Cluster Comput (2018) 21:1395–1410
https://doi.org/10.1007/s10586-017-1344-z

An accurate resource scheduling system for virtual machines based on CPU load monitoring and assessment

Ying Li[1] · **Jing Zhang**[2] · **XiaoJun Chen**[2] · **JunHuai Li**[2] · **JuLan Ding**[2]

Received: 18 May 2015 / Revised: 12 August 2017 / Accepted: 6 November 2017 / Published online: 13 November 2017

Abstract An accurate resource scheduling system (RSS) for virtual machines based on CPU monitoring and load assessment is presented to solve the shortcoming of resource scheduling in cloud computing systems. A new architecture is designed to improve Credit scheduler, including three core components: CPU load monitoring component (CLMA), CPU load assessment component (CLAA), and the resource adjustment component (RSA). On the basis of the prototype design, we make an evaluation between Credit scheduler and our system with a typical example in Xen platform. The experimental results show that the proposed system could satisfy the personalized resources requirements from users with higher tasks resource utilization and lower system resource utilization when compared with Credit scheduler. RSS has a strong sensitivity to meet the requirements of cloud computing systems, since it can accelerate the executions of applications via dynamic resource scheduling.

Keywords Cloud computing · Virtualization · Load monitoring · Load assessment · Resource adjustment · Resource scheduling

✉ Ying Li
gniyil_xaut@126.com

Jing Zhang
zhangjing@xaut.edu.cn

[1] School of Automation and Information Engineering, Xi'an University of Technology, Xi'an, China

[2] School of Computer Science and Engineering, Xi'an University of Technology, Xi'an, China

1 Introduction

As a new computing mode, cloud computing provides resources as services to users via network. Infrastructure service is the most basic service in cloud computing systems, but it is urgent to solve the problem that how to use the increasingly powerful resources to satisfy the users' flexible application requirements in cloud computing systems [1]. The advent of system virtualization provides a new way of solving the problem. The virtual machine technology, satisfying the dynamic hardware resource allocation, is currently widely used because of the advantages in flexibility and efficiency. The influential products in virtual machine include VMware, Hyper-V, Xen and so on. Xen, a virtual machine monitor (VMM) developed by Systems Research Group of Computer Laboratory in the University of Cambridge, has been employed in many cloud computing systems, such as Amazon Elastic Computing Cloud, AbiCloud, Eucalyptus, Enomaly's Elastic Computing Platform, etc. The Multitasking, as the key characteristic of Xen, is implemented as multiple guestOS to share resources in a host, which makes Xen allocate host's CPU time slices to guestOS in turn. Resource scheduling refers to the scheduling of VCPUs over virtual machines in cloud computing systems [2]. The CPU scheduling in virtualized environment is divided into three levels: (1) VMM allocates the CPU time slices to VCPUs; (2) guestOS allocates VCPU time slices to processes; and (3) the process allocates the time slices to threads. The CPU scheduling algorithms based on virtual machines directly determines the performance of Xen virtualized environment with the goal of equity and efficiency [3]. Xen employs BVI, SEDF, Credit and some others as the CPU scheduling algorithms, among which, Credit scheduler performs the best on SMP host because of its support for load balancing of SMP and accurate allocation of VCPUs [4]. In

consideration of this, it is set as the default scheduler in Xen hypervisor.

Currently, methods for scheduling resources accurately in virtual computing systems are still few. However, the accurate resources scheduling is the key to satisfy the personalized resource requirement of users in cloud computing systems [5]. The accurate resources scheduling is required to monitor CPU load over virtual machines dynamically and determine the satisfaction of resources periodically. The aim of CPU load monitoring is to determine a reasonable calculation of CPU utilization, and then create a scientific monitoring mechanism to collect and record resource utilization. The collected and recorded data provide an implication for making strategies of scheduling virtual machines. The CPU load assessment aims to evaluate the rationality of CPU load monitoring results, in order to make further judgement if the resource utilization could satisfy the requirements of tasks [6]. The results of the CPU load assessment would be taken as the evidence to complete CPU scheduling (that VMM takes to virtual machines). In current research results, there is less related works on load monitoring and assessment methodologies for applications, and the calculation of resource utilization tends to be limited to the entire systems. Therefore, the resource scheduling methodologies based on the results of monitoring and assessment are less [7]. The contribution of this paper is to solve such problems in Xen hypervisor. In a Para-virtualization environment, the shortcoming of Credit is the static weights based on the priority of guestOS, we improve Credit architecture via computing the dynamic weights of guestOS from the feedback of their loads, and the improved Credit scheduler can satisfy the personalized requirements of tasks better. Then, the implementation of the accurate resource scheduling system goes from the following three ways: (1) to perform the statistics of resource utilization to complete CPU load monitoring, (2) to perform the CPU load assessment to decide the satisfaction of resource utilization, and (3) to perform the CPU scheduling to complete the scheduling of guestOS. We developed a prototype to improve the Credit CPU scheduler and rebuild the treatment details of Xen API functions in this prototype to implement an accurate resource scheduling system.

2 Related work

Xen hypervisor, as a high performance VMM, is taken as the infrastructure virtualization environment by some cloud computing systems. There are several mainstream Xen CPU scheduling algorithms: (1) BVT algorithm [8]: BVT sets a weight to each domains in systems to allocate the time slices of CPU in their proportion, and it is applied to the occasion oriented real-time demand, (2) SEDF algorithm [9]: SEDF also allocates the time slices of CPU in their proportion, but a domain can't occupy all the CPU resources in a time. We can reserve a part of them for the services in other domains, and it is applied to the occasion with the demand of real-time, (3) Credit algorithm [10]: Credit is designed for SMP hosts system, and each CPU of them is a local queue of VCPU. Each VCPU in this queue has two priorities: over or under. In addition to these mainstream CPU scheduling algorithms, some scheduling algorithms were also presented. For example, literature [11] presented a scheduling methodology based on the priority choosing the virtual machines to run in accordance with their I/O status. Literature [12] discussed the I/O performance of a domain scheduling algorithm in Xen hypervisor with more stress on resources exchanging in VMM. In the studies of resources allocation to multiple virtual machines, literature [13] presented that the traditional CPU scheduling algorithms can allocate resources to processes with fairness, but in virtualization environment, the scheduler should adopt other flexible scheduling policies, of which, it is a excellent policy for guestOS to avoid preemptive blocking [14]. Literature [15] evaluated a CPU scheduling algorithm in Xen by analyzing the effects of parameter settings. These methodologies have two characteristics: on the one hand, they stress on the effect that the I/O has on the real time scheduling. On the other hand, the open-source virtualization software is taken to perform the experiments to test I/O overheads and CPU multithreading abilities in file server, web server, and high performance computing server. Beside those studies, dynamic configurations of CPUs are a vital research area, and in this area, dynamic configurations policies oriented the virtual machines status [16] and applied requirements [17] were presented. The CPU scheduling includes the determination of resources requirements, the setting of interrupt cycle, and the selection of diversity scheduling algorithms [18].

The virtual machines monitors are shortage of the knowledge in virtual machines, so unexpected assignments make it more difficult to allocate the resources accurately. Some researchers presented a scheduling algorithm with sense perception [19]. The virtual machines scheduling mechanism with sense perception was used in reasoning about knowledge to infer the I/O roundedness of user-level tasks combined with the event in I/O binding tasks. Literature [20] designed a scheduling algorithm based on the priority of tasks to ensure the fairness of CPU oriented dynamic requirements from applications in SMP hosts. Literature [21] proposed that a modified VMM can perceive an implicit guestOS by inferring the information of guestOS. Literature [22] presented a CPU scheduling algorithm for communication perception with a better cost and performance. Some researches have verified the feasibility of the virtualization in high performance computing in clusters. Literature [23] proposed an algorithm for virtual machine placement in clusters, providing the function of partition in hardware and an extended

operation system, which can effectively transform a physical cluster into a virtual cluster. Literature [24] presented a resource allocation criterion in a virtual cluster including a scheduling algorithm with several continuous operations.

The shortcomings of related work in CPU scheduling based on virtual machines are as follows:

(1) The current established methods to monitor the resource utilization for application are less effective because of coarse-grained calculations. For example, in literature [25–27], the CPU utilization is usually determined by collecting CPU working time. Given *total* as a period of time and *idle* as the running time of idle tasks, the CPU utilization is *(total-idle)/total* . It is a very simple method, but we can only determine the entire system resource expenditure and suppose that the CPU is working in full load. Despite the system shows that tasks have higher resource utilization, in the use of resources, most of them are not the effective utilization of application. Even most of their working time is in system status, such as communication, synchronization and switching, not the executing time of tasks [28]. It is concluded that the load of applications should be computed by using the real instruction execution time. Therefore, it is necessary to calculate the applications' effective resource utilization in a more fine-grained way, and make an accurate resources scheduling based on these results.

(2) The system usually sets the priority and weight to virtual machines according to Service-Level Agreement (SLA), which decides the resource allocation rate, so the resource allocation rate would keep unchanged once they were set to tasks at first [10]. There are currently some methods to maintain Quality of Service (QoS) for assessing the satisfaction of resource utilization of applications during system operation [29–31]. The SLA is usually determined by the estimated resource requirements. The application may be affected by many factors, such as other applications and hardware adjustment, so there are great inconsistencies between QoS and SLA [32]. It is necessary to adjust the resources dynamically and make QoS close to SLA as much as possible. As a result, the resource utilization and billing management would become more reasonable.

(3) Despite some researchers have presented some methodologies to monitor and assess CPU load, but they generally take the minimum overall load, the maximum assignments, the best performance and load balancing as the goals [33,34]. As a result, they attached great importance to system optimization and paid less attention to the satisfaction of individual tasks. Cloud computing systems, stressing on personalized services, would not exchange the overall objectives at the expense of the individuals. So we should make an accurate CPU scheduling for the tasks in virtual machines [35].

An improved Credit scheduler to solve above shortcomings is proposed in this paper. The difficulties of our work are: (1) CPU monitoring and load assessment architecture for resources scheduling involves multiple levels, because its components locate at different CPU-rings with different permissions. For example, the collection of CPU utilization should be in Xen hypervisor, which owns the highest privilege level, but at the same time, the collection of VCPU utilization is in guestOS with subsidiary permissions and the load assessment and storage are designed as an application in the lowest CPU permission [36]. It is very complicated and difficult to switch the CPU-rings in different permissions. Secondly, resource monitoring and load assessment methodologies should reflect the effective resource utilization comprehensively. We solve the first problem by using the Para-virtualization mechanism of Xen hypervisor. The system-call and kernel device drivers provide an approach for the switching of resource manager to guestOS. Meanwhile, the hyper-call and event channel provide a way to exchange the data between guestOS and Xen hypervisor. We reference the present methodologies to solve the second problem and improve their shortcomings, in which, the pre-control chart method of quality management is extended and used in this paper to judge the satisfaction of resource requirement. We complete the resource scheduling by adding a Credit property to the VCPU structure, and then adjust the weights of domains to change the time slices of VCPU based on the results of load assessment.

3 The architecture design

At present, the majority of modern operating systems support parallel processing of multi-task and multi-process, and tasks and processes do not interfere during performing. It is inseparable from the CPU architecture of X86 and X64. X86 CPU architecture provides four privilege levels, which is the mechanism whereby the OS and CPU cooperate to restrict the ability to execute certain machine instructions. These four privilege levels are called rings which are respectively ring0, ring1, ring2 and ring3 [37]. Ring0 is most privileged while ring3 is least privileged. Every privilege level can access the data in itself and less privileged ones. X64 CPU architecture uses only two privilege levels, ring0 and ring3. Ring0 is still most privileged. Xen hypervisor runs in ring0, and applications run in ring3.

Our work involves three components, CPU load monitoring component (CLMA), CPU load assessment component (CLAA), and the resource adjustment component (RSA). These three components are implemented in different levels of Xen hypervisor. Figure 1 is the structural schematic diagram of accurate resource scheduling system, which simply introduces the relationship between the key components and subcomponents. The main subcomponents of CLMA are Resource collector, Resource reporter, Resource Manager

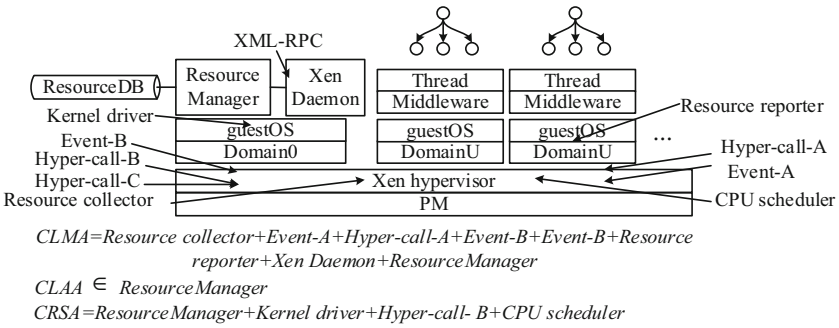


Fig. 1 The components of accurate resource scheduling system

and Xen Daemon. Resource collector, as a tool to compute the load of CPU, VCPU and process, collects resource utilization in Xen hypervisor. It is the core of resource monitoring and resource collection of Xen hypervisor, and it keeps running soon after startup of Xen hypervisor, and helps guestOS on domainUs to collect resource utilization of VCPU and tasks. Resource reporter is a calculator to compute the resource utilization of VCPU and process in guestOS, which is achieved by modifying the operating system kernel. Resource Manager not only manages the static and dynamic information from Xen hypervisor and guestOS, but also stores them into the database, it runs after launches of guestOS. Xen Daemon, is a service process, manages Xen by providing an XML-RPC interface to users. Resource Manager and Xen Daemon execute in guestOS of domain0, and run in ring3. CLMA, as a component of CPU load monitoring, needs to collect the resource utilization of CPU, VCPU and process. However, collecting these information involves multiple privilege levels of Xen. For instance, the resource utilization of CPU needs to be collected in Xen hypervisor which runs in ring0, while that of VCPU and process need to be collected in guestOS which runs in ring1. It is very difficult to exchange information among different privilege levels, and Xen hypervisor cannot directly acquire the resource utilization of VCPU and process. In order to solve the problem, event channels and hypercalls, which are Event-A, Hyper-call-A, Event-B and Hyper-call-B, were added in CLMA. Event channels, working in Xen hypervisor, are asynchronous communication channels between Xen hypervisor and guestOS, and guestOS communicates with Xen hypervisor through hypercalls. Event-A informs the virtual machines periodically to report the VCPU and process utilization on domainUs. Hyper-call-A, a hyper-call that Xen hypervisor provides to guestOS on domainUs, is used to report their resource usage. Event-B is used by Xen hypervisor to periodically notice guestOS on domain0 to save the resource utilization of CPU, VCPUs and processes. Hyper-call-B, a hyper-call that Xen hypervisor provides to guestOS on domain0, is used to collect the dynamic and static information of platform in guestOS of domainU. CLAA algorithm, also implemented in Resource Manager, reads the resource monitoring logs and performs the load assessment. RSA algorithm involves Resource Manager, Kernel driver, Hyper-call-C, CPU scheduler. Resource Manager would make resource adjustment policies according to the results of load assessment. Kernel driver, working in the kernel of guestOS, provides a methodology for Resource Manager working in ring3 to switch to the guestOS Kernel, while Hyper-call-C provides a way to transfer the adjustment results to CPU scheduler in Xen hypervisor working in ring0. CPU scheduler, working in Xen hypervisor, is a part of the CPU scheduling subsystem.

Above 11 subcomponents, included in CLMA, CLAA and RSA components, distributed in Fig. 2a in X86 CPU architecture, and distributed in Fig. 2b in X64 CPU architecture. One more thing to be aware of in Fig. 2 is that Kernel driver works in non-root-ring0 and guestOS runs in non-root-ring3, which is different from that in X86 CPU architecture. It is due to the Para-virtualization mechanism of Xen hypervisor. In a Para-virtualization environment, Xen hypervisor runs in root operation mode, and guestOS runs in non-root operation mode.

To further explain how RAS works, the following is workflow of RSS:

Step 1: Xen hypervisor notices the virtual machines to collect the resource utilization of VCPU and tasks in guestOS of domainUs through Event-A;

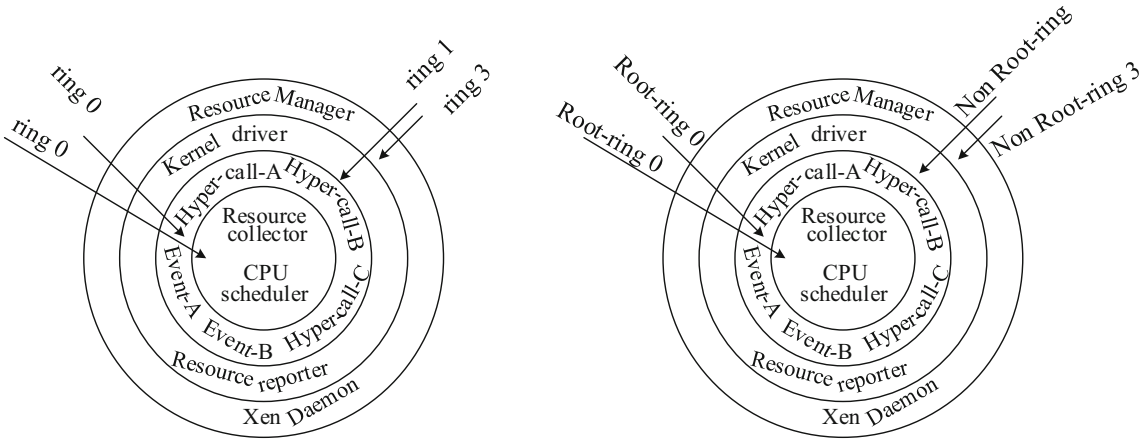
Step 2: Resource reporter computes the resource utilization of VCPUs and tasks in guestOS of domainUs.

Step 3: Xen hypervisor computes the CPU utilization directly, and obtains the resource utilization of VCPUs and tasks computed in Step 2 through Hyper-call-A, then sends them to Resource Manager in guestOS of domain0 through Hyper-call-B.

Step 4: Xen hypervisor notices domain0 to save the resource utilization of CPU, VCPUs and tasks to ResourceDB through Event-B.

Step 5: Resource Manager performs the load assessment. If the assessment result is not satisfied, go to Step 6. Otherwise, go to Step 1 after the cycle is complete.

Step 6: Resource Manager calculates the weights of domainUs for Credit scheduler, and then calls Hyper-call-C to



(a) X86 CPU Architecture **(b)** X64 CPU Architecture

Fig. 2 The architecture design

transfer the adjustment to CPU scheduler in Xen hypervisor.

Step 7: CPU scheduler adjusts the weights of domainUs, go to Step 1 after the cycle is complete.

We will discuss the work principle of above components in detail in next section.

4 The key technologies design

4.1 CPU monitoring based on virtual machines

We implement CPU monitoring in CLMA. Load monitoring based on virtual machines is to determine a calculation approach for CPU utilization. Present methodologies tend to calculate the CPU utilization in systems, which rarely reflect the amount of resources utilized by applications. Hence, we present a way to monitor and calculate CPU utilization of applications. There are a number of indicators about the CPU utilization of applications as follows:

(1) The calculation of average CPU utilization

The timer of Xen hypervisor records the working time of each CPU from power time to t time. The timer of guestOS in virtual machines keeps the records of the working time of each VCPU from power time to t time. The working time is the number of clock ticks for CPU, VCPU or process in running time.

CPU utilization can be determined by the ratio of the quantity of working CPUs and the quantity of all CPUs. But we cannot obtain the instantaneous status of CPU, so we estimate their average utilization in a period. The operation system measures the utilization through the ratio of working time to total time. In normal circumstances, Xen hypervisor records the working time from power time to t time, We take the total working time with Δt seconds from (t − Δt) to t to compute the CPU utilization, the value of ith CPU utilization is:

$$U_i^{CPU}(t) = \frac{T_i(t) - T_i(t - \Delta t)}{\Delta t} \tag{1}$$

where $T_i(t)$ and $T_i(t - \Delta t)$ respectively denotes the total working time of the ith CPU ($0 \leq \mathrm{i} \leq \mathrm{n} - 1$) from power time to t time and that of ith CPU ($0 \leq \mathrm{i} \leq \mathrm{n} - 1$) from power time to $(t - \Delta t)$ time, and $T_i(t) - T_i(t - \Delta t)$ represents the total working time of the ith CPU ($0 \leq \mathrm{i} \leq \mathrm{n} - 1$) during Δt.

$$\begin{aligned} U^{CPU}(t) &= \frac{\sum_{i=0}^{n-1} T_i(t) - T_i(t - \Delta t)}{n\Delta t} \\ &= \frac{\sum_{i=0}^{n-1} T_i(t)}{n\Delta t} - \frac{\sum_{i=0}^{n-1} T_i(t - \Delta t)}{n\Delta t} \end{aligned} \tag{2}$$

where n represents the quantity of CPU. The average CPU utilization is an indicator embodying the performance of virtualization platform.

(2) The calculation of virtual machines resource utilization

Under virtual environment, it is hard to directly monitor the CPU utilization of application. Thus, our research goes the way of monitoring resource utilization that virtual machines take to CPU in Xen hypervisor and resource utilization that applications take to VCPU in guestOS respectively, and multiplying them together to get the CPU utilization of application.

The CPU utilization of the lth VCPU in the jth virtual machine is:

$$U_{VCPU_{lj}}{}^{CPU}(t) = \frac{T_{lj}(t) - T_{lj}(t - \Delta t)}{n\Delta t} \tag{3}$$

where $T_{lj}(t)$ denotes the total working time of the lth VCPU ($0 \leq l \leq k(t)-1$) in jth virtual machine from power time to t time, while $T_{lj}(t-\Delta t)$ represents the total working time of lth VCPU ($0 \leq l \leq k(t)-1$) in jth virtual machine ($0 \leq j \leq m-1$) from power time to $(t-\Delta t)$ time.

Let the quantity of VCPU in jth virtual machine is k in t time (k would change with the time, so $k \sim k(t)$). Then the CPU utilization of the jth virtual machine is:

$$U_j^{CPU}(t) = \frac{\sum_{l=0}^{k(t)-1} T_{lj}(t) - \sum_{l=0}^{k(t-\Delta t)-1} T_{lj}(t-\Delta t)}{n\Delta t} \tag{4}$$

Resource utilization that virtual machines take to CPU is an indicator embodying the performance of all guestOS in virtualized platform.

(3) The calculation of application resource utilization

Assuming there is only one application task$_j$ and several service processes run in the jth virtual machine. At t time, let the total working time that a task occupies the lth VCPU in virtual machine be $V_l(t)$, thus, the VCPU utilization that task$_j$ takes to VCPU is:

$$U_{task_j}{}^{VCPU}(t) = \frac{\sum_{l=0}^{k(t)-1} V_l(t) - \sum_{l=0}^{k(t-\Delta t)-1} V_l(t-\Delta t)}{\sum_{l=0}^{k(t)-1} T_{lj}(t) - \sum_{l=0}^{k(t-\Delta t)-1} T_{lj}(t-\Delta t)} \tag{5}$$

In the formula above, the numerator is the total working time that task$_j$ occupies the lth VCPU in virtual machine during Δt time, and the denominator is the total working time of the lth VCPU ($0 \leq l \leq k(t)-1$) in jth virtual machine during Δt time.

(4) The calculation of application CPU utilization

Based on the above analysis, we can obtain the CPU utilization of task$_j$ is:

$$U_{task_j}{}^{CPU}(t) = U_{task_j}{}^{VCPU}(t) \cdot U_j^{CPU}(t) \tag{6}$$

The *CPU* utilization of task$_j$ is an indicator embodying the performance of application in virtualized platform.

(5) The calculation of application resource utilization with domain0 resource expenditure

There are usually several virtual machines running in Xen, which are called as domain. Domain is divided into domain0 and domainU. Domain0, a special virtual machine, manages domainUs. I/O drivers that domainUs require are installed in domain0. The front-back drivers are created by Xen to provide the input and output services for domainUs. The resource expenditure that domain0 should be taken as the resource expenditure of related domainUs. Let the task in domain0 be an idle process, unrelated to other domainUs, then, the resource expenditure of Domain0 kernel is:

$$S_{domain0}(t) = \sum_{l=0}^{k(t)-1} [R_l(t) + W_l(t)] - \sum_{l=0}^{k(t-\Delta t)-1} [R_l(t-\Delta t) + W_l(t-\Delta t)] \tag{7}$$

where $R_l(t)$ represents the total working time that guestOS kernel occupies the *lth* VCPU in the jth virtual machine from power time to t time, and $W_l(t)$ represents the total working time that the daemon process occupies the lth VCPU in the jth virtual machine. In our study, the virtual machine $j=0$ is called as domain0, then, $S_{domain0}$ is attributed to other virtual machines respectively. The application resource utilization with Domain0 resource expenditure is:

$$U'_{task_j}{}^{CPU}(t) = U_{task_j}{}^{CPU}(t) + \frac{U_j{}^{CPU}(t)}{\sum_{j=1}^{m-1} U_j{}^{CPU}(t)} \cdot \frac{S_{domain0}(t)}{n_\Delta t} \tag{8}$$

As you can see from Formula 8, the domain0 resource expenditure was divided according the CPU utilization of related virtual machines and taken as the resource expenditure of tasks running in the virtual machines. The application resource utilization with domain0 resource expenditure is the subscription basis for users in systems.

(6) The calculation of working efficiency of the task

We take the ratio of the application CPU utilization to the application CPU utilization with domain0 expenditure as the working efficiency of the task:

$$r_j(t) = \frac{U_{task_j}{}^{CPU}(t)}{U'_{task_j}{}^{CPU}(t)} \tag{9}$$

The working efficiency of the task is an indicator embodying the matching degree between the application and the architecture of virtualized platform.

$U'_{task_j}{}^{CPU}(t)$, the most concerned indicator from the view of users, can embody the personalized resource requirement of the QoS, so we make the CPU load assessment with it in next section.

(5) CPU monitoring flows

The CPU scheduling subsystem of Xen hypervisor collects $T_i(t)$ for CPUs periodically in a time interval Δt. The kernels of guestOS in domain0 and domainUs transfer the total working time of $T_{ij}(t)$ for VCPU via the hypercall in Xen hypervisor. The CPU load monitoring flows are described as follows:

Step 1: Resource collector in the kernel of Xen hypervisor obtains the static resource information: CPU, domain, VCPU and process.

Step 2: Resource collector obtains the current time, and then initializes the timer for CPU, domain, VCPU and process.

Step 3: Wait a period of time Δt, Resource collector collects $T_i(t)$ from CPU.

Step 4: The event channel Event-A notices the guestOS kernel in domain0 and domainUs to collect $T_{ij}(t)$, $V_l(t)$, $R_l(t)$, $W_l(t)$ for all VCPU, and then, domain0 and domainUs transfer the$T_{ij}(t)$, $V_l(t)$, $R_l(t)$, $W_l(t)$ to Xen hypervisor by calling Hyper-call–A.

Step 5: Compute $U^{CPU}(t)$, ${U_j}^{CPU}(t)$, ${U_{task_j}}^{CPU}(t)$, ${U'_{task_j}}^{CPU}(t)$ and $r_j(t)$.

Step 6: The event channel Event-B transfers $U^{CPU}(t)$, ${U_j}^{CPU}(t)$, ${U_{task_j}}^{CPU}(t)$, ${U'_{task_j}}^{CPU}(t)$ and $r_j(t)$ to the kernel of domain0. Xen Daemon would wake Resource Manager to call the Kernal Drivers, and then execute hyper-call Hyper-call-B. Finally, the data would be received and stored into ResourceDB.

Step 7: Go to Step 3.

4.2 CPU assessment based on load stability and capability

We implement CPU assessment in CLAA. Present researches about CPU scheduling tend to lack of accuracy and effectiveness. Cloud computing systems take the resource utilization as the billing base. If we allocate a fewer resources to virtual machines, the requirement would not be satisfied. But if we allocate too many resources to virtual machines, it would lead to the waste. The accuracy in resource allocation requires us to make a dynamic assessment to virtual machines based on CPU load. So, it is important to figure out the distribution of tasks' resource utilization. In general, the CPU scheduler limits the resource utilization in a certain scope, and the fluctuation of the resource requirements belongs to normal distribution.

Five indexes of $U^{CPU}(t)$, $U_j^{CPU}(t)$, $U_{task_j}^{CPU}(t)$, ${U'_{task_j}}^{CPU}(t)$ and $r_j(t)$ are the assessment objects of CPU load. In this section, we take ${U'_{task_j}}^{CPU}(t)$ as the study focus, and the other indexes are similar to it. Figure 3 is the frequency diagram of tasks' resource utilization in one of guestOS we collected during the testing, which reveals that most of them have size in a specific range.

As we can see from the above figure, resource utilization of tasks is almost in accord with normal distribution, and it is a very important index in describing SLA for computing task. As CPU utilization belongs to the normal distribution, so SLA can be determined as a specification in the scope with centerline, upper deviation, and lower deviation. We need to

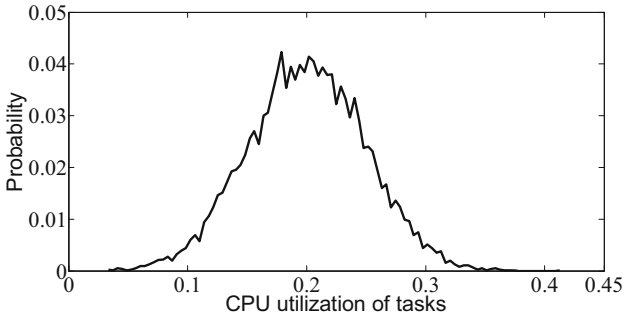


Fig. 3 The frequency diagram of resource utilization

select a methodology to determine whether the resource monitoring results can satisfy SLA. We choose the pre-control chart in quality management to make an assessment to the CPU load. We select the pre-control chart, as it is very similar to the assessment of quality characteristics in small-batch production. Combined with the CPU load monitoring results, we divide it into load stability assessment and load capability assessment.

(1) Basic principle

Based on the basic thought of pre-control chart, we define some important concepts for CPU load assessment.

Load distribution For a task in Xen hypervisor, CLMA collects a series of values of resource utilization that computing task takes to host in a period of time. As previous mentioned, so they are approximately thought as normal distribution:

$$ {U'_{task_j}}^{CPU}(t) \sim (\mu, \sigma) \tag{10} $$

Load specification The scope of CPU load, satisfying users' requirement, is named as load specification. Load specification consists of specification center M, upper limit T_U and lower limit T_L. The specification center is SLA, and upper and lower limit is its upper deviation and lower deviation.

It is known from load distribution and load specification that if the centerline of load distribution is not coincident with the specification center, there must exist a deviation ε ($\varepsilon = |M - \mu| > 0$).

CPU load stability We use the process stability in quality management to perform the assessment combined with load specification, and then, judge whether load distribution is in a stable status in a period of time. CPU load stability would determine whether the systems can provide a continuous and good QoS. If CPU load stability is not in stable status, the system requires to adjust parameters.

CPU load capability We use the process capability index Cp to determine load capability combined with the load specification, and then, to judge whether load distribution is in a capability satisfying status in a period of time. CPU load

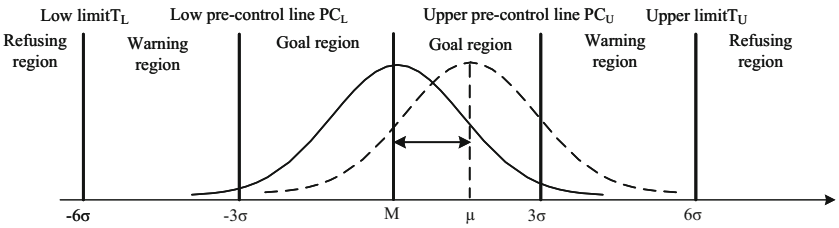


Fig. 4 The limits of pre-control chart

capability would determine whether the systems can provide enough resource to the task. If CPU load cannot satisfy the capability status or CPU load has exceeded the capacity, the system requires to adjust parameters.

The control limits of pre-control chart are shown in Fig. 4. We describe its determination approach as follows: The load specification is divided into four equal sizes. The center line between the specification centerline M and the upper and lower limits $T_U \backslash \mathrm{T}_L$ are defined as the control limits $\mathrm{PC}_L \backslash PC_U$. The goal region is the zone between PC_L and PC_U, the warning regions are the zones between PC_L and T_L, and PC_U and T_U, and refusing regions are the regions outside T_U and T_L [38].

We determine the assumptions for the pre-control charts as two conditions:

(I) Assumption in load distribution coinciding with the load specification is that the load monitoring results belong to normal distribution and the load capability is $C_p \geq 1$ [39].

(II) Considering the probability, the actual load distribution center coinciding with the load specification center is very low, so we would pay more attention to the non-overlapping status. When $\varepsilon > 1.5\sigma$, combined with the features of 6σ whose process capability index is $C_p \geq 2$ [40], the load distribution can be thought as unsatisfied because of its great deviation. We study on $\varepsilon \leq 1.5\sigma$. When the load distribution does not coincide with load specification, we assume that they are in normal distribution and load capability is $C_p \geq 2$.

(2) Load assessment rules

We divide the CPU load assessment into three stages: load stability assessment in initial stage, load capability assessment in initial stage and load stability assessment in running stage. The initial stage means the task loading process. In this time, the QoS that the task takes to resource requirement is higher than later time, so the CPU load should keep consistent with SLA. We are required to perform the CPU load stability assessment and CPU load capability assessment in initial stage. After the task is loaded, we need to perform CPU load stability assessment in running stage.

(1) Rule 1: Load stability assessment in initial stage

In initial stage, CLMA component continuously acquires the CPU monitoring data to make an assessment. But the systems should determine a reasonable sample size to analyze firstly, which is the base of assessment.

Based on the basic assumption $\varepsilon \leq 1.5\sigma$, combined with the process capability index [41,42]:

$$C_{pm} = \frac{C_P}{\sqrt{1 + (\varepsilon/\sigma)^2}} \tag{11}$$

When C_p is 1, 1.33, 1.67, 2 and 2.33 respectively, C_{pm} is 0.5547, 0.7377, 0.9263, 1.1094 and 1.2924. Literature [38] sums up the probability of different samples falling to goal regions under different C_p in load distribution center coinciding with the load specification center, and suggests that when $C_p \geq 1$, the sample size should be 10 in initial stage. When load distribution center does not overlap with specification center and $C_p \geq 2$, we assume that $C_{pm} \geq 1.1094$ would coincide with $C_p \geq 1$, so we decide the sample size 10 [43].

Combined with the setting of pre-control chart [44], we present the rule of pre-control in initial stage. For ten samples from ResourceDB in architecture: (I) If the monitoring results of CPU utilization X fall into the goal regions, the initial stage can satisfy the requirement. (II) If there is one sample in ten falling into the warning regions, the system should read next sample. If next sample is falling into the goal regions, the initial stage can satisfy the requirement. Otherwise, it cannot satisfy the requirement. (III) If there are two samples in ten samples falling into warning regions or one sample falling into refusing regions, the initial stage cannot satisfy the requirement.

(2) Rule 2: Load capability assessment in initial stage

Based on the assumption, when the distribution center does not coincide with the specification center, there are:

$$\varepsilon = \left| \bar{X} - \frac{T_u + T_l}{2} \right| = \left| \frac{1}{n} \sum_{i=1}^{n} x_i - \frac{T_u + T_l}{2} \right| \tag{12}$$

$$\sigma \approx \mathrm{S} = \sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}} \frac{(x_i - \bar{X})^2}{\mathrm{n} - 1}} \tag{13}$$

The relationship of process capability index and the *Taguchi* capability index is:

$$C_{pm} = \frac{T}{6S\sqrt{1+(\varepsilon/S)^2}} = \frac{C_P}{\sqrt{1+(\varepsilon/S)^2}} \tag{14}$$

Based on the basic assumption, when the CPU load satisfying 6σ, $\sigma < \mathrm{T}/12$, $\varepsilon < 1.5\sigma$ and load capability index $C_p > 2$, we get the *Taguchi* capability index $C_{pm} > 1.667$. When the condition of load distribution center does not overlap with the load specification, we take $C_{pm} > 1.667$ as the load capability assessment base.

(3) Rule 3: Load stability assessment in running stage

When the task has been loaded or entered into the running state, the assessment in stable status should be different from that in initial stage. The system should determine a reasonable sample size to make an assessment.

Literature [45] suggests the sample size to be 3 in $C_p \geq 1$ under the condition of load distribution center coinciding with the load specification. When the condition of load distribution center does not overlap with the load specification and $\varepsilon \leq 1.5\sigma$, we also suggest sample size to be 3in$C_p \geq 2$ and $C_{pm} \geq 1.1094$.

Based on the basic assumption, let *CPU* utilization of a task be $F(X)$ in Fig. 3. As X has randomness, and it belongs to normal distribution ($T = T_U - T_L, \sigma \leq T/12$), the upper bound is M$+6\sigma$, the lower bound is M$-6\sigma$, and the deviation is $\varepsilon \leq 1.5\sigma$. Let the probability of X falling into goal regions be P_g, and the probability of X falling into 2 warning regions be P_{yl} and P_{yu}, and the probability of X falling into two refusing regions be P_{rl} andP_{ru} then:

$$\begin{aligned}
P_g &\geq P\{-4.5\sigma \leq x \leq 1.5\sigma\} \\
&= P\left\{\frac{-4.5\sigma-\mu}{\sigma} \leq \frac{x-\mu}{\sigma} \leq \frac{1.5\sigma-\mu}{\sigma}\right\} \\
&= P\left\{-\frac{1/4T+\varepsilon}{\sigma} \leq \frac{x-\mu}{\sigma} \leq \frac{1/4T-\varepsilon}{\sigma}\right\} \\
&= P\left\{-4.5 \leq \frac{x-\mu}{\sigma} \leq 1.5\right\} \\
&= \Phi(1.5) - \Phi(-4.5) = 0.9331866
\end{aligned} \tag{15}$$

$$\begin{aligned}
P_y &\leq P_{yl} + P_{yu} = P\{-7.5\sigma \leq x < -4.5\sigma\} \\
&\quad + P\{1.5\sigma < x \leq 4.5\sigma\} \\
&= P\left\{\frac{-7.5\sigma-\mu}{\sigma} \leq \frac{x-\mu}{\sigma} < \frac{-4.5\sigma-\mu}{\sigma}\right\} \\
&\quad + P\left\{\frac{1.5\sigma-\mu}{\sigma} < \frac{x-\mu}{\sigma} \leq \frac{4.5\sigma-\mu}{\sigma}\right\} \\
&= P\left\{-\frac{1/2T+\varepsilon}{\sigma} \leq \frac{x-\mu}{\sigma} < -\frac{1/4T+\varepsilon}{\sigma}\right\} \\
&\quad + P\left\{\frac{1/4T-\varepsilon}{\sigma} < \frac{x-\mu}{\sigma} \leq \frac{1/2T-\varepsilon}{\sigma}\right\} \\
&= [\Phi(-4.5) - \Phi(-7.5)] + [\Phi(4.5) - \Phi(1.5)] \\
&= 0.0000034 + 0.0668066 = 0.06681
\end{aligned} \tag{16}$$

$$\begin{aligned}
P_r &\leq P_{rl} + P_{ru} \\
&= P\{x < -7.5\sigma\} + P\{4.5\sigma < x\} \\
&= P\left\{\frac{x-\mu}{\sigma} < \frac{-7.5\sigma-\mu}{\sigma}\right\} \\
&\quad + P\left\{\frac{4.5\sigma-\mu}{\sigma} < \frac{x-\mu}{\sigma}\right\} \\
&= P\left\{\frac{x-\mu}{\sigma} < -\frac{1/2T+\varepsilon}{\sigma}\right\} \\
&\quad + P\left\{\frac{1/2T-\varepsilon}{\sigma} < \frac{x-\mu}{\sigma}\right\} \\
&= \Phi(-7.5) + [1 - \Phi(4.5)] \\
&= 0 + 0.0000034 = 0.0000034
\end{aligned} \tag{17}$$

In a period of time interval, the system reads three samples from ResourceDB, and there are $5^3 = 125$ statuses because these three samples may fall into the goal region, warning regions and refusing regions [46]. We sum up the probability of three samples falling into goal region, and 10 combinations are shown in Table 1.

The principle of pre-control chart is the same as control-chart, and when a small probability event (less than 0.01) happens, the process is considered as abnormal load. It is seen from Table. 1 that the system continuously reads three samples to make this assessment. (I) If the probability of three samples falling into the goal regions is greater than 0.01,the load is judged as normal. (II) If $\varepsilon \geq 0.6815\sigma$, or the probability of a sample falling into right regions of warning area and two others falling into goal regions is greater than 0.01 (not belongs to small probability event), the load is judged as normal. But if a sample falls into left regions of warning area and two others fall into goal regions is less than 0.01 (small probability event), the load is judged as abnormal. (III) If $\varepsilon < 0.6815\sigma$, the probability that a sample falls into right regions of warning area and two others fall into goal regions is less than 0.01 (small probability event), and the load is judged as abnormal. (IV) If a sample falls into goal regions and two others fall into warning regions, the probability is less than 0.01, and the load is judged as abnormal. (V) If three samples fall into refusing regions, the load is judged as abnormal.

(3) Load assessment steps

Domain0 in Xen hypervisor runs a Daemon process from power time, which collects and stores the CPU monitoring results. Resource manager reads these data and performs the assessment. The assessment results can be divided into four unsatisfied marks *A, B, C, D* and a satisfied mark *S*, load assessment steps are described as follows:

Step 1: Compute the deviation ε. If $\varepsilon \leq 1.5\sigma$, the algorithm return the unsatisfied marks A, which means that the load distribution is far from the specification.

Table 1 Probability for three samples falling into different regions

Regions			Probability		
Anti-side regions	Goal regions	Ipsilateral regions	$\varepsilon = 1.5\sigma$	$\varepsilon = 0.6815\sigma$	$\varepsilon = 0$
1	1	1	$2.1 \times 10^{-7} < 0.01$	$1.2 \times 10^{-6} < 0.01$	$1.8 \times 10^{-6} < 0.01$
1	2		$3 \times 10^{-6} < 0.01$	$0.000114 < 0.01$	$0.001343 < 0.01$
1		2	$1.5 \times 10^{-8} < 0.01$	$1.2 \times 10^{-8} < 0.01$	$2.5 \times 10^{-9} < 0.01$
2	1		$1.1 \times 10^{-11} < 0.01$	$1.3 \times 10^{-8} < 0.01$	$1.8 \times 10^{-6} < 0.01$
	1	2	$0.004165 < 0.01$	$0.000102 < 0.01$	$1.8 \times 10^{-6} < 0.01$
2		1	$7.7 \times 10^{-13} < 0.01$	$1.4 \times 10^{-10} < 0.01$	$2.5 \times 10^{-9} < 0.01$
	2	1	$0.933^2 \times 0.067 = 0.058178$	0.01	$0.001343 < 0.01$
	3		$0.933^3 = 0.812654$	$0.9897^3 = 0.969457$	$0.9973^3 = 0.991922$
3			$3.9 \times 10^{-17} < 0.01$	$1.6 \times 10^{-12} < 0.01$	$2.5 \times 10^{-9} < 0.01$
		3	$0.000298 < 0.01$	$1.1 \times 10^{-6} < 0.01$	$2.5 \times 10^{-9} < 0.01$

Step 2: Determine the stage: initial stage or running stage. Go to Step 7 if it is in running stage.

Step 3: Wait for a period of time, read 11 load monitoring data ${U'_{task_j}}^{CPU}(t)$ of current cloud computing task from ResourceDB and take them as the data source of assessment.

Step 4: Perform load stability assessment in initial stage based on rule 1. If the result is unstable, return the unsatisfied mark B, which means that the load distribution is unstable in initial stage.

Step 5: Perform load capability assessment in initial stage based on rule 2. If the result is shortage of capability or exceeds the capacity, then returns the unsatisfied mark C, which means that the load distribution is incapacity in initial stage.

Step 6: Go to Step 1.

Step 7: Wait for a period of time, and read 3 load monitoring data ${U'_{task_j}}^{CPU}(t)$ from ResourceDB and take them as the sources of assessment.

Step 8: Load stability assessment in running stage based on rule 3. Return the unsatisfied mark D if the result is unstable, which means that the load distribution is unstable in running stage.

Step 9: Return the satisfied mark S, and then go to Step 7.

4.3 Resource adjustment based on CPU assessment

We implement CPU assessment in RSA. Credit is a mainstream scheduler in Xen hypervisor, designed for SMP host systems and suitable for cloud computing systems. But the shortcoming of Credit is that Xen determines the static weights according to the priority of guestOS. We improve the Credit to be a dynamic adjustment algorithm via the changing of weights, in which, the time slices of all VCPUs in every cycle are determined dynamically.

(1) The calculation of time slice and the weights

We determine the relationship between VCPU time slices and CPU utilization in this section. In Credit scheduler, the Credit value is created in domain, but in RSS, we set Credit into VCPU. Although domain has several VCPU, they also have their difference in scheduling. The Credit of $k(t)$ VCPU is called as $Credit_{ij}$ in each virtual machine. Let Xen hypervisor run m virtual machines and their weights be $Weight_j$ $(0 \le \mathrm{j} \le \mathrm{m}-1)$, and let the time slice in a fixed executing cycle H be $H_{lj}(0 \le \mathrm{j} \le \mathrm{m}-1, 0 \le \mathrm{l} \le \mathrm{k(t)}-1)$, then the relationship between $Weight_j$ and $Credit_{lj}$ is:

$$Weight_j = \sum_{l=0}^{k(t)-1} Credit_{lj} \tag{18}$$

The relationship among H, H_{lj} and $Credit_{lj}$ is:

$$H_{lj} = H \cdot \frac{Credit_{lj}}{\sum_{l=0}^{k(t)-1} Credit_{lj}} \tag{19}$$

The relationship between ${U_{VCPU_{lj}}}^{CPU}(t)$ and H_{lj} is:

$${U_{VCPU_{lj}}}^{CPU}(t) = \frac{H_{lj}}{\sum_{l=0}^{k(t)-1} H_{lj}} \tag{20}$$

Based on the relationship among $U_{task_j}^{CPU}(t)$, ${U'_{task_j}}^{CPU}(t)$, $U_j^{CPU}(t)$ *and* ${U_{VCPU_{lj}}}^{CPU}(t)$ mentioned above, we can necessarily compute $weight_j$ based on ${U'_{task_j}}^{CPU}(t)$ with the function $f : weight_j = f({U'_{task_j}}^{CPU}(t))$.

(2) Resource adjustment flows

If the load assessment is unsatisfied, Resource Manager analyzes the ${U'_{task_j}}^{CPU}(t)$ distribution based on the unsatisfied marks and decides resource adjustment policies, and then, notices the Xen Daemon to transfer the policies. Xen Daemon posts requests to guestOS, and then it is received by Xen hypervisor through hyper-call. CPU scheduler in Xen hypervisor changes the weights of domains, and then change the Credit of VCPU. Therefore, the resource utilization would be adjusted in next executing cycle. Resource adjustment flows are described as follows:

Step 1: Resource Manager decides the reasons of abnormal data based on the unsatisfied marks.
Step 2: Determine the abnormal ${U'_{task_j}}^{CPU}(t)$.
Step 3: Determine the adjustment of ${U'_{task_j}}^{CPU}(t)$.
Step 4: Compute the weights for domains.
Step 5: Resource Manager notices the Xen Daemon and Kernal driver, and then calls Hyper-call-C to transfer the adjustment to CPU scheduler in Xen hypervisor.
Step 6: CPU scheduler adjusts the weights for domains.

5 System evaluations

Combined with the thought of RSS mentioned above, we design their components and implement their functions. Our prototype has completed their key technologies in CPU monitoring, load assessment and resource adjustment. The prototype is based on Xen3.3 and VZlinux2.6.48 for Xen3.3 source codes, and is developed via Eclipse for Linux with C language and GCC compiler in operation system Fedora core 12.0. For a complex parallel program, there is only one the starting point, which is development machine. It is running on the domain0. Similar to a management node, development machine is not responsible for the actual executions of tasks. The testing experiments are launched and initiated from it, and all experimental data are recorded in development machine for general viewing. The development machine is a PC machine with an Intel core2 CPU (2.8 GHZ), 2 GB memory with PAE, and a 160 GB hard disk. After the source codes are developed and compiled, they are placed into a platform to deploy and install. Because of the limit by conditions, we install the prototype in a high performance host. The host has 4 Intel Xeon 1.6 GHz CPU and with 4 GB DDR RAMS. The system adopts Xen 3.3 as the virtualized platform and takes Red Hat Enterprise Linux 5 as the guestOS of virtual machines. The virtual machine is packaged as the virtual appliance to save in Xen domain0.

In this section, the experiments are made to evaluate the prototype via an example. We take the simulation of cold flow impulsive experiment for a car engine (CFIE) as this instance, which is the typical coupling process considering the affecting relations between flow field and structure. Because of the large scale and complicated computation, the simulation would run for a long time. The task has a higher requirement to its stability of SLA, so it can well verify the efficiency of RSS.

We divide CFIE into four subtasks to accelerate the execution of computation. Because of the parallel computing, four domainUs in Xen 3.3 are created, and four applications are implemented with C language for them. During the analysis of four subtasks, we determine their SLA via ${U'_{task_j}}^{CPU}(t)$. SLA1, as the leading task of four tasks, is 25% in the specification of resource utilization. SLA2, as the main computing process, is unstable and present the increasing trend in the specification of resource utilization, but its minimum value is lower than 15%. SLA3 and SLA4, as the auxiliary computing process, are 22% in the specification of resource utilization. The upper and lower deviations of above specification are controlled in 4%.

We run CFIE in Credit scheduler and RSS respectively. The experiments are designed to achieve three goals: (1) study the features of three components: CLMA, CLAA and RSA to verify the feasibility of themselves in input, output and running, (2) compare the differences between the Credit scheduler and RSS in resource utilization, and (3) compare the differences between Credit scheduler and RSS in completion time.

Based on such requirements, after the task has been completed, we present the results as follows:

(1) The analysis of the components CLMA, CLAA and RSA

For the example in RSS, we save the important data for the input, output and running of CLMA, CLAA and RSA. CLMA saves $U^{CPU}(t)$, $U_j^{CPU}(t)$, ${U_{task_j}}^{CPU}(t)$, ${U'_{task_j}}^{CPU}(t)$ and $r_j(t)$ into the ResourceDB by using the CPU monitor. Their average values in different stages are listed in Table 2, in which, $U_j^{CPU}(t)$, $U_{task_j}^{CPU}(t)$, ${U'_{task_j}}^{CPU}(t)$ and $r_j(t)$ are from domain1.

We can see from Table 2 that $U^{CPU}(t)$ are above 98.9%, which shows that the CPUs are in full load. The effective resource utilization that virtual machines take to host is above 90%. The total resource expenditure of Xen kernel and CPU idle is controlled and limited within 10%. The working efficiency of the task is above 60%, which shows that the total time of Xen kernel, guestOS kernel, service process, idle process and CPU idle time in virtualized platform is limited to 40%. The matching degree between the features of application and the architecture is much higher in RSS than that in Credit scheduler. It is concluded from the data in ResourceDB that the CLMA component has implemented the goal of CPU load monitoring.

CLAA made an assessment for ${U'_{task_j}}^{CPU}(t)$ by using the CPU assessment method above. Four unsatisfied marks, i.e.

Table 2 The statistics of resource utilization in CLMA

Stages (%)	$U^{CPU}(t)$ (%)	$U_j{}^{CPU}(t)$ (%)	$U_{task_j}{}^{CPU}(t)$ (%)	$U_{task_j}{}^{CPU}(t)$ (%)	$r_j(t)$ (%)
0–5	98.92	90.89	15.42	25.00	61.68
10–15	98.96	90.47	15.53	25.02	62.07
50–55	99.02	91.25	16.08	23.50	68.43
75–80	99.03	94.36	19.71	26.50	74.38
85–90	99.12	95.23	18.71	25.14	74.54
95–100	99.31	95.56	17.59	23.55	74.69

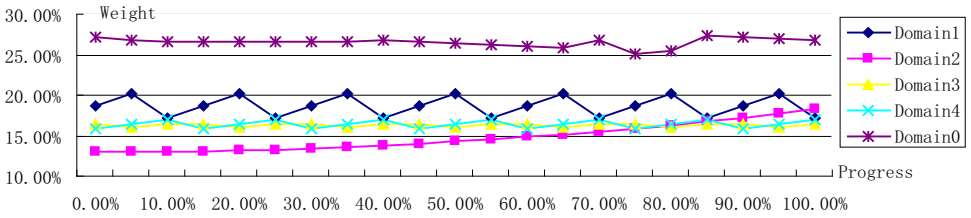


Fig. 5 The trends of *Weights* for five domains under RSA

A, B, C, D, and the satisfied mark S are recorded and discussed below. With the increase of progress, the weight of satisfied mark S presents the trend of increase. In the frontier of task loading, the weight is 48.37%, but in the end of task, the weight is 92.16%. In the beginning of task, the load distribution center is far from the load specification center, so the system requires to make much adjustment in this stage than that in others.

While the system runs for a certain time, and it arrives to relative stable status, the unsatisfied weight would be necessarily decreasing. The weight progress, and the load distribution is much closer to load specification center. In initial stage, the unsatisfied mark B has the greatest weight of all and arrives to 27.15%. The weight of unsatisfied mark C is only 10.53%, which shows that load stability occupies most of them in initial stage. After the task is loaded, the weight of unsatisfied mark D has a key effect on the load balancing. It is concluded that the load assessment, as the premise of resource scheduling, is indispensable.

In the adjustment of RSA component, the curves of trends for the weights of five domains in Xen hypervisor is shown in Fig. 5. In Fig. 5, the weights in domains are ultimately in line with resource demands of tasks running in them, because they are adjusted according to the recorded loads. The weight of domain0 barely changes. The weight of domain1 presents the trend of low fluctuation. The weight of domain2 presents the trend of rising with a small margin. The weights of domain3 and domain4 present the trend of rising with fluctuations. The cyclical fluctuations of weights in domain1, domain3 and domain4 are due to that the resource requirements of tasks are periodical in general. For five domains, domain0 has the maximum weight, and then, the domain1. Because domain0 provides the input, output and management service for other domains. Domain1, as a main computing process coordinating other domains, has a higher weight of time slices.

Generally speaking, in RSS, the prototype can complete the task within a lower period of time than Credit scheduler. Although CLMA, CLAA and RSA make the Credit scheduler more complicated, the fast convergence features also have been embodied fully in our experiments. Furthermore, it is concluded that our algorithms are very stable and robust. As a result, we can implement a higher system performance and a faster task executing in the context of computing center under these components.

(2) CPU utilizations comparison

To observe the relation of CPU utilizations in Credit scheduler as processing of tasks, we put them together in Fig. 6, a cumulative graph. This type of graph can also avoid overlapping curves. In addition to the CPU utilizations of SLA1 $\sim$ SLA4, Fig. 6 also shows that of Xen kernel and idle system. The widths of color bars represent the CPU utilizations of them, respectively. The specific value for they each is calculated by the ratio of their own working time to the total CPU working time, and they are limited in a certain scope according to their weights. The bottom axis is the progress of tasks, in which 0% represents the beginning of tasks and 100% represents that they are completed. In Fig. 6, The CPU utilization under SLA1 presents the trend of load fluctuation. The CPU utilization under SLA2 presents the trend of arithmetic increase. The CPU utilization under SLA3 and SLA4 present the trend of arithmetic decrease. The experiment shows that the average CPU utilizations are 20.00, 17.00, 17.00, 13.00, 18.90 and 14.10% respectively.

In RSS, the relationship of CPU utilizations as processing of tasks is shown in Fig. 7. It is seen from Fig. 7 that the idle expenditure is controlled within 5%. SLA1 presents the trend of fluctuation. SLA2 presents the trend of accelerated increase. SLA3 and SLA4 present the balancing fluctuation with the increase of progress, but it is not changing overall. The CPU utilization of Xen kernel presents the trend of

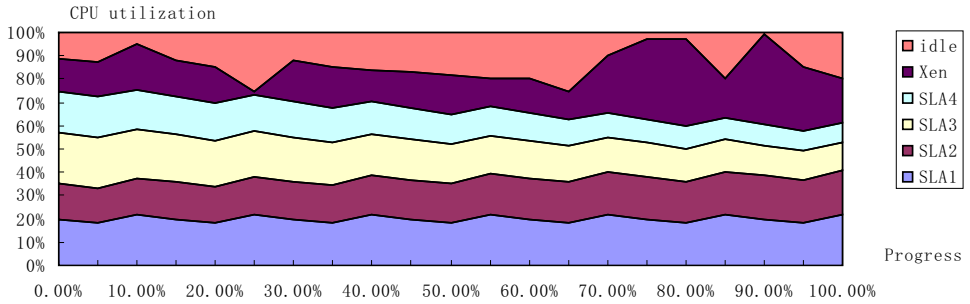


Fig. 6 The relationship of CPU utilization and the progress in Credit scheduler

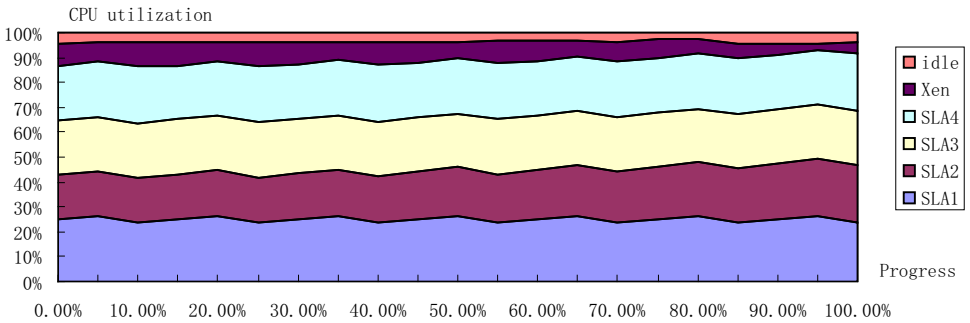


Fig. 7 The relationship of CPU utilization and the progress in RSS

decrease. The experiment data shows that the average CPU utilization are 25.00, 19.83, 21.90, 22.03, 7.43, and 3.81% respectively.

From Figs. 6 and 7, we can observe the relation of the CPU utilizations and the progress of tasks in Credit scheduler and RSS, combining with the specific experimental data, four conclusions can be drawn:

(1) In RSS, the resource utilization occupied by tasks keeps consisting with load specification, and load distribution belongs to normal distribution of SLA. Overall, the four average CPU utilizations (25.00, 19.83, 21.90, and 22.03%) are closer to the load specification (25, 15, 22, and 25%). They meet the requirements of load fluctuation while the upper and lower deviations are within 4%, which is due to the dynamically adjusted weights in RSS. And the average resource utilization in Credit scheduler are 20.00, 17.00, 17.00, and 13.00%, so it did not achieve the requirement of load fluctuation and did not belong to normal distribution. We conclude that Credit scheduler cannot satisfy the personalized resource requirement of tasks, but can RSS.

(2) RSS makes the resource utilization, which represented by the deep purple color bar, very low and stable in Xen kernel. In the experiment, the average value of Xen kernel's resource utilization in RSS is 7.43%, and the maximum value is less than 10%, as shown in Fig. 7. With the increase of task progress, it is lower due to that more stable tasks. However, the Credit scheduler enables high resource utilization in Xen kernel. The average CPU utilization arrives to 18.90%. In Fig. 6, it is obvious that the width of deep purple color bar changes greatly, from the maximum resource utilization 38.40% to the minimum 2.00%. It means the resource utilization in Xen kernel of Credit scheduler is very unstable. We conclude that Credit scheduler cannot satisfy the high resource utilization for tasks, but can RSS.

(3) RSS enables a lower idle resource expenditure, whose average value is only 3.81% and the maximum value is 4.65%, but in Credit, the value arrives to 14.10%, and the maximum value arrives to 25.00% with the great fluctuation. Corresponding to it, the pink color bar in Fig. 6 is much wider than that in Fig. 7 mostly. We conclude that RSS is higher in resource utilization than Credit scheduler in experiments.

(4) RSS has strong sensitivity, and can compute the resource utilization from CPU, VCPU and process accurately all the time. We can see that the resource utilization of RSS represented by color bars in Fig. 7 are more stable and stay within specified range (the deviation is within 4%), while those of Credit scheduler in Fig. 6 are rather fluctuating. The experiment data shows that the value of resource utilizations under SLA1-SLA4 are consisting with load specification. As a result, we conclude that the load assessment makes the resource scheduling more accurate than that in Credit scheduler, which has satisfied the requirement of accurate resource management.

(3) The completion time comparison

For the comparison of performance, the completion time of Credit scheduler and RSS in is shown in Fig 8. In Fig. 8, horizontal axis, the percent of task progress, is set as the marks to track the state of tasks based on the scale of application in development. Vertical axis, the completion time, reflects trend of completion time changing with the increase of progress.

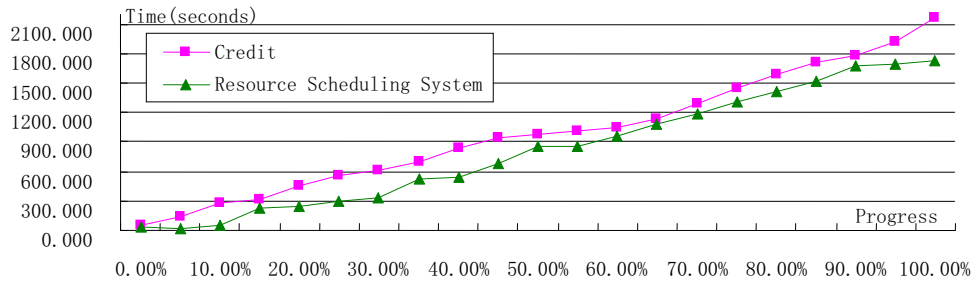


Fig. 8 The completion time comparison in Credit and RSS

It is seen from Fig. 8 that, with the increase of progress, either the Credit or the RSS keeps uniformly increasing in terms of the completion time, but RSS is less time-consuming than Credit either in total completion time or in each stages. The completion time in RSS is at least 200 seconds less than Credit scheduler in experiment. RSS can exchange the completion time with the resource by considering the features of tasks and making a reasonable resource allocation. RSS can decrease idle and kernel resource expenditure, so it is better than Credit in performance to adapt to parallel computing. We conclude that RSS provides the load monitoring and assessment to perform an accurate resource scheduling, so they can accelerate the executing of program with much higher resource utilization in tasks.

6 Conclusions

We present the key technologies of CPU monitoring, load assessment and resource adjustment for virtual machines to solve the shortage of cloud computing systems in accurate resource scheduling. Based on Xen hypervisor, we improve Credit scheduler to present a resource scheduling system (RSS), which is constituted of three core components: CLMA, CLAA and RSA. Our study uses the system-call and kernel driver in Xen hypervisor to exchange the data between resource manager and guestOS. The hyper-call and event channel have also provided an effective approach for the data sharing between guestOS and Xen hypervisor. We present 5 indexes to compute the CPU utilization in virtualized platform. CLMA continuously monitors and records the resource utilization. CLAA references the thought of pre-control chart to propose the basic principle and assessment rules for load assessment based on SLA. We add a property of Credit to VCPU data structure in RSA to adjust the weight values in domains dynamically to change the time slices of VCPUs.

We design a prototype for RSS. And then, a CFIE example is taken to make the experiments to compare the effectiveness and the performance with Credit scheduler. It is concluded from these experiments that RSS can satisfy the personalized resource requirement, because RSS can keep the load distributions which coincide with the SLA. The system has lower resource expenditure in system and higher resource expenditure in tasks with the low idle resource, so it implements the accurate resource management via strong sensitivity. The resource monitoring and assessment have increased the matching degree between application features and architecture. It is also concluded that RSS can accelerate the execution of applications with higher resource utilization.

The shortcoming of our study is that we take multiple virtual machines over a host, rather than a large number of hosts, so we require making the experiments in a large cluster in later work. Furthermore, because of the complex relationships among parallel computing tasks in large–scale cloud computing systems, we should consider the communication and synchronization among tasks based on serial and parallel relations, which would be more complicated to implement the accurate resource scheduling. These problems would be discussed in future work.

References

1. Ian, F., Yong, Zh, Ioan, R., Shiyong, L.: Cloud Computing and Grid Computing 360-degree compared. In: Grid Computing Environments Workshop Gce, vol. 5, pp. 1–10 (2008)
2. Jin, H.: Virtualization technology for computing system. In: High Performance Computing and Communications (2008). https://doi.org/10.1109/HPCC.2008.167
3. Julia, C., Joseph, S.N.: New hardness results for congestion minimization and machine scheduling. J. ACM **53**(5), 707–721 (2006)
4. Dirgo, O., Alan, L.C., Scott, R.: Scheduling I/O in virtual machine monitors. In: International Conference on Virtual Execution Environments, pp. 1–10 (2008). https://doi.org/10.1145/1346256.1346258
5. McDermott, J., Kirby, J., Montrose, B., Johnson, T., Kang, M.: Re-engineering Xen internals for higher-assurance security. Form. Secur. Tech. Rep. **13**(1), 17–24 (2008)
6. Gabor, K., Gabor, T., Peter, K., Zsolt, N.: An approach for virtual appliance distribution for service deployment. Future Gener. Comput. Syst. **27**(3), 280–289 (2011)
7. Mark, S., David, S., Frederic, V., Henri, C.: Resource allocation using virtual clusters. In: The 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 260–267 (2009). https://doi.org/10.1109/CCGRID.2009.23
8. Ian, L., Derek, M., Richard, B., Timothy, R., Paul, B., David, E., Robin, F., Eoin, H.: The design and implementation of an operating system to support distributed multi-media applications. IEEE J. Sel. Areas Commun. **14**(7), 1280–1297 (1996)

9. Kenneth, J.D., David, R.C.: Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. ACM Sigops Oper. Syst. Rev. **34**(2), 27–28 (1999)
10. Ludmila, C., Diwaker, G., Amin, V.: Comparison of the three CPU Schedulers in Xen. Acm Sigmetrics Perform. Eval. Rev. **35**(2), 42–51 (2007)
11. Young, C.L., Albert, Y.Z.: Rescheduling for reliable job completion with the support of clouds. Future Gener. Comput. Syst. **26**(8), 1192–1199 (2010)
12. Kinshuk, G., Dan, T., Yongqiang, H., Mendel, R.: Cellular disco: resource management using virtual clusters on shared-memory multiprocessors. Acm Trans. Comput. Syst. **18**(3), 229–262 (2000)
13. Volkmar, U., Joshua, L., Espen, S., Uwe, D.: Towards scalable multiprocessor virtual machines. In: The 3rd Conference on Virtual Machine Research and Technology Symposium, vol. 3, pp. 1–14 (2004)
14. Mendel, R., Tal, G.: Virtual machine monitors: current technology and future trends. Computer **38**(5), 39–47 (2005)
15. Hiroshi, Y., Kenji, K.: Foxy Technique: tricking operating system policies with a virtual machine monitor. In: The 3rd International Conference on Virtual Execution Environments, pp. 55–64 (2007). https://doi.org/10.1145/1254810.1254818
16. Jonas, P., Christian, S., Claudia, E.: Formal model for virtual machine introspection. ACM Workshop on Virtual Machine Security, pp. 1–10 (2009). https://doi.org/10.1145/1655148.1655150
17. Fumio, M., Dong, S.K., Jong, S.P.: Towards optimal virtual machine placement and rejuvenation scheduling in a virtualized data center. In: IEEE International Conference on Software Reliability Engineering Workshops, vol. 7(5), pp. 1–3 (2008). https://doi.org/10.1109/ISSREW.2008.5355515
18. Dongsung, K., Hwanju, K., Myeongjae, J., Euiseong, S., Joonwon, L.: Guest-aware priority-based virtual machine scheduling for highly consolidated server. In: The 14th International Euro-Par Conference on Parallel Processing, vol. 5168, pp. 285–294 (2008). https://doi.org/10.1007/978-3-540-85451-7_31
19. Cota-Robles, E.C., Flautner, K.: Real-time scheduling of virtual machines. U.S. (2008)
20. Lei, Sh, Deqing, Zh, Hai, J.: Xen Virtualization Technlogy. Huazhong University of Science & Technology Press, Wuhan (2009)
21. Zohar, L., Dimitri, G.G., Baruch, K.: Optimal booking of machines in a virtual job-shop with stochastic processing times to minimize total machine rental and job tardiness costs. Int. J. Prod. Econ. **111**(2), 812–821 (2008)
22. Jinpeng, H., Qin, L., Chunming, H.: Research and design on hypervisor based virtual computing environment. J. Softw. **18**(8), 2016–2026 (2007)
23. Jian, W., Jianling, S., Xinyu, W., Xiaohu, Y., Shenkang, W., Junbo, Ch.: Efficient scheduling algorithm for hard real-time tasks in primary-backup based multiprocessor systems. J. Softw. **20**(10), 2629–2637 (2009)
24. Weizhe, Zh, Zhihong, T., Hongli, Zh, Hui, H., Wenmao, L.: Multi-cluster co-allocation scheduling algorithms in virtual computing environment. J. Softw. **18**(8), 2027–2037 (2007)
25. Sisu, X., Justin, W., Chengyang, L., Christopher G.: Rt-xen: towards real-time hypervisor scheduling in Xen. In: International Conference on Embedded Software, pp. 39–48 (2011). https://doi.org/10.1145/2038642.2038651
26. Like, Zh., Song, W., Huahua, S., Hai, J., Xuanhua, Sh.: Virtual machine scheduling for parallel soft real-time applications. In: The 20th IEEE International Symposium on Modelling, pp. 525–534 (2013). https://doi.org/10.1109/MASCOTS.2013.74
27. Jin, H., Gao, W., Wu, S., Xuanhua, Sh, Xiaoxin, W., Fan, Zh: Optimizing the live migration of virtual machine by CPU scheduling. J. Netw. Comput. Appl. **34**, 1088–1096 (2011)
28. Xiangtong, Q., Jonathan, F.B., Gang, Y.: Disruption management for machine scheduling: the case of SPT schedules. Int. J. Prod. Econ. **103**(1), 166–184 (2006)
29. Rui, W., Dara, M.K., Naganajan, K.: A distributed control framework for performance management of virtualized computing environments. In: The 7th International Conference on Autonomic Computing, pp. 89–98 (2010). https://doi.org/10.1145/1809049.1809066
30. Dara, K., Jeffrey, O.K., James, E.H., Naganajan, K., Guofeng, J.: Power and performance management of virtualized computing environments via lookahead control. Int. Conf. Automonic Comput. **12**(1), 3–12 (2008). https://doi.org/10.1109/ICAC.2008.31
31. Rajiv, R., Rodrigo, N.C., Rajkumar B.: Virtual machine provisioning based on analytical performance and QoS in cloud computing environments. In: International Conference on Parallel Processing, pp. 295–304 (2011). https://doi.org/10.1109/ICPP.2011.17
32. Jim, S., Ravi, N.: Virtual Machines: Versatile Platforms for Systems and Processes. Morgan Kaufmann Publishers Inc., San Francisco (2007)
33. Jin, H., Li, D., Song, W., Like, Zh: Automatic power-aware reconfiguration of processor resource in virtualized clusters. J. Comput. Res. Dev. **48**(7), 1123–1133 (2011)
34. Timothy, W., Prashant, S., Arun, V., Mazin, Y.: Black-box and gray-box strategies for virtual machine migration. In: The 4th USENIX Conference on Networked Systems Design & Implementation, pp. 229–242 (2007). https://doi.org/10.1109/ICAC.2006.1662416
35. Hwanju, K., Hyeontaek, L., Jinkyu, J., Heeseung, J., Joonwon, L.: Task-aware virtual machine scheduling for I/O performance. In: International Conference on Virtual Execution Environments, pp. 101–110 (2009). https://doi.org/10.1145/1508293.1508308
36. Sriram, G., Arjun, R.N., Amitayu, D., Bhuvan, U., Anand, S.: Xen and co.: communication-aware CPU scheduling for consolidated Xen-based hosting platforms. In: The 3rd International Conference on Virtual Execution Environments, pp. 126–136 (2007). https://doi.org/10.1145/1254810.1254828
37. David, C.: The Definitive Guide to the Xen Hypervisor. Pearson Education Inc., Upper Saddle River (2008)
38. Pan, J.N.: A study of multivariate pre-control charts. Int. J. Prod. Econ. **105**(1), 160–170 (2007)
39. Ramadgel, P.J., Wonham, W.M.: Supervisory control of discrete event processes. In: Feedback Control of Linear and Nonlinear Systems, pp. 202–214 (1982). https://doi.org/10.1007/BFb0006830
40. Stanescu, A.M., Dumitrache, I., Curaj, A., Caramihai, S.I., Chircor, M.: Supervisory control and data acquisition for virtual enterprise. Int. J. Prod. Res. **40**(15), 3545–3559 (2002)
41. Lai, K.C., Smiley, W.C., Fred, S.: A new measure of process capability index C_{pm}. J. Qual. Technol. **20**(3), 162–175 (1988)
42. Maria, T.C., Aysun, S., Jose, M.S.: A new approach for measurement of the efficiency of C_{pm} and C_{pmk} control charts. Int. J. Qual. Res. **7**(4), 605–622 (2013)
43. Ledolter, J., Swersey, A.: An evaluation of pre-control. J. Qual. Technol. **29**(2), 163–171 (1997)
44. Pearn, W.L., Chien-Wei, W.: Production quality and yield assurance for processes with multiple independent characteristics. Eur. J. Oper. Res. **173**(2), 637–647 (2006)
45. George, D., Coste, E.C., Luminita, R., Sebastian, M.R.: The role of virtual networks in virtual enterprise. J. Mech. Eng. **52**(7), 526–531 (2006)
46. Mohamed, S.C., Habib, C., Belaı, A.: Quality control system design through the goal programming model and the satisfaction functions. Eur. J. Oper. Res. **186**(3), 1084–1093 (2008)

Ying Li Ph.D. Candidate. She received bachelor degree in 2009 and the master degree in School of Automation and Information Engineering of Xi'an University of Technology in 2012. Now she is engaged in the researches of virtual technology and cloud computing.

Jing Zhang Professor, Doctoral supervisor. He received the doctor degree in Department of Systems Engineering of Xi'an Jiao-Tong University in 1994. He has worked in Department of Computer of Xi'an University of Technology since 1977, and now he is a professor and the Ph.D. supervisor of School of Computer Science and Engineering, Xi'an University of Technology. He has published 60 papers and hosted 20 research projects in recently 10 years. Recently he concentrates on the researches of distributed system, virtualization, big data and cloud computing.

XiaoJun Chen Doctor, Senior Engineer. He received bachelor degree in 2004, the master degree in 2009, and received the doctor degree in 2012 at School of computer of Xi'an University of Technology. Now he engages on the automation of electric power systems and information technology in Shaanxi Regional Electric Power Group Co. LTD, Recently he concentrates on virtual technology and cloud computing.

JunHuai Li Professor. He received the bachelor degree in electrical automation from Shaanxi Institute of Mechanical Engineering in 1992, the master degree in computer application technology from Xi'an University of Technology in 1999, and the doctor degree in computer software and theory from Northwest University in 2002. He is currently a professor with School of Computer Science and Engineering, Xi'an University of Technology. His research interests include Internet of things technology and network computing.

JuLan Ding Physician. She received bachelor degree at Department of medicine of Xi'an JiaoTong University in 1977. Recently she concentrates on the researches of bioinformatics and big data.