Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2018) 26: 2512 – 2524

doi:10.3906/elk-1709-112

Dynamic CPU scheduling for load balancing in virtualized environments

Suresh Chandra MOHARANA*, Sibani SAMAL, Amulya Ratna SWAIN, Ganga Bishnu MUND
[1]School of Computer Engineering, KIIT Deemed to be University, Bhubaneswar, Odisha, India

Received: 12.09.2017 • **Accepted/Published Online:** 08.04.2018 • **Final Version:** 28.09.2018

Abstract: In the modern era of computing, the cloud computing platform became popular with its on-demand resource scalability feature. Virtualization is the key technology to achieve resource scalability in the cloud environment. Virtual machine monitors (VMMs) like Xen and KVM are the enabling tools for virtualizing the resources in the cloud environment. The major role of VMMs is to map virtual CPUs of virtual machines to physical CPUs, popularly known as CPU scheduling. In this study, we analyzed the CPU scheduler of Xen VMM, called Credit CPU scheduler, with respect to CPU utilization. In order to maximize the physical CPU utilization in Xen VMM, the existing Credit CPU scheduling scheme distributes the physical CPU time among all the virtual CPUs available in virtual machines based on the current weight of the virtual machine. However, this scheduling approach wastes a considerable amount of CPU time in context switching due to random allocation of virtual CPUs to the real CPU cores. In addition to that, the Credit CPU scheduler in Xen is least concerned about load balancing at both virtual and physical CPUs level. Considering all these above issues, in this paper, a dynamic CPU scheduling algorithm is presented that will distribute both single and multithreaded load fairly among virtual and real CPUs, and also handle the issues related to context switching. The proposed CPU scheduling approach is implemented in Xen VMM using the virsh interface. The experimental results indicate that the proposed CPU scheduling approach distributes the available real CPU time evenly among virtual CPUs, which leads to balanced load in the Xen environment.

Key words: Virtual machine, CPU scheduling, load balancing, virtualization

1. Introduction

Cloud computing is a modernistic computing paradigm based on the pay-per-use model. Cloud consumers are required to pay as per their resource usability. The scalability feature of the Cloud supports contraction of capital costs by reducing the physical infrastructure. In order to achieve scalability, the cloud computing model makes use of virtualization technology. Virtualization [1] is the process of creating virtual machines over underlying hardware that simulates a real system. Virtual machine monitors (VMMs) like Xen [2], KVM [3], and Vmware are the key environments for virtualizing the physical systems into virtual systems. In addition to that, there is a need for proper virtual machine management that should not only maximize resource utilization but also minimize the physical infrastructure. Virtual machine management is now drawing the attention of researchers in the field of virtualization.

In virtualization, multiple guest operating systems (called guest domains) run on top of the virtual machine manager. These guest domains can be treated as a virtual representation of real machines that possess the virtual resources to become operational. The major resources in a virtual machine are virtual CPUs (vCPUs) and virtual memory. The CPU requirements of these virtual domains are facilitated by the vCPUs attached

*Correspondence: sureshmoharana@gmail.com

to that domain. These vCPUs are mapped to real CPUs and allowed to consume CPU time from real CPUs. Normally, the number of vCPUs is always higher compared to the number of real CPUs. Hence, the allocation of real CPUs to the vCPUs is a challenging task for VMMs. The mapping of vCPUs to real CPUs can be termed as CPU scheduling within VMMs. The CPU scheduling techniques are responsible for efficient utilization of the whole system by balancing the load among all the CPUs, which is in turn known as load balancing. The aforesaid CPU scheduling approach comprises pinning vCPUs to each active domain and also pinning CPUs to vCPUs. In order to achieve load balancing, we need to balance the load of active domains in vCPU and CPU core level. The objective of this paper is to design a CPU scheduling approach that will lead to efficient utilization of both vCPUs and real CPUs. The major contribution of the paper is as follows. We have conducted a theoretical study and identified that the existing CPU scheduling approach in Xen wastes a marginal amount of CPU time in context switching by randomly pinning vCPUs to real CPUs. Hence, a dynamic CPU scheduling algorithm is presented here that distributes the single as well as multithreaded loads fairly onto vCPUs and real CPUs. It also handles the issues related to context switching.

The rest of the paper is planned as follows. The next section describes the existing CPU scheduling approaches present in the Xen environment. Section 3 presents the CPU scheduling problem and various design issues. Section 4 provides a dynamic CPU scheduling framework and its implementation details. Section 5 focuses on experimental results of the proposed approach in the Xen environment. The last section provides the concluding remarks and related future directions.

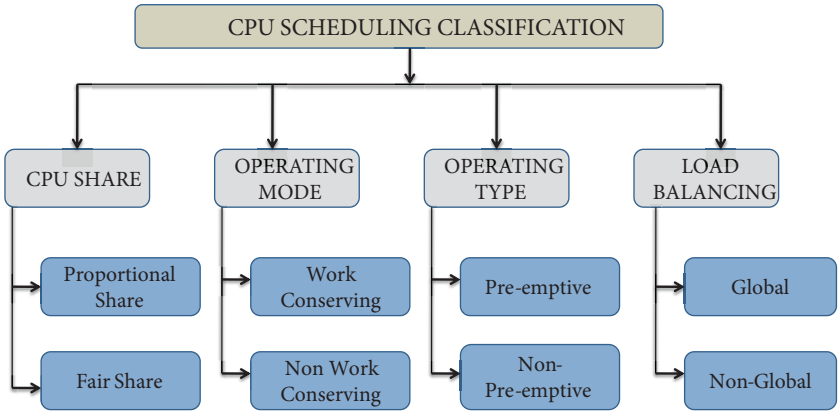

Figure 1. CPU scheduling classification.

2. Related Work

In virtualization, the scheduling of vCPUs to real CPUs is of major concern as it can affect the system performance. The CPU scheduler classification as mentioned below motivates us to explore CPU scheduling in the Xen environment. We can broadly classify the CPU scheduler as a proportional share (PS) scheduler and fair share (FS) scheduler. PS scheduling can be distinguished from FS scheduling with respect to the CPU share obtained by active domains. In PS scheduling, the available CPU is shared among active domains as per their individual weight, whereas in FS scheduling the CPU share allocated to the active domain depends on the current requirements of the CPU. We can further classify CPU schedulers into non-work conserving (NWC) mode and work conserving (WC) mode. In NWC mode, the scheduler caps the CPU share of each domain to a fixed limit, which is unupdatable. With WC mode, the active domains are given priority over inactive ones for CPU share allocation. CPU schedulers can also be distinguished as preemptive and non-preemptive

schedulers. The preemptive schedulers take scheduling decision based on the inclusion of active domain in the list of domains, whereas in non-preemptive type schedulers, the scheduling decision gets delayed until the completion of the CPU slice allocated to vCPUs of the available domains.

The scheduling classification as given in Figure 1 assists us to explore the properties of CPU schedulers used in the Xen environment. The CPU schedulers in the Xen environment can be thought of as a a) borrowed virtual time (BVT) scheduler, b) simplest earliest deadline first (SEDF) scheduler, and c) credit scheduler.

Type of Scheduling	Execution	Strategy	Mode	Global Load Balancing
BVT	Non - Preemptive	Porportional Share	WC	No
SEDF	Preemptive	Porportional Share	NWC	Yes
CREDIT	Non - Preemptive	Fair Share	WC	No

Figure 2. Characterstics of CPU schedulers in xen

An overview of the characteristics of CPU schedulers available in the Xen environment is given in Figure 2. The BVT [4] scheduling mechanism is based on virtual time of runnable virtual machines. In the BVT scheduling approach, the virtual machines having the lowest virtual time are dispatched first. Moreover, the BVT scheduling prioritizes latency-sensitive clients over real-time and interactive applications. BVT is a fair share scheduler that works in work-conserving mode. The absence of non-work conserving mode makes it inconvenient for different environments that conceptualize the SEDF scheduler in the Xen environment.

The SEDF [5] is a dynamic scheduling mechanism that ensures fairness in CPU share allocation. SEDF fairly distributes the available CPU time among the active domains that need some extra CPU share. SEDF is a proportional share scheduling mechanism that works in both work and non-work conserving mode. The SEDF scheduling mechanism misses the global load balancing, which is taken into consideration by the current implementation of the scheduler in Xen known as the credit scheduler.

In the current Xen implementation, the credit scheduler is used. The credit scheduler is a proportional share CPU scheduler for allocation of virtual CPUs to physical CPUs on SMP hosts. This scheduler ensures that no physical CPU core should remain idle when the active domains have a runnable workload. Load balancing is a part of the credit scheduler, but it has been observed that the credit scheduler looks for equal distribution of workload in vCPUs, which is not suitable for real-time situations. The load balancing at real CPU core level is also not considered in the credit scheduler.

There was a recent effort made on improving the existing credit scheduler in Xen. Lee et al. [6] provide an enhanced credit scheduler by considering the awareness of active domains towards CPU intensive applications. Govindan et al. [7] improve the existing CPU scheduling in the Xen environment by diminishing the priority inversion problem during co-scheduling of guest domains with domain 0. In the Xen ARM platform, similar techniques are employed by Yoo et al. [8] for the improvement of the existing credit scheduler. RT-Xen 1.0 [9] provides an open-source platform to integrate real-time scheduling techniques with Xen VMM in a single core environment. RT-Xen 2.0 [10] extends RT-Xen 1.0 for the inclusion of real-time scheduling approaches in multi-core environments.

Cavdar et al. [11] found that the existing scheduling algorithms in virtualization focus on sharing of

resources among different domains for efficient resource utilization. These approaches apply generic policies without taking user requirements into consideration and are mostly static in nature. This situation can be handled by employing a manager module that will assign the resources dynamically to the active domains and also take user requirements into account in order to improve system global usage. In this work, an external application is proposed that will manage CPU resource in Xen VMM by taking the requirements of host applications into consideration. This approach takes the help of Xen facilities to monitor CPU resource usage and assignment. The performance of the external manager will decrease due to the overhead of the resource monitoring and reallocation along with native scheduling. However, it ensures efficient utilization of CPU resources.

Wang et al. [12] proposed a vCPU scheduling scheme that is applicable to multi-core virtualization environments. The proposed scheduling scheme tries to address the lock holder preemption problem in virtualized multicore systems, which may lead to significant wastage of CPU cycles resulting in degradation in VM performance. In this work, the authors have investigated the available vCPU scheduling algorithms, such as co-scheduling, yield-to-head, and yield-to-tail under different vCPU over-commitment rate settings. After analyzing the above results, the authors have proposed an efficient consolidation aware vCPU scheduling scheme that dynamically selects the most suitable vCPU scheduling technique. In this vCPU scheduling scheme, the load balancing at host level and the wastage of CPU time due to context switching are not taken into consideration.

Asyabi et al. [13] proposed a novel CPU scheduler for VMMs with an objective to mitigate the variations in performance due to both I/O and CPU intensive workloads running over VMs. The authors have identified that the VMs' delivered performance depends upon the number of co-located VMs and the type of workload assigned to the VMs. To tackle this issue, the authors have presented ppXen, a CPU scheduling approach that attempts to resolve the performance variability by minimizing resource interference among co-located VMs. Firstly, ppXen classifies the running VMs in terms of processor time and I/O requirements. Then, it schedules vCPUs with complementary resource demands to reduce the interference among the vCPUs. This vCPU scheduling approach results in reduction of performance variability for I/O and CPU intensive workloads running over VMs.

Wu et al. [14] proposed a load-awareness vCPU scheduling model for appropriate allocation of resources to the VMs. Here, the authors specify that the existing credit CPU scheduler provides a proportional fair share to each VM based on its predefined weight value. Under dynamic load conditions, the performance of VMs may get degraded in the case the pre-defined weight value does not match the current workload. To address this issue, this paper proposed a CPU scheduling algorithm, known as load-awareness credit (LA-credit), that dynamically adjusts the weight values of VMs for improving the overall system performance. In this approach, the LA-credit algorithm monitors the current workload of each VM and then it modifies the existing weight of each VM so that a heavily loaded VM can obtain the required CPU share.

Miao et al. [15] identified that the existing CPU scheduling model in VMMs is based on double scheduling, where not only the processes are scheduled to vCPUs but also vCPUs are scheduled to pCPUs. As per the authors' perspective, double scheduling may result in issues including lock-holder preemption, vCPU stacking, CPU fragmentation, priority inversion, etc., which lead to severe performance degradation of the virtual machine. This paper proposed a new CPU scheduling algorithm known as FlexCore that reduces the effect of double scheduling in a virtualization environment. In this approach, the number of vPCUs of a VM is dynamically adjusted by taking CPU ballooning into consideration. This significantly improves the performance by eventually

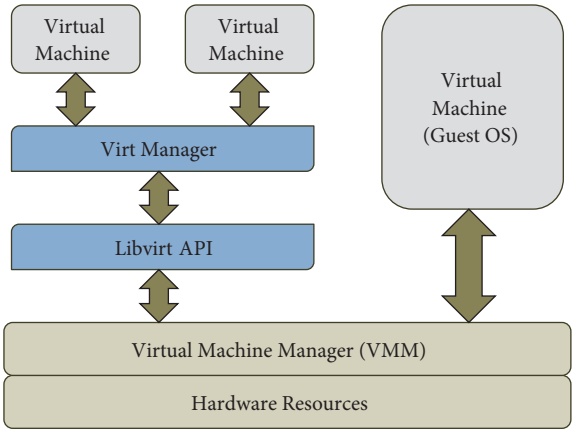

Figure 3. Virtualization model.

avoiding the unnecessary scheduling at the VMM layer.

Xilong et al. [16] presented a novel VM scheduling approach to handle energy efficiency issues in the presence of mixed-workloads in cloud systems. In this paper, it is identified that the traditional VM schedulers distribute CPU time fairly among multiple VMs with an objective to improve the CPU energy efficiently. However, this approach cannot be directly applicable to I/O energy improvement. In addition to that, the presence of I/O intensive workload may increase the rate of context switching, which leads to wastage of energy. In order to address these issues, this paper proposed a VM scheduling approach, known as share-reclaiming policy, where extra CPU time is allocated to the VMs frequently interrupted by I/O operations. This approach gives more chance for the I/O intensive workload to get scheduled and improves energy management in a better way.

3. CPU scheduling

In this section, we briefly describe the virtualization environment and then provide a mathematical overview of the CPU scheduling problem applicable to Xen or any other VMM.

3.1. Virtualization stack

Virtual machine managers like Xen and KVM provide a virtualization layer that emulates the underlying hardware for running operating systems of different architectures as shown in Figure 3. In these VMMs, the virtual machine can either be built directly on a virtual machine manager or use the virsh interface of libvirt. The virtual machines can also be managed on top of user-friendly interfaces like virt manager. Virt manager [17] takes the assistance of libvirt API to manage the active domains built upon it. CPU scheduling is an important functionality of the virtual machine manager where the vCPU of active domains is mapped to real CPUs for execution of tasks allocated to the active domains. The mapping between vCPUs and real CPUs can change dynamically as the load can vary with time in the active domains. The load balancing at vCPU and real CPU core level is an inherent module of CPU scheduling for effective use of available resources. The existing approach of proportionate CPU time allocation in Xen VMM does not take care of load balancing at vCPU and real CPU core level. The next section gives a mathematical overview of the CPU scheduling problem in a virtualization environment.

3.2. CPU scheduling problem

This section provides a mathematical model of the CPU scheduling problem and defines the cost function of scheduling vCPUs to real CPUs within VMMs. The CPU scheduling problem [18] can be defined as the mapping of vCPUs of active domains to the real CPUs of the physical system for completion of tasks assigned to the active domains. In a virtualized environment, the allocation of tasks to the active domains includes assignment of tasks to vCPUs and allocation of vCPUs to real CPUs. Here, it is assumed that the tasks can be a single or multi-threaded type and tasks once assigned to the active domain cannot be preempted during its execution. Considering the above facts, the CPU scheduling problem can be formalized in the following way.

Let T denote the set of tasks that need to be distributed over available active domains. Let $C = \{C_1, C_2, ...C_n\}$ denote a homogeneous set of real CPUs available in the physical system. Let the domains or guest operating systems running on the physical system be denoted by $D = \{D_1, D_2, ..., D_p\}$, where p denotes the number of domains on the physical system. $V(D_k) = \{V_1, V_2, ..., V_m\}$ denotes the number of virtual CPUs belonging to the k_{th} domain. The scheduling of set of tasks to the active domain D_k can be defined as a tuple $s = (\alpha, \beta)$, where α maps the set of tasks to the vCPUs (α: $T \rightarrow V(D_k)$) and β maps the vCPUs to real CPU cores (β: $V(D_k) \rightarrow C$) of the physical system.

The cost function $f(s)$ of CPU scheduling s can be defined as

$$f(s) = \sum_{i=1}^{m} \sum_{j=1}^{n} CT_{ij} \tag{1}$$

such that $\forall k$, $|V(D_k)| \leq |C|$ and $|T| \leq |V(D_k)|$.

The cost function $f(s)$ denotes the cumulative load of a host, n denotes the maximum number of vCPUs assigned to each pCPU, m denotes the number of pCPUs, and CT_{ij} denotes the CPU cycles taken by the i_{th} vCPU from the j_{th} pCPU. The ultimate goal for the vCPU scheduler is to minimize $f(s)$.

4. Proposed scheduling model

The workload assigned to the virtual machines can be either single or multi-threaded type. The multi-threaded applications use the real CPUs, one for each thread; hence it requires a higher proportion of CPU time for execution. By keeping these applications in mind, the scheduling framework is designed as shown in Figure 4.

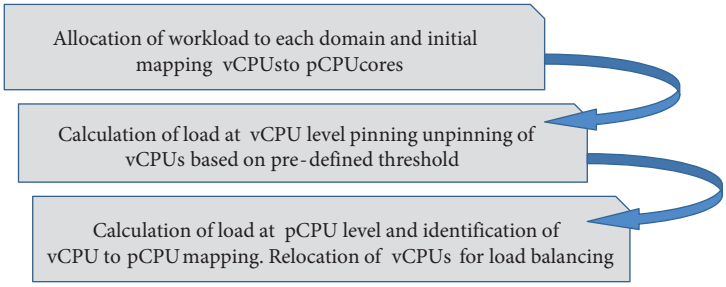

Figure 4. Proposed CPU scheduling framework.

The existing CPU scheduling approach maps the vCPUs of the active domain to real CPUs in a random way as per the domain requirement. After assigning workload to the active domains, this approach will waste additional CPU time for pinning of an appropriate number of vCPUs to real CPUs of each active domain.

In order to avoid the wastage of CPU time, the *domain_initialization*() module as given in Algorithm 1 initially maps each vCPU of each active domain to a unique real CPU core. As per [19] the mapping of vCPU to unique real CPU core may lead to load balancing. In addition to this, the proposed scheduling approach moves around *balance_vcpus*() module as given in Algorithm 2 followed by *balance_cpus*() module as given in Algorithm 3 until the load balancing at vCPU and real CPU core level is achieved. The role of each module of the proposed scheduling model is described below. The proposed CPU scheduling approach is applicable to the symmetric multiprocessing (SMP) environment. The SMP environment is based on uniform memory access (UMA), wherein the real CPU cores share the main memory among each other. Thus, the proposed CPU scheduling approach does not make use of non-uniform memory access (NUMA) for its implementation.

The *domain_initialization*() module will find the current CPU mapping using the bitmap $cpumaps_i$. The vCPUs, which are pinned to more than one CPU core, are now pinned to the unallocated real CPU core since it is assumed that the number of vCPUs belonging to a domain should not be more than the real CPU core. Therefore, each vCPU of a domain can be mapped to a unique CPU core.

Procedure *domain_initialization()*
  $no\_act\_domains$=number of domains currently active
  $nrVirtCpu_i$=number of vCPU pinned to domain i
  $cpumaps_i$=bitmap representing vCPU to pCPU mapping
  Establish a connection with XEN VMM
  Find the number of currently active domains $no\_act\_domains$
  for i=1 to $no\_act\_domains$ **do**
    find $nrVirtCpu_i$ of each domain i
    for j=1 to $nrVirtCpu_i$ **do**
      find the mapping of vCPU to pCPU using $cpumaps_i$
    end for
    for j=1 to $nrVirtCpu_i$ **do**
      if number of set bits in $cpumaps_i > 1$ **then**
        find pCPU not allocated to domain i
        pin the vCPU to that pCPU
      end if
    end for
  end for
  for i=1 to $no\_act\_domains$ **do**
    $total\_domain\_load = \sum_{k=1}^{nrVirtCpu_i} cputime_j$
  end for
  if $no\_act\_domains > 0$ **then**
    call procedure *balance_vcpus*()
    call procedure *balance_cpus*()
  end if

Algorithm 1: Algorithm for domain initialization.

After the initialization phase, the *balance_vcpus*() module as well as *balance_cpus*() module are initiated for load balancing at vCPU and pCPU level. The *balance_vcpus*() module initially computes the current threshold to decide the vCPU allocation. The current threshold represents the minimum percentage of load received by any vCPU from pCPUs considering all vCPUs are fully loaded. Hence, the current threshold,

$curr\_thr$ can be represented as

$$curr\_thr = 100/max\_vcpu\_count, \tag{2}$$

where max_vcpu_count denotes the maximum number of vCPUs pinned to any active domain i. After the threshold gets computed, the average load of each active domain is compared with the threshold. In the case the domains' average load per vCPU is greater than $curr\_thr$ then additional vCPU is assigned to the the current domain. The assignment of extra vCPU to the domain will reduce the load of existing vCPUs of the domain as compared to the vCPUs of other domains. This can be achieved due to the concurrent execution of the available task in the vCPUs of the domain, which in turn are pinned to different real CPUs. This approach can also work for sequential tasks as an additional vCPU may not get any load of the domain and later may be unpinned for underutilization. In the final step of the above algorithm, the vCPUs with very less utilization are identified and unpinned from active domains.

Procedure *balance_vcpus()*
  Let $nCPU$=number of real CPU cores
  Let max_vcpu_count=maximum number of vCPUs pinned to any domain i
  Compute the current threshold $curr\_thr$=100/max_vcpu_count
  for i=1 to $no\_act\_domains$ **do**
    if $total\_domain\_load/nrVirtCpu_i > curr\_thr$ and $nrVirtCpu_i < nCPU$ **then**
      pin one additional vCPU for domain i
      update the $cpumaps_i$
    end if
    for j=1 to $nrVirtCpu_i$ **do**
      if set bits in $cpumaps_i > 1$ **then**
        find the pCPU with minimum number of pinned vCPU
        pin the pCPU to the vCPU
      end if
    end for
  end for
  for i=1 to $no\_act\_domains$ **do**
    find the underutilized vCPU
    unpin the vCPU from domain i
  end for

Algorithm 2: Algorithm for load balancing at vCPU level.

After the completion of the *balance_vcpus*() module, the *balance_cpus*() module is initiated for balancing load at real CPU core level. The *balance_cpus*() module identifies the maximum loaded pCPU for load balancing at real CPU core level in the following approach. Let C_i and C_j be two pCPUs having N_i and N_j number of vCPUs pinned to it. Let L_i and L_j be the total load available in C_i and C_j, respectively. In the case L_i is the same as L_j ($L_i = L_j$), the pCPU having maximum numbers of pinned vCPUs will be selected as maximum loaded pCPU. However, if L_i is higher as compared to L_j ($L_i > L_j$) and the average load of C_i is less as compared to average load of C_j, i.e. $L_i/N_i < L_j/N_j$ then the i_{th} pCPU is considered as maximum loaded pCPU. In the case the above conditions are not applicable, then if the average load of C_i by taking the number of vCPUs of C_j into consideration is less than the average load of C_j by taking the number of vCPUs of C_i into consideration, i.e. $L_i/N_j < L_j/N_i$, the i_{th} pCPU is considered as maximum loaded pCPU. This approach is helpful in identifying the maximum loaded pCPU when some pCPUs have fewer highly loaded

vCPUs and some pCPUs have a higher number of lightly loaded vCPUs pinned to them.

Procedure *balance_cpus()*
max_loaded_cpu=pCPU with maximum load
min_loaded_cpu=pCPU with minimum load
min_vCPU=vCPUs with minimum load pinned to minimum loaded CPU
for i=1 to *nCPU* **do**
find the $pcpu\_vcpu\_count_i$ = number of vCPU pinned to ith pCPU
find the $total\_load\_cpu_i$ = total load of ith pCPU
find the $avg\_load\_cpu_i = total\_load\_cpu_i / pcpu\_vcpu\_count_i$
end for
find the *max_loaded_cpu* and *min_loaded_cpu*
if average load of *max_loaded_cpu*-average load of *min_loaded_cpu* is less **then**
find *min_vCPU* pinned to *min_loaded_cpu*
find the vCPU with minimum load among *min_vCPU*
end if
pin the vCPU to the *max_loaded_cpu*

Algorithm 3: Algorithm for load balancing at pCPU level.

The maximum loaded pCPU is used in the *balance_cpus*() module to decide on any update in vCPU and pCPU mapping for load balancing at real CPU core level. This module is initiated by identifying the maximum and minimum loaded pCPU among the available real CPUs. The current vCPU to pCPU mapping remains unchanged, if the load difference between the maximum and minimum loaded pCPU is very less. However, if the load difference is higher, then the minimum loaded vCPU from the vCPUs pinned to the minimum loaded pCPU is identified. Then the minimum loaded vCPU is pinned to the maximum loaded pCPU to achieve load balancing at real CPU core level.

4.1. Complexity analysis

The *balance_vcpus*() module considers both the active domains as well as the vCPUs pinned to each active domain. The complexity of the *balance_cpus*() module will be $O(mn)$, assuming there are m real CPU cores and each CPU core is pinned with maximum n number of vCPUs. The complexity of *balance_cpus*() is also $O(mn)$ by taking the above assumptions into consideration. The *domain_initialization*() module after mapping the vCPUS with unique real CPU cores makes use of both *balance_vcpus*() and *balance_cpus*() module in order to achieve load balancing at host level. Under similar assumptions, the overall complexity of the proposed CPU scheduling model will be $O(mn)$.

5. Experimental results and discussion

This section provides the performance evaluation of the proposed CPU scheduling model under various load conditions of the active domains. The proposed CPU scheduling model is implemented on a Lenovo Thinkstation server with core i5 processor and 8 GB of RAM. Xen 4.2.0 is used as the virtual machine manager and Centos 6.0 is used as virtual machine instances. The virtual machine is built using the para-virtualized operating systems.

5.1. Experimental design

The proposed CPU scheduling algorithm is implemented on top of libvirt by using the features of libvirt API. Libvirt provides the virsh interface to interact with the underlying virtual machine manager. The virsh console

is used for the implementation of our proposed scheduling model. Here it is observed that the virtual machine instances built using the full virtualization technique are unable to adopt the dynamic changes in vCPU-pCPU mapping for load balancing at vCPU as well as pCPU level. Nevertheless, it has been found that the para-virtualized virtual machines can adopt the changes in vCPU to pCPU mapping dynamically. Thus, the para-virtualized operating systems are used here as the virtual machine instances for experimenting with the proposed pCPU scheduling model, which is based on dynamic update of vCPU to pCPU mapping for load balancing. In order to test the proposed CPU scheduling approach, two different kinds of workload are taken into consideration [20]. Specifically, we have considered a load that can run in a single thread and another multi-threaded load that will utilize multiple threads for its execution. These loads are assigned to the active domains during the execution of the proposed approach in the background to evaluate the performance of the proposed approach.

The total domain load of each active domain is taken into consideration for comparing the existing credit scheduler approach with the proposed CPU scheduling approach. The total load of the ith domain can be mathematically defined as

$$total\_domain\_load_i = \sum_{k=1}^{n} cpu\_time(vcpu_k) \tag{3}$$

where n is the number of vCPUs pinned to the i_{th} domain. The total domain load of the i_{th} domain is the sum of CPU time consumed by each vCPU of the domain.

For evaluating the proposed CPU scheduling approach, we have considered the situations where single and multiple virtual machines are active at the same instance of time. The virtual machines are configured with 4 virtual CPUs and 2048 MB of RAM. The virtual machines are assigned to single as well as multi-threaded loads. The proposed algorithm runs in iterations and the experimental results are recorded for each iteration. The experimental results marked with "without load balancing" show the total domain load without applying the proposed approach. The results marked with "with load balancing" show the improved total domain load after application of the proposed CPU scheduling scheme. At first, a single multi-threaded load is assigned to a single VM and the total CPU load is computed using the existing and proposed CPU scheduling approach in different iterations. It has been identified that the total CPU load obtained using the proposed approach is better than the existing CPU scheduler in Xen as shown in Figure 5.

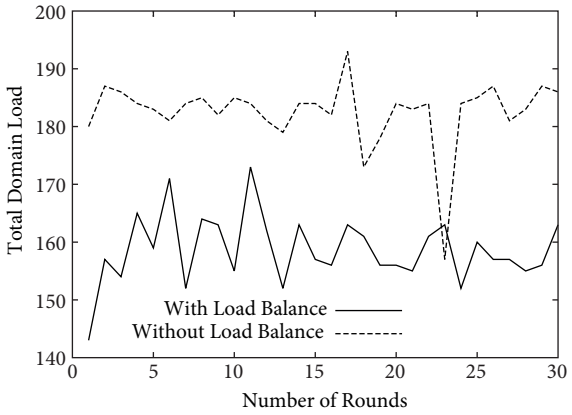

Figure 5. One VM and one multithreaded load.

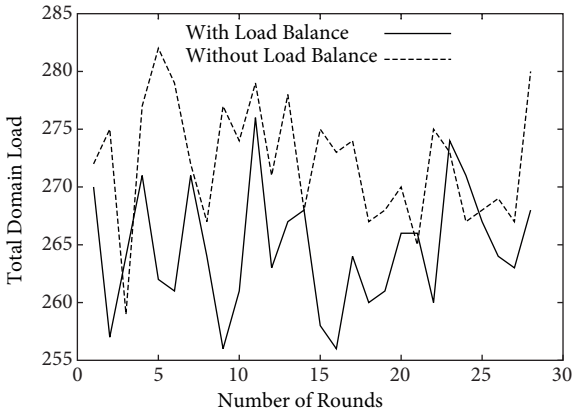

Figure 6. One VM and many multithreaded load.

The next experiment considers the application of multiple multi-threaded load onto a single virtual machine. The total CPU load both with and without load balancing is recorded for different iterations as depicted in Figure 6. In this figure, it is observed that the "with load balancing" scheduler performs better in terms of total domain load in most of the iterations as compared to the "without load balancing approach".

In another part of the experiment, two non-multi-threaded applications are taken into consideration and a single virtual machine is used for their execution. Based on this experimental result, Figure 7 indicates that the proposed scheduling approach marginally reduces the total CPU load of the active domain for each iteration as compared to the existing CPU scheduler in Xen VMM.

The next layout takes multiple active virtual domains for evaluation of the proposed CPU scheduling scheme. In this case, two active virtual machines are considered simultaneously and a single multi-threaded load is allocated to these active domains. It has been identified that with multiple iterations the total CPU load consumed by the active domains using the "with load balancing" scheduler is less than the total CPU load consumed in the "without load balancing" scheduler as illustrated in Figure 8.

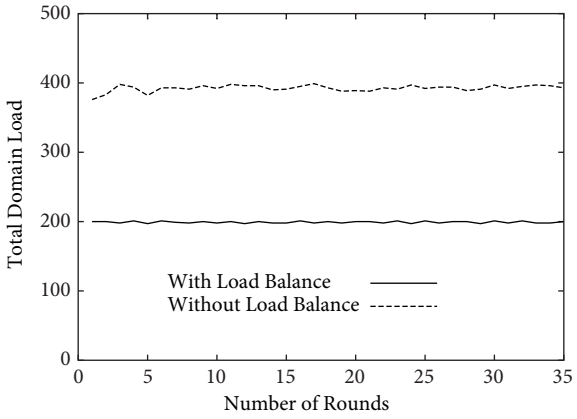

Figure 7. One VM and two non-multi-threaded load.

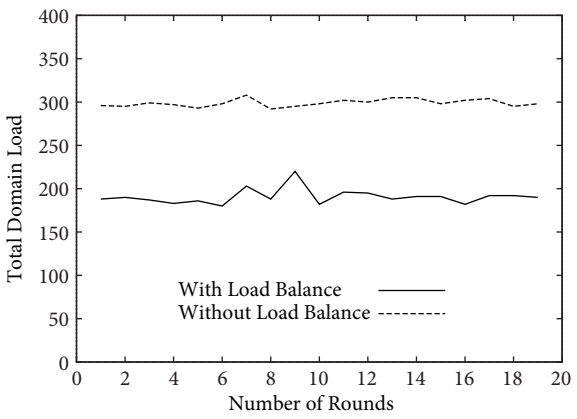

Figure 8. Two VMs and one multi-threaded load.

In the next evaluation, multiple multi-threaded loads are assigned to two virtual machines for their execution. It has been observed that the performance of the proposed CPU scheduler in terms of total CPU load is better than that of the existing credit scheduler in Xen. Figure 9 shows that the total CPU load in the active domains is always less as compared to that in the "with load balancing" scheduler for a different number of iterations. Finally, Figure 10 shows the total CPU load of the active domains by considering two virtual machines for execution of multiple non-multi-threaded application. The graph given in this figure indicates that the total CPU load of all the active domains is almost the same for both the "with load balancing" and "without load balancing" scheduler in each iteration. Thus, it can be stated that apart from the currently stated case the proposed "with load balancing" scheduler outperforms the "without load balancing" scheduling with respect to the total CPU load of active domains.

From the above analysis and experimental results of the proposed model, a few limitations have been identified that require future attention for the improvement of the proposed model. As the proposed model makes use of the virsh interface for its realization, the experimental results may be biased due to incurred CPU time for communication between virsh and Xen VMM. Hence, the proposed model needs to be tested in the actual Xen environment by incorporating the proposed scheduling model in Xen distribution. In addition to this, since the proposed approach equally distributes the domain loads among all the available pCPUs, unlike

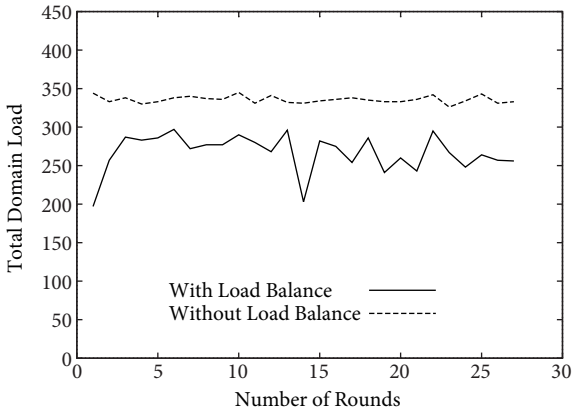

Figure 9. Two VMs and many multi-threaded load.

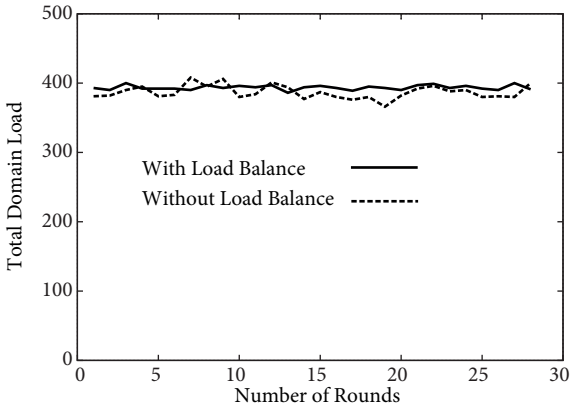

Figure 10. Two VMs and many non-multi-threaded load.

overloading a few available real CPU cores, the proposed approach may be more suitable for multi-threaded applications as compared to non-multi-threaded applications.

6. Conclusion

This paper proposed a dynamic CPU scheduling approach in Xen VMM. The existing credit CPU scheduler of Xen VMM randomly maps vCPUs to real CPU cores, which consumes a significant amount of CPU time for context switching. Moreover, this credit CPU scheduler does not take load balancing into account at the host level. Hence, to address the aforesaid issues, a CPU scheduling approach at the host level is proposed. This proposed approach performs load balancing both at vCPU level as well as at pCPU level in a novel way. At the initial stage, the vCPUs of VMs are mapped to real CPU cores uniquely and then further exchange of pinning between vCPUs and real CPU cores is done depending upon current load at the vCPUs.

The proposed model is implemented in Xen VMM using the virsh interface. The experimental results indicate that the proposed CPU scheduling approach fairly distributes the available real CPU time among vCPUs and achieves load balancing at the host level. As a future work, the proposed CPU scheduling model can be implemented in Xen VMM at the kernel level to evaluate the effectiveness of the same.

References

[1] Xing Y, Zhan Y. Virtualization and cloud computing. In: Zhang Y, editor. Future Wireless Networks and Information Systems. Lecture Notes in Electrical Engineering. Berlin, Germany: Springer, 2012. pp. 305-312.

[2] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles; 19–22 October 2003; Bolton Landing, NY, USA. New York, NY, USA: ACM. pp. 164-177.

[3] Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. KVM: the linux virtual machine monitor. In: Proceedings of the Linux symposium; 27–30 June 2007; Ottawa, Ontario, Canada. pp. 225-230.

[4] Duda KJ, Cheriton DR. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In: Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles; 12–15 December 1999; Charleston, SC, USA. New York, NY, USA: ACM. pp. 261-276.

[5] Cherkasova L, Gupta D, Vahdat A. Comparison of the three CPU schedulers in Xen. SIGMETRICS Performance Evaluation Review 2007; 35: 42-51.

[6] Lee M, Krishnakumar AS, Krishnan P, Singh N, Yajnik S. Supporting soft real-time tasks in the xen hypervisor. In: Proceedings of the 6th ACM SIGPLAN International Conference on Virtual Execution Environments; 17–19 March 2010; Pittsburgh, PA, USA. New York, NY, USA: ACM. pp. 97-108.

[7] Govindan S, Nath AR, Das A, Urgaonkar B, Sivasubramaniam A. Xen and co.: communication-aware CPU scheduling for consolidated xen-based hosting platforms. In: Proceedings of the 3rd International Conference on Virtual Execution Environments; 13–15 June 2007; San Diego, CA, USA. New York, NY, USA: ACM. pp. 126-136.

[8] Yoo S, Kwak K, Jo J, Yoo C. Toward under-millisecond I/O latency in xen-arm. In: Proceedings of the Second Asia-Pacific Workshop on Systems; 11–12 July 2011; Shanghai, China. New York, NY, USA: ACM. pp. 14:1-14:5.

[9] Xi S, Wilson J, Lu C, Gill C. RT-Xen: Towards real-time hypervisor scheduling in xen. In: 2011 Proceedings of the Ninth ACM International Conference on Embedded Software; 9–14 October 2011; Taipei, Taiwan. New York, NY, USA: ACM. pp. 39-48.

[10] Xi S, Xu M, Lu C, Phan LTX, Gill C, Sokolsky O, Lee I. Real-time multi-core virtual machine scheduling in Xen. In: 2014 International Conference on Embedded Software; 12-017 October 2014; New Delhi, India. New York, NY, USA: ACM. pp. 27:1-27:10.

[11] Çavdar D, Chen LY, Alagoz F. Priority scheduling for heterogeneous workloads: tradeoff between evictions and response time. IEEE Syst J 2017; 11: 684-695.

[12] Wang B, Cheng Y, Chen W, He Q, Xiang Y. Efficient consolidation-aware VCPU scheduling on multicore virtualization platform. Future Gener Comp Sy 2016; 56: 229-237.

[13] Asyabi E, Sharifi M, Bestavros A. ppXen: a hypervisor CPU scheduler for mitigating performance variability in virtualized clouds. Future Gener Comp Sy 2018; 83: 75-84.

[14] Wu J, Wang C, Li J. LA-credit: a load-awareness scheduling algorithm for xen virtualized platforms. In: Proceedings of IEEE International Conference on Big Data Security on Cloud; 9–10 April 2016; New York, NY, USA. New York, NY, USA: IEEE. pp. 234-239.

[15] Miao T, Chen H. Flexcore: dynamic virtual machine scheduling using VCPU ballooning. Tsinghua Sci Technol 2015; 20; 7-16.

[16] Xilong Q, Peng X. An energy-efficient virtual machine scheduler based on CPU share-reclaiming policy. International Journal of Grid and Utility Computing 2015; 6: 113-120.

[17] Goto Y. Kernel-based virtual machine technology. Fujitsu Sci Tech J 2011; 47: 362-368.

[18] Weng C, Wang Z, Li M, Lu X. The hybrid scheduling framework for virtual machine systems. In: Proceedings of the ACM International Conference on Virtual Execution Environments; 11–13 March 2009; Washington, DC, USA. New York, NY, USA: ACM. pp. 111-120.

[19] Tseng Y, Chung Y. An enhanced CPU scheduler for xen hypervisor to improve performance in virtualized environment. Comm Com Inf Sc 2012; 7: 62-67.

[20] Xu X, Shan P, Wan J, Jiang Y. Performance evaluation of the CPU scheduler in xen. In: 2008 International Symposium on Information Science and Engineering; 20–22 December 2008; Shanghai, China. New York, NY, USA: IEEE. pp. 68-72.