing works are summarized in Sect. 6.

2. Related Work

Currently, dynamic resource provisioning of enterprise applications in virtualized servers arouses great interest from industry and academia.

In industry, VMware develops a resource management system, named VMware DRS [2]. VMware DRS that continuously monitors utilization across the resource pools and intelligently allocates available resources among virtual machines based on resource utilization. The HP-UX Workload Manager (WLM) is a component of the HP Virtual Server Environment [7]. The administrator specifies the target of the resource usage. As the applications run, WLM compares the actual resource usage against the target and then automatically adjusts processors allocations for the workload groups to achieve the goal. IBM develops IBM Tivoli Provisioning Manager. It is also a system base on resource utilization for resource allocation.

In academia, there is also a lot of related work [10]–[18]. J. Rolia et al. [10] design a workload controller for resource allocation. The workload controller algorithm introduces parameters including minimum and maximum allocation, gain, and a range of utilization of allocation values. They use a predefined threshold for deciding whether current CPU allocation for the workload is sufficient or not. In order to meet application-level QoS goals, P. Padala et al. use SISO (Single Input Single Output) [11] and MIMO (Multiple Input Multiple Output) [12] controller to dynamically adjust CPU and disk I/O resources. However, they decide resource allocation by comparing the desired utilization level and the monitored utilization level. Y. Song et al. [14] rely on utility function to decide CPU and memory resources allocation. They define the desired resource utilization thresholds according to applications that are running on the virtual machines. J. Xu et al. [15] use fuzzy prediction method to forecast future resource demands based on the current resource utilization. J. Heo et al. [16] use feedback control to adjust memory allocation for virtual machines in a consolidated environment. They define a desired level of memory utilization for the virtual machine and design a memory utilization controller that ensures the value of memory allocation to remain with a specified range between the minimum amount and the maximum amount. Z. Wang et al. [17] use feed-forward controller to decide CPU allocation according to the desired utilization targets. These works are almost based on desired resource utilization for resource reallocation. According to the above analysis in Sect 1, resource allocation based on resource utilization has many disadvantages.

Response time is a key user-perceived metric [23]. In earlier study of resource management in virtualized servers, X. Liu et al. [24] design performance controller to decide CPU allocation. However, they gather response time from clients, and then return to servers. This approach has a big network delay. The application performance degradation is likely to be caused due to transmission bottlenecks of the network and routing, rather than the insufficient resources on the server side. The method can only be used to adjust resources in off-line way. T. Horvath et al. [25] measure end-to-end latencies by adding a module and modifying apache source code. But this method is not transparent to the application. Our approach has some similarities with Sangpetch’s work [26]. However, their work requires to establish a dedicated sensor virtual machine to analyze packets and obtain response time. There is a great deal of overhead. They also don’t consider the constraints of limited resources. This is contrary to reality that the capacity of physical machine is limited. We design an optimization model that has the constraint of the limited resources. We firstly guarantee resource requirements of applications that have higher priority, when resource competition occurs.

Based on the above analysis, we propose a resource allocation control architecture that allocates resources based on user experience. We calculate current satisfaction level according to our designed satisfaction model. The method
of monitoring user experience has low overhead, which is transparent to application and operating system. We rely on optimization and control theory as a theoretical foundation for resource allocation. In our model, we consider service differentiation and firstly guarantee resource requirements of applications that have higher priority, when resource competition occurs.

3. System Architecture

As seen in Fig. 1, multiple virtual machines run on the physical server. Each virtual machine contains an application or a component of the application. Each such application is assumed to specify a desired response time. The goal of system is to ensure that percentile of the response time of application requests is no greater than the desired target percentile of response time.

Since each resource has a finite capacity and the application workload can exceed capacity during the periods of heavy loads. When resources are relatively insufficient, we provide service differentiation and give priorities to high-priority applications.

To solve the above problems, we design a system (illustrated in Fig. 1). The system consists of five components: monitor, analyzer, resource request controller, resource arbiter controller, and actuator. The monitor periodically collects information of request packets and response packets. We design a satisfaction model. The analyzer analyzes and calculates current satisfaction level for every virtualized application according to our satisfaction function. In order to obtain the desired satisfaction level, the resource request controller decides to increase or decrease the amounts of resource according to our designed performance model. The resource request controllers submit resource requirements to resource arbiter controller. When resource requirements of all virtual machines are less than resource capacity of physical machine, we allocate resources according to demands. When lack of resources or resource competition occurs, we provide service differentiation. Taking into account the limited resource amounts, we design an optimization model to maximize the satisfaction level for the entire system and give priority to high-priority service.

We use Xen as our hypervisor in this paper. However, there are two differences. Firstly, Sangpetch et al. [26] create and set up a dedicated sensor virtual machine for each virtual machine which is running application or application component. They use xtables [27] or open vswitch [28] to replicate these packets in domain0 and transfer these replicated packets to the corresponding sensor virtual machine. Their method has three drawbacks: 1) Each sensor virtual machine need to be allocated certain resources for normal running, e.g. operating system consumes certain resources for normal running. 2) Each packet requires additional replication in domain0, and need to be transmitted from back-end device driver which is located in domain0 to front-end device driver which is located in the corresponding sensor virtual machine. This may bring about a lot of overhead. 3) The virtual machine monitor or hypervisor have provided the interfaces of regulating resource allocation. The domain0 have been given this right. Other virtual machines have no such right. So the information in the sensor virtual machine needs to be transferred out to the domain0. The additional transmission also brings about overhead. We analyze the packet path and the working principle of tshark. We find the following interesting phenomenon: 1) Tshark grabs packets from data link layer. 2) The virtual bridge located in domain0 works in the data link layer. All packets go through the virtual bridge and are routed to the corresponding virtual machine which is running application or application component. Based on the above findings, we easily obtain necessary information from domain0, as shown in Fig. 2. Our work avoids replication overhead and transmission overhead for packets. Our work does not keep these additional sensor virtual machines. Secondly, by default, the VCPU number of domain0 is equal to the number of physical CPUs. The capabilities of modern servers are continuing to grow as the number of CPU cores on a typical computer rises from one or two to several dozen. Our monitor located in the domain0 is implemented with the multi-threaded manner and can achieve good parallelism by taking full advantage of the ability of multi-core.

![Fig. 1 System architecture.](image-url)
4.2 Analyzer

The application’s response time is determined by recording the timestamp of packets. Two timestamps are used, request timestamp and response timestamp. The time difference is used as the server-side response time. We assume that network transmission delay is constant and are only concerned with server-side response time. We mark the same connection by a four tuple <source ip, destination ip, source port, destination port>. Firstly we introduce the following notions and concepts:

- \( N \): The number of hosted applications.
- \( R_{\text{total}} \): Total resources of CPU in a physical server.
- \( R_{\text{min}} \): The minimum threshold of resource allocated to application \( i \), which is used to guarantee the normal running of the operating system and application hosted in the VM.
- \( t \): The controlled time interval
- \( R_{ij} \): The really amount of resource for application \( i \) in \( j \) th time interval.
- \( A_i \): Application \( i \)
- \( W_i \): The priority of application \( i \). The higher the value is, the greater the priority is.
- \( D_i \): The desired response time of application \( i \).
- \( d_{ij} \): The actual response time of application \( i \) in the \( j \) th time interval.
- \( \xi_i \): Tolerable time deviation from the desired response time about application \( i \).
- \( T_{ij} \): Throughput of application \( i \) in the \( j \) th time interval.
- \( N_{ij} \): The number of requests for application \( i \) which response time is located in \( [D_i - \xi_i, D_i + \xi_i] \) in the \( j \) th time interval.
- \( L_{ij} \): The number of requests for application \( i \) which response time is located in \( (0, D_i - \xi_i] \) in the \( j \) th interval.
- \( U_{ij} \): The number of requests for application \( i \) which response time is located in \( (D_i + \xi_i, +\infty) \) in the \( j \) th interval.
- \( \alpha_i \): The maximum tolerance of application \( i \).

Analyzer statistics the number of packets \( N_{ij} \), \( L_{ij} \) or \( U_{ij} \) for each application in each time interval. Such information is passed to resource request controller.

4.3 Resource Request Controller

The resource controller calculates the current satisfaction level of application for each application as shown in Fig. 3. Based on its current resource allocation, we use the utility notion to represent the satisfaction level of each application. In the simplest case, we model a satisfaction function to represent the satisfaction level.

Satisfaction function:

\[
P_{ij} = \frac{N_{ij}}{T_{ij}}
\] (1)

We define parameter \( \alpha_i \) which represents the worst tolerance level for application \( i \). That is, if \( P_{ij} > 1 - \alpha_i \), satisfaction with the application \( i \) is met.

The content grows with more requests which the observed response time falls into \([D_i - \xi_i, D_i + \xi_i]\). If \( Q_{ij} = \frac{L_{ij}}{T_{ij}} > \alpha_i \), the application is considered to be over provision, we should decrease resources for that application. This is because service providers can get regular income when the response time falls into \([D_i - \xi_i, D_i + \xi_i]\). When the response time falls into \((0, D_i - \xi_i)\), the service providers can’t acquire additional revenue. On the contrary, the cost becomes higher due to taking up more resources, so net income will become less.

If \( M_{ij} = U_{ij}/T_{ij} > \alpha_i \), the application is considered to be under provision, we should increase resources for that application. Otherwise, the application performance will be severely affected.

In order to determine the resource amounts of increasing or decreasing, we design a performance model between response time and resource. In [30], X. Wang et al. show that the relationship between response time of web requests and the resource amounts is normally nonlinear due to the complexity of computer systems. Instead of directly using \( d_{ij} \) and \( R_{ij} \) to model the system, we build a linear model by using their differences.

- \( \Delta d_{ij} \): Average response time of application \( i \) in \( j \) time interval.
- \( \Delta R_{ij} = ||R_{ij} - R_{ij-1}|| \): Deviation from the desired response time.

We use the following difference equation to model the relationship between performance and resource.

\[
\Delta R_{ij} = \beta_i (1 - P_{ij}) * \Delta d_{ij} + r_i
\] (2)

Note that the model represented in formula (2) is adaptive, because the model parameters \( \beta_i \) is also a function of control interval \( j \). This parameter is updated at the end of
each time interval using the recursive least squares (RLS) method. The model is updated recursively instead of being computed from scratch every interval. The time that it consumes for this computation can be negligible because of the recursive nature.

As shown in Fig. 4, when application satisfaction level is greater (the more requests fall into $[D_i - \xi, D_i + \xi]$) and the average response time is closer to desired response time, the fewer resource amounts need to be increased or decreased. On the contrary, when application satisfaction level is worse, more resource amounts need to be increased or decreased.

4.4 Resource Arbiter Controller

The resource arbiter controller calculates final resource allocation according to the resource requirements of each virtual machine and the resource capacity of physical machine. When the total required resource amounts of all virtual machines are less than resource capacity, we allocate resource for each virtual according to the requirement. When the resource competition occurs, we design an optimization model to determine the resource shares dynamically. The overall system goal is that the total system-wide content is maximized. The request arbiter controller module needs to determine the resource share $R_{ij}$ for application $i$ in $j$ th time interval such that the system content level is maximized. The problem is abstracted as the following constrained optimization problem:

$$\text{Maximize}_{(R_{ij})} = \sum_{i=1}^{N} W_{ij} P_{ij} \tag{3}$$

s.t.

$$\sum_{i=1}^{N} R_{ij} \leq 1$$

$$R_{ij, \text{min}} \leq R_{ij} \leq 1 \quad i = 1, 2, ..., N$$

Dynamic resource allocation algorithm is shown as follows.

```
Input: req_{ij}, R_{ij-1}, i = 1, 2, ..., N
Output: \Delta R_{ij} should be increased or decreased for each application
Algorithm:
Sort applications according to the priority, the higher the priority is, the former the position is
while \(1\)
{
    for (i=1, i<N; i++)\n    {
        Count the numbers of N_{ij}, L_{ij}, or U_{ij} ;
        Calculate P_{ij}, Q_{ij}, or M_{ij} ;
        Compare and if \((P_{ij} > 1 - \alpha)\)
        Continue;
        else if \((Q_{ij} > \alpha)\)
        Calculate the amounts of resource that should be decreased according to formula \((2)\);
        else if \((M_{ij} > \alpha)\)
        Calculate the amount of resource that should be increased according to formula \((2)\);
    }
    if \(\sum_{i=1}^{N} R_{ij} <= 1\)
    Allocate resource on demand according to calculated values;
    else
    Allocate resource according to formula \((3)\), give priority to high-priority application;
    Notify Actuator to make specific actions for resource allocation;
    sleep \(t\);
}
4.5 Actuator

Actuator adjusts the hypervisor parameters as specified by the resource arbiter controller. Currently our actuator can control the amounts of CPU shares for virtual machines on the physical machine. In our test environments, we use xen as our hypervisor. Credit scheduler is the default scheduler, which is a proportional fair share CPU scheduler. The credit scheduler assigns each virtual machine a weight and optionally a cap. The weight indicates the relative CPU allocation of a virtual machine. The cap optionally fixes the maximum amount of CPU that a virtual machine will be able to consume, even if the physical system has idle CPU cycles. The cap is expressed in percentage of one physical CPU: 100 is 1 physical CPU, 400 is 4 CPUs, etc. The default value of cap is 0, representing no upper cap. In this paper, we use default value for cap, which means there is no upper cap. We give an example that shows how we adjust the weight for each virtual machine according to the results of formula 3. We assume a physical machine that has one PCPU (physical CPU). There are two virtual machines running on it, which are virtual machine A and virtual machine B. Each virtual machine has one VCPU (virtual CPU). Initially, we allocate 50% of PCPU for each virtual machine. We use the digital number 256 represents the 1% of PCPU. The initial value of weight is 12800 for each machine. We assume that the results are that the virtual machine A should be increased 5% CPU and the virtual machine B should be decreased 10% CPU. We adjust that the weight for virtual machine A should be 12800+256*5=14080 and the weight for virtual machine B should be 12800-256*10=10240.

5. Evaluation

To evaluate the effectiveness of our method, we implement
the architecture and algorithms on a testbed that host two applications in a virtualized server environment. Experimental results showed that our proposed methods are valid and effective in such an environment.

5.1 Testbed

As shown in Fig. 5, our testbed consists of three servers. Each server contains two 2211 MHz 8 Core AMD Opeteron (tm) processors with 512 KB of cache, 32 GB of RAM, one Gigabit Ethernet interface card, two local SCSI disks. One server is use as test machine and the other two as clients. These servers are connected with a Gigabit Ethernet. The test server has two virtual machines. Each virtual machine is allocated 1 GB of RAM with two virtual CPU.

The implementation of system prototype is based on xen-4.1.0, which can be easily extended to other hypervisors. The deployed operating system on all systems is centos 5.2 with Linux 2.6.18.8 kernel. The capturing tool is tshark-0.99.6. Inside the management domain (domain0) of the test physical server, we use xentop tool to get CPU statistics of each domain, and use sysstat tool to get network statistics at fixed intervals. The version of sysstat is 9.1.7.

In order to compare overhead with Sangpetch’s work [26], we implement the method of acquiring packet information in [26]. The open Vswitch package is openvswitch-1.1.0pre2.tar.gz and the version is service ver. 0.91.

5.2 Workloads and Benchmarks

To investigate our method’s efficiency in virtualized server environment, we design three sets of experiments. In the first set of experiments, we measure the overhead compared with Sangpetch’s work [26]. In the second set of experiments, we measure the deviation of user experience from desired targets using our system compared with not using, when the total required resource amounts of all virtual machines are less than resource capacity. In the last set of experiments, we measure the deviation of user experience from desired targets using our system compared with not using, when the resource competition occurs.

We choose two typical server workloads in modern data center running in virtual machine in our experiments. They are web server and database server. Web server is pervasive in modern data centers and is a representative workload for consolidation. Apache is used as the back-end server and the version is 2.2.19. We use the Webbench [29] to send client requests for web document of size 1kB. Cherkasova [19] shows that web server performance is CPU bound under a mix of small size files. Database is needed to support running transactional workloads in many modern applications. TPC-W [30] is an E-Commerce benchmark that models after an online book store. It consists of two tiers, that is front-end application tier and the back-end database tier. The back-end database used in the experiment is Mysql v5.0.92, which is primary CPU-intensive. The front-end application used is Tomcat v6.0.14, which is running together with client emulator.

5.3 Experiments Results and Analysis

Our evaluations suggest that our system can be used to maintain the service level objectives for the hosted applications. Compared with [26], our work has lower overhead. When the resource requirements of all virtual machines are less than the capacity of the physical machine, all applications have good service level objectives. When resource competition occurs, the applications that have higher priorities have less deviation than the applications that have lower priorities.

In the first set of experiments, compare our work with [26] with respect to overhead.

In Fig. 6, curve sensor represents the CPU consumption of domain0 when acquiring response time in [26]. Curve ours represents the CPU consumption of domain0 in our designed method and curve no-acquired represents the CPU consumption of domain0 not taking any method to acquire response time. Sensor brings about 30% higher overhead to domain0 compared to ours. Figure 7 shows that average CPU utilizations in domain 0 are 60.7%, 61.4%, and 88.9%, respectively. This is because: 1) Each packet requires additional replication in domain0 in [26], and need to be transmitted from back-end device driver which is located in domain0 to front-end device driver which is located in the corresponding sensor virtual machine. This may bring...
ZHANG et al.: SLA-DRIVEN ADAPTIVE RESOURCE ALLOCATION FOR VIRTUALIZED SERVERS

about a lot of overhead. 2) Domain0 in [26] communicates with each sensor virtual machine to acquire the status information of the corresponding service virtual machine. This further increases the overhead in domain0. We analyze the packet path and the working principle of tshark. All packets go through the virtual bridge located in domain0. Tshark grabs packets from data link layer. And the virtual bridge works in the data link layer. We easily obtain necessary information directly from the virtual bridge in domain0. Our method avoids packet replication and communication with each sensor virtual machine in domain0. So our method has lower overhead compared with [26]. Our method is implemented in multi-thread manner and has the ability to use multi-core. This is because by default the VCPU number of domain0 is equal to the number of physical CPU.

Figure 8 shows that sensor virtual machine has about 70% CPU utilization. In [26], each sensor virtual machine need to be allocated certain resources for normal running. The front-end device driver in sensor virtual machine receives packets from the back-end device driver in domain0. Each sensor virtual machine needs to transmit out the status information which it acquires. The virtual machine monitor or hypervisor have provided the interfaces of regulating resource allocation. The domain0 has been given this right. Other virtual machines like sensor virtual machines have no such right. So the information in the sensor virtual machine needs to be transferred out to the domain0.

In our method, we have not these sensor virtual machines. So the overhead of sensor virtual machines does not exist.

Our evaluations suggest that our system can be used to maintain the service level objectives for hosted applications. In the experiment, we set the demand so that the expected response time for web service should be 750 ms while the response time for DB service should be 35 ms. We assume the DB service has the higher priority. We expect that our system will try to adjust CPU allocation so that both SLAs of two services could be met. When resource competition occurs, the system gives priority to DB service.

In the second set of experiments, we measure the deviation of user experience from desired targets using our system compared with not using, when the total requested resource amounts of all virtual machines are less than resource capacity.

In Fig. 9, the curve without control represents the results when we don’t use our system. The curve with control represents the results when we use our system. With our control, the SLA of web service can be satisfied as our system adjusts the CPU share through tracking the actual response time. When the actual response time is higher than interval of the expected response time, we increase CPU resource. When the actual response time is lower than interval of the expected response time, we decrease CPU resource. The amounts of increase or decrease are decided by the formula (3).

Figure 10 shows that the maximum deviation percentage for web service is 57% without control. However, the maximum deviation percentage is 9.3% with control.

In Fig. 11, with our control, the SLA of DB service can be satisfied through adjusting the CPU share, according to the deviation of the actual response time from the interval of the expected response time for DB service.

Figure 12 shows that the maximum deviation percentage for DB service is 51% without control. However, the maximum deviation percentage is 14% in the stabilization process. Initially, the deviation percentage is higher than the stabilization process. This is because that it attempts to find the stable operating points.

In Table 1, the absolute mean deviations from the
Table 1  Mean deviations when resource competition doesn’t occur.

<table>
<thead>
<tr>
<th>Service</th>
<th>Mean Deviation from SLAs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Without Control</td>
</tr>
<tr>
<td>DB</td>
<td>9.82</td>
</tr>
<tr>
<td>Web</td>
<td>173.16</td>
</tr>
</tbody>
</table>

target response time for DB service and web service are 9.82 ms and 173.16 ms respectively without control. However, the absolute mean deviations from the target response time for DB service and web service are 2.63 ms and 23.79 ms respectively with control. Without control, the mean deviation percentage for DB service and web service is \((9.82/35+23.79/750)/2=5.34\%\) with control.

In the last of experiments, we measure the deviation of user experience from desired targets using our system compared with not using, when the resource competition occurs. We design and implement a program that consumes CPU resource. And the user can set the parameter that how many percentages it consumes. In our experiment, we give this program 7 CPUs and consume 100 percentages in each CPU.

Figure 13 shows that the deviation with control is higher than without control for web service. This is because web service has lower priority compared with DB service. When the resource competition occurs, we firstly ensure the requirements of DB service. This leads to performance loss for web service.

Figure 14 shows that the maximum deviation percentage for web service is 49.73% without control. And the maximum deviation percentage is 40.27% with control.

Figure 15 shows that the deviation with control is much lower than without control for DB service. We give higher priority for DB service. When the resource competition occurs, we give priority to DB service. The SLA of DB service is satisfied at the expense of performance loss for web service.

Figure 16 shows that the maximum deviation percentage for DB service is 65.71% without control. However, the
maximum deviation percentage is 22.86% with control in the stabilization process.

In Table 2, the absolute mean deviations from the target response time for DB service and web service are 12.76 ms and 90.47 ms respectively without control. However, the absolute mean deviations from the target response time for DB service and web service are 3.00 ms and 163.11 ms respectively with control. Without control, the mean deviation percentage for DB service and web service is 

\[
\frac{12.76}{35+90.47/750} = 24.26% 
\]

However, the mean deviation percentage for DB service and web service is

\[
\frac{3.00+163.11}{750}/2 = 15.16% 
\]

with control.

In our experiment, we test our system using only two virtual machines. However, our system can work when multiple virtual machines are hosted on a physical machine. Our system is implemented with multi-thread manner. It maintains a descending order list about virtual machines that are running. The list is sorted according to the priority of the service. Even though a new virtual machine is created, our system still can face to this phenomenon. Because a thread examines results of command “xm list”. If it finds that a new virtual machine is created, information of the new virtual machine will be added to the order list of virtual machines. We use a thread to statistics the actual response time for each service. A new thread is created and uses to statistics the actual response time for the new virtual machine. In each resource allocation interval, the resource request thread will scan the order list from begin to end. So our system has better scalability. It can be extended to multiple virtual machines.

6. Conclusions and Future Works

This paper designs a system architecture for resource allocation in virtualized servers according to real time users’ experience. The way we acquire information has low cost compared to [26]. We consider the constraints of limited total resources. When resource competition occurs, we present an optimization model to provide service differentiation. We have confirmed the superiority through experiment. It can reduce the mean deviation of the response time from desired response time.

Acknowledgments

This study is sponsored by the Hi-tech Research and Development Program of China (863 Program) under Grant No. 2011AA01A205, the National Natural Science Foundation of China under Grant No. 60973008, the Doctoral Fund of Ministry of Education of China under Grant No. 20101102110018, Beijing Natural Science Foundation (4122042) and the fund of the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2012ZX-06.

References


Jiajun Liu was born in 1986. He received master degree from Beihang University in 2012. His main research areas are resource management, virtualization, cloud computing and networking.

Xiaolan Tang was born in 1987, China. She is now a Ph.D. candidate in Beihang University. Her current research interest includes wireless sensor networks and delay tolerant networks.

Yiduo Mei received the Ph.D. degree in Computer Science and Technology from Xi’an Jiaotong University 2011. His main research interests include cloud computing, grid computing, system virtualization and performance optimization. He is currently a postdoc in the China Center for Industrial Security Research, where he works on systems and applications security and related research.

Ying Song was born in 1979. She is an assistant professor at Institute of Computing Technology Chinese Academy of Sciences. She received a Ph.D. degree from Institute of Computing Technology Chinese Academy of Sciences. Her main research areas are resource management, virtualization and cloud computing.

Yuzhong Sun was born in 1968. He is a professor at Institute of Computing Technology Chinese Academy of Sciences. He received a Ph.D. degree from Institute of Computing Technology Chinese Academy of Sciences. His main research are computer system software, virtualization and cloud computing.