Hindawi
Journal of Sensors
Volume 2018, Article ID 4628245, 10 pages
https://doi.org/10.1155/2018/4628245

Research Article

Mcredit2: Enhanced High-Performance Xen Scheduler via Dynamic Weight Allocation

Minsu Kang and Sangjun Lee

Department of Computer, Soongsil University, Seoul 156-743, Republic of Korea

Correspondence should be addressed to Sangjun Lee; sangjun@ssu.ac.kr

Received 31 October 2017; Accepted 22 January 2018; Published 26 March 2018

Academic Editor: Mucheol Kim

Generally, operating only a single host on a single server results in hardware underutilization. Thus, hypervisors (e.g., Xen) have been developed to allow several hosts to operate on a single server. The Xen hypervisor provides processor schedulers (e.g., Credit and Credit2 schedulers) to assign processors to each host. The Credit2 scheduler provides work assurance to the domain relative to latency and it evenly assigns processors to each domain. In addition, the Credit2 scheduler can assign a weight value to each host. A greater host weight value allows processors to be assigned to a host for longer periods. However, the Credit2 scheduler shows poorer performance than the basic Credit scheduler, which utilizes idle processors. In this paper, we propose the Mcredit2 scheduler, which improves the Credit2 scheduler. The Credit2 scheduler takes no action when the load on a specific domain causes increased processor usage. The proposed Mcredit2 scheduler allows a domain to quickly process loads by temporarily assigning a greater weight value to a host with high processor usage. In addition, we introduce a processor monitoring tool that visualizes the processor usage.

1. Introduction

In recent years, with the significant advances in computer and communication technologies, which are new big leaps in the digital world, a term called "Internet of Things" (IoT) made its appearance [1]. Only in 2011 did the number of interconnected devices on the planet overtake the actual number of people, and it is expected to reach 24 billion devices by 2020 [2]. In such an environment, IoT servers are required to efficiently process large-scale data collected from those devices. Generally, using only a single host on a high-availability server can result in underutilized hardware. Table 1 shows the hardware performance of an IBM server [3]. To minimize hardware underutilization, hardware virtualization techniques [4, 5] have been studied. Server virtualization [6, 7] techniques allow several hosts to operate on a single server. The use of hypervisors [8, 9] to support server virtualization and reduce hardware underutilization has also been studied. Currently, hypervisors, such as KVM [10] and Xen [11], have gained popularity.

With Xen, the host is called the domain, and Xen requires a scheduler to assign processors to each domain. The scheduler allocates a physical processor to the virtual processors of each domain to perform operations.

Borrowed Virtual Time, Atropos, and Round Robin schedulers [12] were used in early versions of Xen. Currently, Xen 4.5.0 supports the Simple Earliest Deadline First (SEDF) [13], Credit [14], Credit2 [15], Real-time Deferrable Server [16], and Avionics Application Standard Software Interface [17] schedulers. The Credit scheduler is commonly used among these schedulers.

The Xen hypervisor adopts the Credit2 scheduler beginning with Version 4.6 [17]. The Credit2 scheduler can assign a weight value to each domain. The higher the weight value, the longer the time for which processors can be assigned. In this paper, we introduce the Mcredit2 scheduler as an improvement to the Credit2 scheduler. The Mcredit2 scheduler temporarily allocates a greater weight value to a domain having higher processor utilization. This allows domains that suddenly require many processors (e.g., web servers) to

Table 1: System x3950 X6 specifications [18].

Processor	Up to eight Intel Xeon E7-8800 v2 processors, each in a compute book. Each processor has 15 cores (up to 2.8 GHz), 12 cores (up to 2.6 GHz), 10 cores (up to 2.2 GHz), eight cores (up to 2.0 GHz), or six cores (up to 3.4 GHz). There are three QPI links (up to 8.0 GTps each). Up to 1600 MHz memory speed. Up to 37.5 MB L3 cache. Intel C602J chipset.
Memory	Up to 192 DIMM sockets (24 DIMMs per processor, installed in the compute book). RDIMMs and load-reduced DIMMs are supported, but memory types cannot be intermixed. Memory speed up to 1600 MHz.
Memory maximums	With RDIMMs: up to 3 TB with 192 × 16 GB RDIMMs and eight processors. With LRDIMMs: up to 12 TB with 192 × 64 GB LRDIMMs and eight processors.
Memory protection	ECC, Chipkill, RBS, memory mirroring, and memory rank sparing.

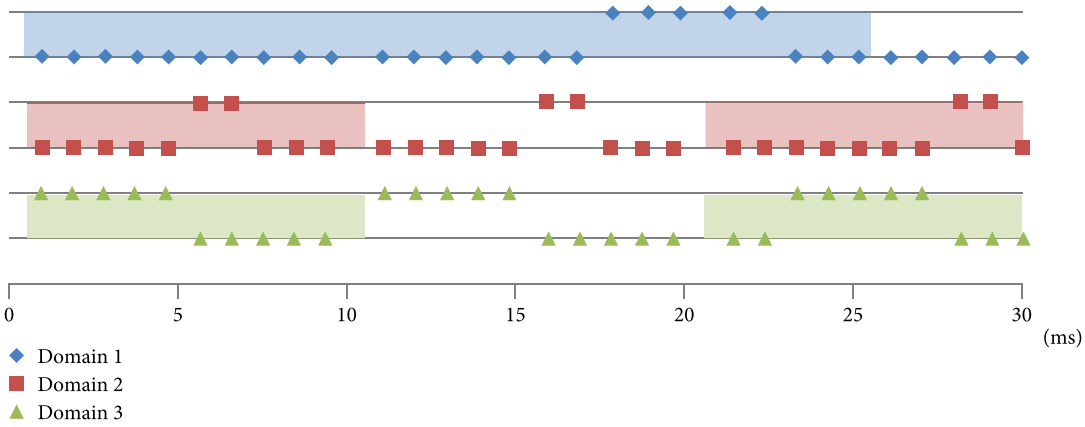

Figure 1: SEDF scheduler CPU allocation.

operate more flexibly. In this paper, we extend our previous work [19] and present more experimental results and intensive analysis that were not reported in the previous work.

The server administrator can allocate domain resources by checking whether the current CPU usage per domain is high. It can also check the average CPU usage in the domain for a certain unit time. The Xen hypervisor provides a monitoring tool such as xentop [20] that shows information about the Xen system in real time. However, with xentop, only the usage time of the current CPU is indicated with text, creating difficulties for a server administrator. In this paper, we propose a virtual processor monitoring tool called Mcredit2 that visually displays the CPU usage for each domain and shows the CPU usage time, usage percentage, and average usage time.

The remainder of this paper is organized as follows. In Section 2, the basic Xen hypervisor scheduler is described. In Section 3, the proposed scheduler and monitoring tool implementation are described. In Section 4, scheduler performance is evaluated based on the implementation results. Finally, the paper is concluded in Section 5.

2. Backgrounds

Here, the SEDF, Credit, and Credit2 schedulers are described to facilitate a comparison with the Mcredit2 scheduler.

2.1. SEDF Scheduler. The SEDF scheduler employs a real-time scheduling method [21], that is, the scheduler assigns higher priority to the process having the earliest deadline.

Each domain has a period parameter and a slice parameter. Each domain is executed for nanoseconds of the slice parameter value and every nanosecond of the period parameter value. These parameters are used to determine the domain having the earliest deadline.

For example, the assumed domains 1, 2, and 3 have period and slice parameters of 25 and 5, 10 and 2, and 10 and 5, respectively. Domains 2 and 3 have the shortest deadlines, that is, a period parameter of 10. Here, domain 2 will be executed within 5 ms and domain 3 will wait for 8 ms. Therefore, domain 3 is scheduled first. Domain 2's deadline becomes shorter than that of domain 3 after scheduling. After domains 2 and 3 are scheduled for 20 ms, domain 1 is scheduled for 5 ms. Figure 1 shows a diagram of this scheduling process. In Figure 1, the horizontal axis shows the elapsed time and the background color changes depending on the period parameter. The increase of the value reflects scheduling.

2.2. Credit Scheduler. Currently, the Credit scheduler [22] is the basic Xen scheduler. The Credit scheduler allocates symbolic money called credit to domains. Each domain pays credits to use processors. The Credit scheduler ensures that all work is processed. Even if all credits assigned to a domain are used, additional processors can be utilized if there are idle processors.

With the Credit scheduler, each domain has a weight value parameter and a capacity parameter. The weight value determines when a domain can use processors. Credits are assigned to a domain in proportion to the weight value. For

example, if domain 1 has a weight value of 512 and domain 2 has a weight value of 256, twice the number of credits are assigned to domain 1. As a result, domain 1 processors can be used for twice as long as domain 2 processors. The default weight value is 256, the minimum is 1, and the maximum is 65,535. The capacity parameter determines the upper limit of the processors to be used in the domain. A capacity value of zero means that there is no upper limit. The upper limit of one processor increases each time the capacity is increased by 100. The processor utilization cannot be over capacity, even with idle processors.

The basic Credit scheduler algorithm is as follows. Credits are assigned to all domains every 30 ms relative to the weight value. The virtual processors using processors pay credits every 10 ms. With the Credit scheduler, virtual domain processors are assigned BOOST, UNDER, OVER, and IDLE states. Processors in the BOOST state are given the highest priority, followed by UNDER, OVER, and IDLE states. The processor enters the BOOST state when an idle virtual processor attempts to process a task. A virtual processor enters the UNDER state when using a physical processor. The processor enters the UNDER state when the virtual processors have remaining credits. The processor enters the OVER state when the virtual processors do not have any credits. When a virtual processor is not in use, it enters the IDLE state. The state of the virtual processors determines the priority queue assigned to each according to priority.

The Credit scheduler has limitations. For example, it saves remaining credits and uses them for subsequent processing tasks. If the credits of a given domain are greater than a certain value, that domain becomes inactive and the unused credits are discarded. If the domain is executing multimedia, processors remain idle much of the time, because they are only required when decoding multimedia. The domain's virtual processors enter the BOOST state and are positioned at the top of the priority queue when decoding. After decoding, they enter the UNDER state and compete with other virtual processors. These processors rank far below other processors in priority and are decoded erratically, because the domain state is changed to inactive and saved credits are discarded. As a result, multimedia playback may be terminated.

2.3. Credit2 Scheduler. The Credit2 scheduler [11] is intended to address the weaknesses of the Credit scheduler. Like the Credit scheduler, the Credit2 scheduler provides credits to domains to use processors, and each domain must pay credits to use the processors. The Credit2 scheduling algorithm differs from that of the Credit scheduler.

The Credit2 scheduler algorithm is described as follows. The same credits are assigned to all domains and virtual processors pay credits when using a processor. In this case, the amount of credits paid by virtual processors varies with the weight value of the domain, even if processors are used for the same amount of time. A greater domain weight value results in lower credit costs. Concurrently, the credits to be paid increase in proportion to the greatest domain weight value. For the domain with the greatest weight value, processors can be used for 10 ms and all credits are used. Regarding the priority assigned to each virtual processor, the more credits remaining, the higher the priority becomes. When a specific virtual processor uses all its credits, the credit value of all virtual processors will be initialized.

Another difference between the Credit2 and Credit schedulers is how virtual processors are prioritized in the queue. The Credit scheduler has one priority queue per processor, and one processor schedules several virtual processors. With the Credit2 scheduler, there is one priority queue for each set of processors.

2.4. Xentop. Xen provides a monitoring tool called xentop [20] that operates as shown in Figure 2.

It is difficult to check the amount of usage of virtual processors in real time because xentop reflects only their execution time. Improved monitoring tools are required for an administrator to check the usage amount of the virtual processors for each domain.

3. Proposed Techniques

In this section, we describe the Mcredit2 scheduler and the implementation of the virtual processor monitoring tools.

3.1. Mcredit2 Scheduler. The Mcredit2 scheduler dynamically changes the weight value depending on the given situation. In contrast, the weight value is only changed by an administrator with the Credit2 scheduler. With the Mcredit2 scheduler, for an insufficient number of credits, the initialization of credits for all processors is executed when virtual processors attempt to use physical processors. At this time, a variable is assigned to the domain that has virtual processors with insufficient credits to increase the variable. At fixed intervals, the weight value of the domain with the greatest variable is increased. This weight value increases during the BOOST state. If the number of processors for a specific domain increases, the virtual processors in that domain will not have sufficient credits and the initialization of credits increases. Therefore, the domain can perform the given job with an increased weight value.

Figure 3 shows the structures used in the Mcredit2 scheduler, such as the mcsched_runqueue_data, mcsched_dom, and mcsched_vcpu structures. Here, runq in the mcsched_runqueue_data structure is a list of viable virtual machines, and svc is a list of all virtual processors in that runqueue. The max_weight saves the greatest weight value in vcpu included in the runqueue. The vcpu in the mcsched_dom structure is a list of vcpu of the domain. In addition, weight is the weight value of the domain and the vcpu of the domain. The nr_vcpus stores the number of vcpu of that domain, and reset_count increases the value each time the vcpu of the domain initializes with credits. The sdom in the mcsched_vcpu structure refers to a domain that contains itself. Here, weight stores the value required for the algorithm used to reduce the number of credits. Credit stores the credit value to be paid when vcpu uses cpu.

Figure 4 shows how to enter the BOOST state after executing the reset_credit() function to initialize credits for all vcpus because a specific vcpu has insufficient credits. Here, sdom is the mcsched_dom structure having vcpu to call the

```
xentop  -  15:14:24     Xen 4.5.0
5 domains: 1 running, 4 blocked, 0 paused, 0 crashed,0 dying, 0 shutdown
Mem: 20664348k total, 5519256k used, 15145092k free            CPUs: 8 @ 3400MHz
      NAME   STATE    CPU (sec)  CPU (%)     MEM (k)  MEM (%)   MAXMEM (k)  MAXMEM (%)  VCPUS
 NETS  NETTX (k)  NETRX (k)  VBDS     VBD OO     VBD RD     VBD WR                VBD WSECT  SS
ID
  Domain-0 -----r           223     1.1      1048576     5.1    no limit         n/a      8
    0         0         0     0         0          0          0           0           0
0
  Domain_1 --b---             9     0.0      1048576     5.1      1049600         5.1      4
    1       521       185     1         0       3179        179       185710        3152
0
  Domain_2 --b---            95    31.7      1048576     5.1      1049600         5.1      2
    1        40         3     1         0       3339        208       185494        5144
0
  Domain_3 --b---           112    37.1      1048576     5.1      1049600         5.1      2
    1        40         3     1         0       3287        208       193718        5224
0
  Domain_4 --b---            48    15.9      1048576     5.1      1049600         5.1      2
    1        35         3     1         0       3282        214       185278        5136
0

Delay   Networks   vBds   Tmem   VCPUs   Repeat  header   Sort  order   Quit
```

Figure 2: Xentop execution screen.

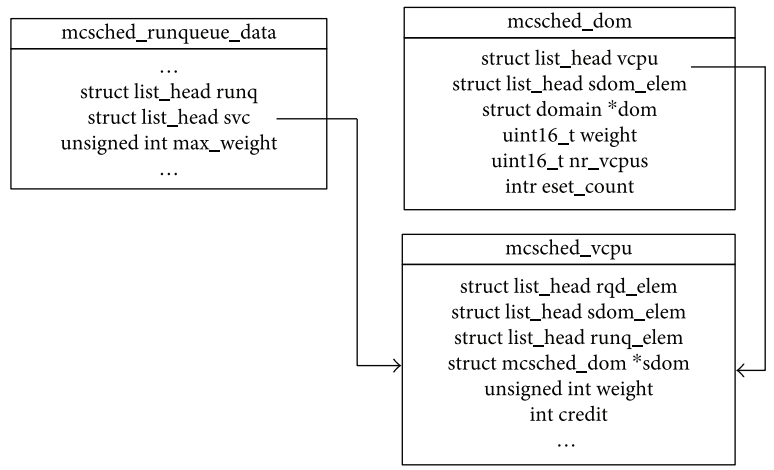

Figure 3: mcsched_runqueue_data, mcsched_dom, and mcsched_vcpu structures.

reset_credit() function owing to insufficient credits. The time at which the reset_credit() function is called and the domain of BOOST state is changed by checking tick variables. A domain in the BOOST state is changed after sufficient time has passed. Otherwise, the value of the reset_count variable of sdom is increased by one. If this reset_count value happens to be the greatest, the domain to be changed to the BOOST state next is reserved as sdom.

Figure 5 shows a flowchart of the process to change the BOOST state domain in the routine for initializing credits. Here, old_d variable is the mcsched_dom structure holding the current BOOST state domain. The d variable is the mcsched_dom structure of the domain to enter the BOOST state. The rate variable saves the increase rate of the current weight value. If the current BOOST state domain and the domain to be changed to BOOST state are the same, the rate increases two times. Otherwise, the weight value is initialized by dividing the current BOOST state domain by the rate. Then, the current BOOST state domain is updated by changing old_d to d after initializing the rate to 2. Finally, multiplying the original weight value of the current BOOST state domain by the rate value allows the weight value to be dynamically changed.

Algorithm 1 shows the process for assigning the BOOST state to a domain and its initialization.

3.2. Virtual Processor Monitoring Tools. The xenstat_domain and xenstat_vcpu structures of Figure 6 are declared in

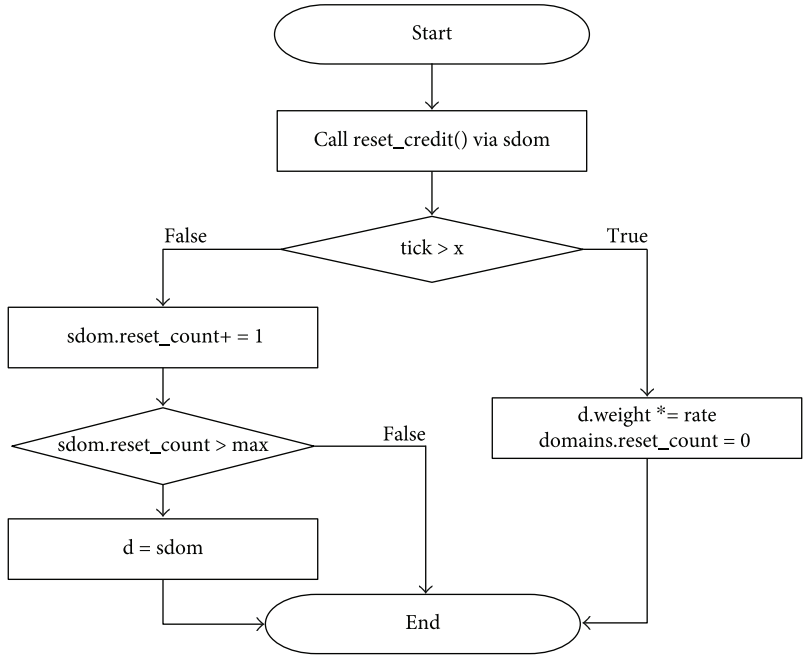

Figure 4: BOOST procedure.

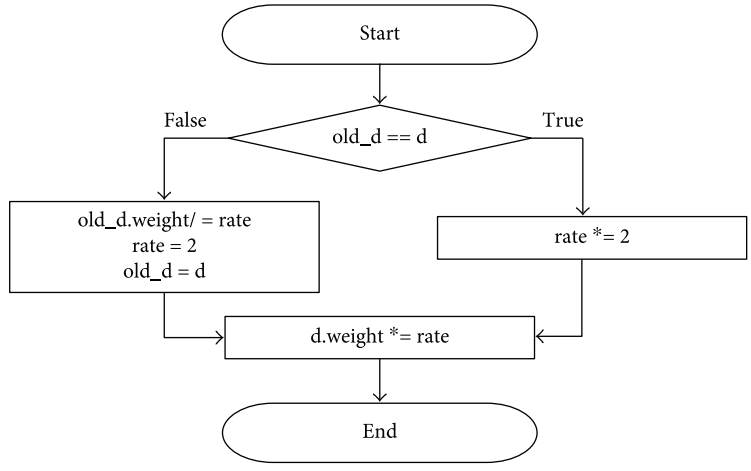

Figure 5: BOOST initialization process.

xenstat_priv.h. cpu_ns in the xenstat_domain structure is a variable that stores the total use time of a domain's processor in nanoseconds. num_vcpus is a variable that stores the number of virtual processors assigned to the domain, and vcpus is a variable with an xenstat_vcpu structure address created using the allocated virtual processors. Here, ns in the xenstat_vcpu structure is a variable that stores the use time of a virtual processor in nanoseconds.

The improved monitoring tools provide the usage of the virtual processors assigned to each domain using the variables described above as percentages. The implementation of the monitoring tools is described as follows.

Essentially, the program saves the xenstat_node structures used previously to calculate the virtual CPU utilization. The time is obtained in microseconds using the previous and the current output time. The use percentage of the virtual processors is obtained using the ns variable of the xenstat_vcpu structure. This process is similar to that shown in Figure 7.

The calculation of virtual CPU utilization (%) is given in Algorithm 2.

Figure 8 shows the complete operation of the proposed virtual processor monitoring tools. The virtual processor utilization for all domains is evaluated using these tools. These tools visually represent the utilization of a virtual CPU for each domain in bar form depending on the utilization specified by each hashtag. The tools also show the average utilization of virtual CPUs.

```
1:  begin
2:    Input: mcsched_dom* snext that calls the reset_credit() function
3:    Output: execution time of the snext->vcpu and snext
4:    // timed wait and enter BOOST state.
5:    if tick > n then
6:      // previous BOOST state: mcsched_dom prev_boost_sdom.
7:      // initialize a weight value
8:      for all vcpus ∈ prev_boost_sdom do
9:        vcpus→weight ← vcpus→weight/rate;
10:     end
11:     prev_boost_sdom→weight ←
                                  prev_boost_sdom→weight/rate;
12:     // future BOOST state: mcsched_dom boost_sdom.
13:     // Increment of the magnification of the weight value.
14:     if prev_boost_sdom = boost_sdom then
15:       rate ← rate*2;
16:       // Initialization of the magnification of the weight value
17:     else
18:       prev_boost_sdom ← boost_sdom;
19:       rate ← 2;
20:     end
21:     // Initialization of the reset_count variables for all domains
22:     for all vcpus ∈ rqd do
23:       svc→sdom→reset_count ← 0;
24:     end
25:     // Increase the weight value for the BOOST state domain.
26:     for all vcpus ∈ boost_sdom do
27:       vcpus→weight ← vcpus→weight*rate;
28:     end
29:     boost_sdom→weight ← boost_sdom→weight/rate;
30:   else
31:     // mcsched_vcpu snext that calls the reset_credit() function.
32:     // Increase the reset_count variable of mcsched_dom using snext
33:     snext→sdom→reset_count ←
                                  snext→sdom→reset_count+1;
34:     // update a mcsched_dom having the highest reset_count value
35:     if snext→sdom→reset_count > max_reset_count then
36:       max_reset_count ← snext_sdom→reset_count;
37:       boost_sdom ← snext→sdom;
38:     end
39:   end
40: end
```

Algorithm 1: BOOST state transition.

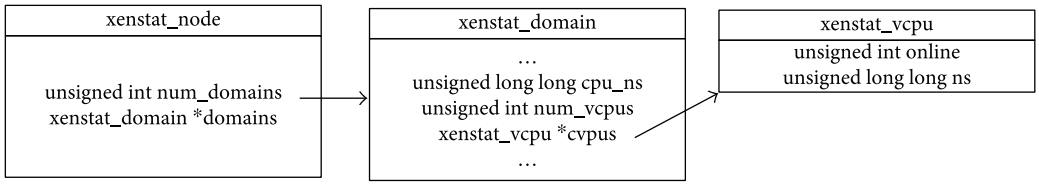

Figure 6: xenstat_node, xenstat_domain, and xenstat_vcpu structures.

4. Performance Evaluation

Here, the performance of the Mcredit2 scheduler is analyzed and compared to extant schedulers.

4.1. Experimental Environments. The experimental environment is as follows: Intel i7-2600 processor (four processors and eight threads) [23], 20 GB RAM, and a 1 TB hard disk. As a result, processor utilization for all domains can be up to 800%. Xen hypervisor version 4.5.0 is used. Ubuntu 14.04 LTS [24] is used as the domain operating system. A total of nine domains are generated. Four virtual processors, 1 GB RAM, and a 40 GB hard disk are assigned to each domain.

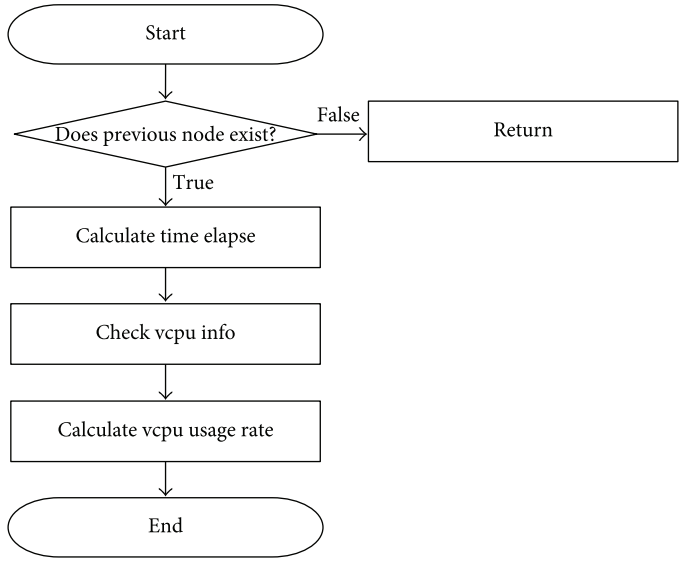

Figure 7: Calculation of virtual CPU utilization (%).

```
1:  begin
2:    Input: xenstat domain d
3:    Output: percentage of the virtual CPU utilization pct
4:    old_domain ← NULL;
5:    us_elapsed ← NULL;
6:    // check if previous node is exist
7:    if prev_node = NULL then
8:      return 0.0;
9:    else if vcpu = NULL then
10:     return 0.0;
11:   else if old_vcpu = NULL then
12:     return 0.0;
13:   end
14:   old_domain ←
      xenstat_node_domain(prev_node, xenstat_domain_id(domain));
15:   if old_domain = NULL then
16:     return 0.0;
17:   end
18:   // calculation of the elapsed time
19:   us_elapsed ←
            (curtime.tv_sec-oldtime.tv_sec)*1000000.0+
            (curtime.tv_usec-oldtime.tv_usec);
20:   // calculate the utilization percentage for the virtual CPU
21:   pct ←
            ((xenstat_vcpu_ns(vcpu)-xenstat_vcpu_ns(old_vcpu))/10.0)
            /us_elapsed;
22:   return pct;
23: end
```

Algorithm 2: Calculation of virtual CPU utilization (%).

The sysbench [25] benchmark tool is used to analyze the scheduler. The sysbench tool measures operating system performance using multithreads. In this experiment, the sysbench cpu mode is used as the test mode. In cpu mode, a request to calculate a decimal value is assigned to the generated thread. Four threads are generated for the sysbench tool. The maximum decimal value is 10,000 and the request to calculate the decimal value occurs 100,000 times. The average results of performing this process 10 times are obtained.

The environment variables for each scheduler are as follows. For the SEDF scheduler, the period and slice parameters of each domain are set to 100 and 10, respectively. For

```
       NAME   STATE    CPU (sec)  CPU (%)  AVR (%)
  Domain_2 --b---             17    30.9     32.7
VCPUS   #0,   15.2%  : [#######                    ]
VCPUS   #1,   15.8%  : [#######                    ]
       NAME   STATE    CPU (sec)  CPU (%)  AVR (%)
  Domain_3 --b---             16    36.9     38.6
VCPUS   #0,   20.8%  : [##########                 ]
VCPUS   #1,   16.1%  : [########                   ]
       NAME   STATE    CPU (sec)  CPU (%)  AVR (%)
  Domain_4 --b---              9    16.6     16.4
VCPUS   #0,    0.0%  : [                           ]
VCPUS   #1,   16.6%  : [########                   ]
```

Figure 8: Monitoring tool for virtual CPUs.

```
1:  begin
2:    Input: the number of threads: n
3:    Output: error number: err_num
4:    // infinite loop
5:    loop
6:      // impose load
7:      for var i < 10000000 do
8:        i ← i+1;
9:      end
10:     // stop to impose
11:   usleep(200000);
12:   end
13: end
```

Algorithm 3: Load test.

the Credit scheduler, the weight value and capacity of each domain are set to 256 and 0, respectively. For the Credit2 scheduler, the weight value of each domain is set to 256. For the proposed Mcredit2 scheduler, the weight value of each domain is set to 256.

The pseudocode of the program used to impose a transient load on each domain is shown in Algorithm 3.

In these experiments, the pseudocode threads are generated in the domain not to execute the sysbench benchmark tool. The domain that executes the sysbench benchmark is called Domain_1, and the other domains are named Domain_2 to Domain_9. In the first experiment, four threads are generated with sufficient idle processors, and in the second experiment, eight threads are generated with insufficient idle processors.

4.2. Experimental Results. Figures 9 and 10 show the total time required by the sysbench benchmark tool to measure performance and the idle processors generated during the process. Figure 9 shows the performance results with sufficient idle processors, and Figure 10 shows the results with insufficient idle processors.

The SEDF scheduler demonstrates the worst performance. It cannot give priority to Domain_1, which requires many virtual processors because the deadlines of all domains are the same.

For the Credit scheduler, even when a specific domain runs out of credits, if it has idle processors, the processors can be used without redistributing credits. This leads to increased virtual processor utilization of domains other than Domain_1. In addition, the Credit scheduler cannot use processors until credits are recharged when all credits are used. As a result, additional processors are not assigned to Domain_1 if there are no idle processors, and the performance results for Domain_1 are not as good as those of Credit2 and Mcredit2.

For the Credit2 scheduler, fewer processors are assigned to Domain_1 even if idle processors are sufficient. As a result, it shows poorer performance than the Credit scheduler relative to utilizing more idle processors.

The Mcredit2 scheduler can use more processors with the same number of credits because it dynamically increases the weight value in domains with significant load. If there are idle processors, processors can be continuously supplied due to the high weight value of Domain_1. Due to the greater weight value of Domain_1, it gains higher processor priority than the other domains and its performance is increased.

5. Conclusion

Hypervisors have been developed to take advantage of the increased performance of server hardware, and scheduling algorithms have also been improved, so that hypervisors can efficiently assign processors to virtual machines. As described in this paper, the Credit2 scheduler algorithm of the Xen hypervisor has been improved. In the proposed method, a domain that requires many processors under the Credit2 scheduler is identified, and the Mcredit2 scheduler gives higher priority to this domain by increasing its weight value.

The performance of the SEDF, Credit, and Credit2 schedulers were compared to that of the proposed Mcredit2 scheduler. The performance was measured using the sysbench benchmark tool. The SEDF scheduler shows significantly worse performance than the other schedulers, and the Credit scheduler uses idle processors. Therefore, higher virtual CPU utilization of domains can be accomplished with lower domain workload. With the Credit2 scheduler, when a load occurs in a specific domain, more virtual processors are assigned to that domain than with the Credit scheduler. With the proposed Mcredit2 scheduler, a domain where load occurs is assigned more processors. The experimental results demonstrate that the Mcredit2 scheduler assigns virtual

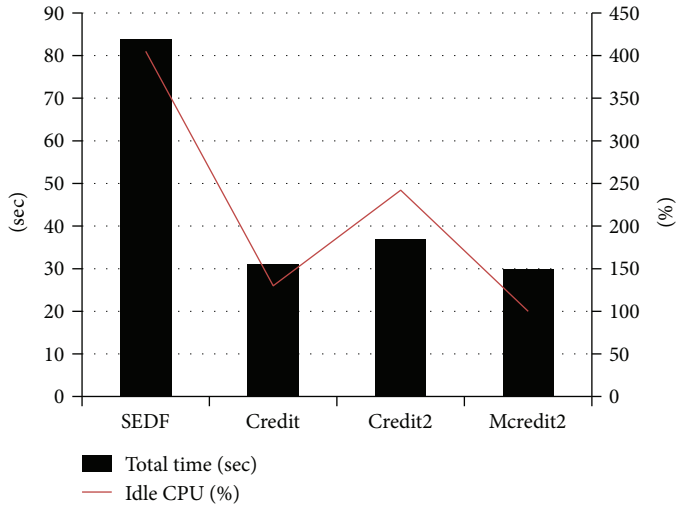

Figure 9: Results with sufficient idle CPUs.

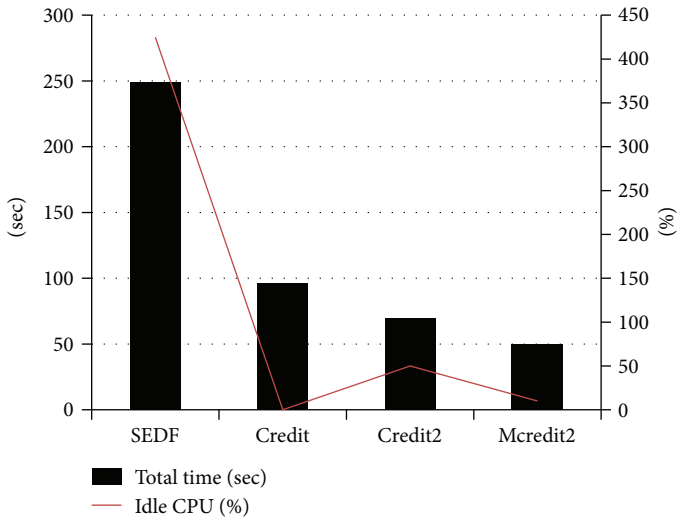

Figure 10: Results with insufficient idle CPUs.

processors to loaded domains more flexibly than the Credit2 scheduler. The experimental results also show that the Mcredit2 scheduler assigns processors more flexibly than the Credit2 scheduler when processor requirements for a specific domain significantly increase.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Korea Government (MSIP) (2015R1D1A1A01057680).

References

[1] A. P. Plageras, K. E. Psannis, C. Stergiou, H. Wang, and B. B. Gupta, "Efficient IoT-based sensor BIG Data collection–processing and analysis in smart buildings," *Future Generation Computer Systems*, vol. 82, pp. 349–357, 2018.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): a vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[3] November 2017, https://www.ibm.com/us-en/.

[4] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, "Virtualization for high-performance computing," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 2, pp. 8–11, 2006.

[5] L. Van Doorn, "Hardware virtualization trends," in *Proceedings of the 2nd international conference on Virtual execution environments '06*, vol. 14no. 16, ACM/Usenix, p. 45, New York, NY, USA, 2006.

[6] M. Steinder, I. Whalley, D. Carrera, I. Gaweda, and D. Chess, "Server virtualization in autonomic management of heterogeneous workloads," in *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*, pp. 139–148, Munich, Germany, 2007.

[7] J. Daniels, "Server virtualization architecture and implementation," *Crossroads*, vol. 16, no. 1, pp. 8–12, 2009.

[8] R. P. Goldberg, "Survey of virtual machine research," *Computer*, vol. 7, no. 6, pp. 34–45, 1974.

[9] M. Rosenblum and M. Garfinkel, "Virtual machine monitors: current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.

[10] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," *Proceedings of the Linux Symposium*, vol. 1, 2007.

[11] P. Barham, B. Dragovic, K. Fraser et al., "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[12] November 2017, https://wiki.xenproject.org/wiki/Xen_Project.

[13] D. Chisnall, *The Definitive Guide to the Xen Hypervisor*, Pearson Education, Upper Saddle River, NJ, USA, 2008.

[14] November 2017, https://wiki.xenproject.org/wiki/Credit_Scheduler.

[15] November 2017, https://wiki.xen.org/wiki/Credit2_Scheduler_Development.

[16] November 2017, https://wiki.xenproject.org/wiki/RTDS-Based-Scheduler.

[17] November 2017, https://wiki.xenproject.org/wiki/ARINC653_Scheduler.

[18] D. Watts and I. Krutov, *System x3950 X6 (3837)*, lenovopress, China, 2015.

[19] M. Kang, *Mcredit2: Enhanced Xen Scheduler for High Performance via Dynamic Weight Allocation [M.S. thesis]*, Soongsil University, Republic of Korea, 2016.

[20] November 2017, https://wiki.xenproject.org/wiki/Xentop.

[21] G. Lipari, *Earliest Deadline First*, Scuola Superiore Sant'Anna, Pisa-Italy, 2005.

[22] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three CPU schedulers in Xen," *SIGMETRICS Performance Evaluation Review*, vol. 35, no. 2, pp. 42–51, 2007.

[23] December, http://ark.intel.com/products/52213/Intel-Core-i7-2600-Processor-8M–Cache-up-to-3_80-GHz.

[24] November 2017, https://wiki.ubuntu.com/TrustyTahr/ReleaseNotes.

[25] A. Kopytov, *SysBench Manual*, MySQL AB, Sweden, 2012.