

A Modified Priority Based CPU Scheduling Scheme for Virtualized Environment

Chia-Ying Tseng* and Kang-Yuan Liu

*Department of Computer Science and Engineering, Tatung University
#40, Sec. 3, Zhongshan N. Rd., Taipei, Taiwan
cytseng@ttu.edu.tw

Abstract

In order to maximum the hardware resource utilization, Virtual Machine (VM) management has become an important research field of virtualization technology. VM scheduling is crucial for the throughput of a system and thus affects the overall system performance. According to the scheduling algorithm of Credit Scheduler using in XEN hypervisor, each virtual CPU is asynchronously assigned to a physical CPU by CPU scheduler in order to maximize the throughput. But for a concurrent program, the implementations of threads are not completely independent. In this paper, a solution is proposed for the synchronization problem of a concurrent program. By modifying the Xen Credit Scheduler, threads in the concurrent program can be synchronized and the performance in concurrent workload is greatly enhanced. The waste of CPU time can be reduced and achieve higher throughput while keeping the design of the scheduler to be light weight, fair, and efficient. With the proposed scheduler, more VMs can be deployed while achieving the same throughput, thus gain greater utilization of resources and better energy efficiency.

Keywords: *Virtual Machine, Xen-hypervisor, Virtual CPU scheduling*

1. Introduction

The traditional technique for reducing the probability of service interruption is to run each critical application on a separate physical server. In this way, if a hardware component or this critical application fails, only one application is affected. When running multiple applications on a single physical server, if one application fails it can cause the operating system and other applications to react in unpredictable ways; some applications require specific hardware whose drivers might as well generate unpredictable errors with some other applications and the hardware component failing may cause the whole host becomes unavailable. But this “one server, one application” model leaves most machines vastly underutilized and makes the servers idle most of the time. This model is based on the thought to ensure the isolation between each application to prevent applications from interfering with each other; it is similar to the concept of Virtual Machine (VM). The practice of partitioning a single server so that it appears as multiple servers has reduced space utilization, the costs of hardware acquisition and energy consumption. In order to maximum the utilization of the hardware resources and minimize the capital costs by reducing physical infrastructure, VM management has become an important research field of virtualization technology, and the scheduling of virtual machines on a physical host machine is crucial for the throughput of a system and thus affects the overall system performance.

In order to maximum the utilization of the hardware resources and minimize the capital

costs by reducing physical infrastructure, Virtual machine management has become an important research field of virtualization technology, and the scheduling of virtual machines on a physical host machine is crucial for the throughput of a system and thus affects the overall system performance.

Virtual Machine scheduling is crucial for the throughput of a system and thus affects the overall system performance. According to the scheduling algorithm of Credit Scheduler, each virtual CPU is asynchronously assigned to a physical CPU by CPU scheduler in order to maximize the throughput. But for a concurrent program, the implementations of threads are not completely independent. In this paper, a solution is proposed for the synchronization problem of a concurrent program. By modifying the Xen Credit Scheduler, a new priority TURBO is added to the proposed scheduler to avoid the scheduling decisions that was made for synchronization to be modified by awaking Virtual CPUs (VCPUs) or the load balancing algorithm. This allows need-to-sync VCPUs to preempt and being picked up to run at the next time slice without impacting overall system fairness; hence the threads in the concurrent program can be synchronized. The waste of CPU time can be reduced while keeping the design of the scheduler to be light weight, fair and efficient, thus gain greater utilization of resources and better throughput. The more Virtual Machines are deployed in a physical host, the less physical hosts are required; hence the hardware acquisition costs and maintenance costs can be reduced.

2. Virtual CPU Scheduling Strategy in Xen

Xen originated as a research project at the University of Cambridge and was presented at ACM symposium on Operating systems principles [1]. It is an x86 virtual machine monitor which allows multiple OSes to share one single hardware in a safe and resource managed fashion, but without sacrificing either performance or functionality. The design is targeted at hosting up to 100 virtual machine instances simultaneously on a modern server. Xen uses a form of virtualization known as para-virtualization, created by the founders of the Xen hypervisor project. By using this technology the VMs and hypervisor can co-operate with the performance overhead typically in the range of 0.1% to 3.5% and 8% in worst case for industry standard performance benchmarks and achieve very high performance for I/O, CPU, and memory virtualization. The performance overhead is extremely low compared with full virtualization, which can be as much as 20%, or more. But para-virtualization requires porting of the guest OSes. HVM such as Intel VT and AMD-V offers new instructions to support direct calls to the VMM, enabling original guest OSes to run within Xen VMs, starting with Xen 3.0.

The overall system structure is shown in Figure 1. A privileged domain, call Domain0 (Dom0), is created at boot time which is permitted to use the control interface to create and terminate other user domains (DomU) and to control their associated scheduling parameters, physical memory allocation and the access they are given to the machine's physical disks and network devices. Xen supports up to 64-way symmetric multi-processing (SMP) machines and a DomU can be configured as either a uni-processor system or an SMP system in the VM environment of full virtualization or paravirtualization running unmodified user software.

Three different CPU schedulers were introduced in Xen: Borrowed Virtual Time (BVT) [2], Simple Earliest Deadline First (SEDF) [3], and the Credit Scheduler [4]. Credit Scheduler is a non-preemptive fair-share scheduler maintained by weight and cap, weight decides the amount of CPU time it can get and cap fixes the maximum amount of CPU a domain will be able to consume. Each CPU manages a local run queue of runnable VCPUs which is sorted by VCPU priority rather than credit. Generally, the next VCPU to run is

picked off the head of the run queue. Credit Scheduler supports for both WC and NWC modes with global load balancing; hence it is set to be the default scheduler in Xen.

Credit Scheduler is designed to be light weight and efficient, each PCPU manages a local run queue of runnable VCPUs. This run queue is sorted by VCPU priority rather than credit. On each PCPU, no matter what scheduling decision is made during the common path of the scheduler, the next VCPU to run is picked off the head of the run queue. Once a VCPU is scheduled, it receives the time slice of 30ms. As a VCPU runs, it consumes credits of 100 every 10ms. When the time slices of a running VCPU expires, if this VCPU is still can be running, it will be put after all other VCPUs of equal priority to it.

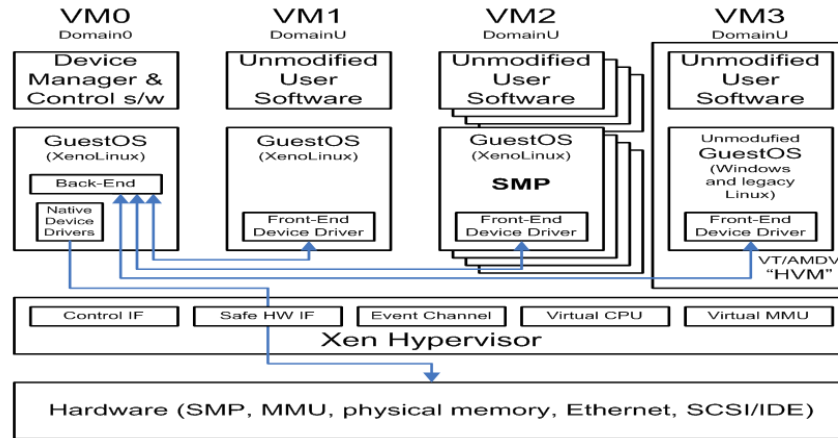


Figure 1. System Structure of Xen

An active VCPU's priority can be one of two values: OVER (-2) or UNDER (-1) representing whether this VCPU has or hasn't yet exceeded its fair share of CPU resource in the ongoing accounting period (30ms). When a non-active VCPU is woken, it is set to priority BOOST, which allows it to be put at the head of the run queue and run ahead of any UNDERs or OVERs. This will make it more effective at scheduling latency-sensitive VMs.

Every credit period (30ms), a system-wide accounting thread re-computes how many credits each active VM has earned and bumps the credits; if the accounting master has modified priorities, a special O(n) sort of the run queue sort and runs at most once per accounting period (30ms). The optimized sort will walk through the run queue and move up any UNDERs that are preceded by OVERs.

When a CPU couldn't find a VCPU with the priority higher than OVER on its local run queue it will look on other PCPUs for one, starting with its immediate neighbor. If it failed to find more important work to steal, the next VCPU to run is picked off the head of the local run queue. This load balancing guarantees each VM receives its fair share of CPU resources system-wide.

If no runnable VCPU is going to be inserted into the run queue of a PCPU then this PCPU is going to idle. Before a PCPU goes idle, it will look at other PCPUs to find any runnable VCPU. This guarantees that no PCPU idles when there is runnable work in the system.

For Credit Scheduler, each VCPU is asynchronously assigned to a PCPU in order to maximize the throughput while guaranteeing the CPU time fairness according to the weight. However, several studies [5, 6, 7] have shown that this strategy may cause CPU allocation errors and result in considerable waste of CPU time for concurrent workloads.

Assuming that this VM has the Credit that gives it three time slices of CPU time from next nine time slices, but there is a synchronization operation between threads at the end of each step, and the length of the step is equal to the length of the time slice. So the second step will be blocked until each VCPU has finished the work of the first step, the third step will be blocked until each VCPU has finished the work of the second step and so on. The multithreaded application only completes two steps in the length of nine slots; while there are four slots of CPU time is being wasted for the synchronization, as shown in Figure 2.

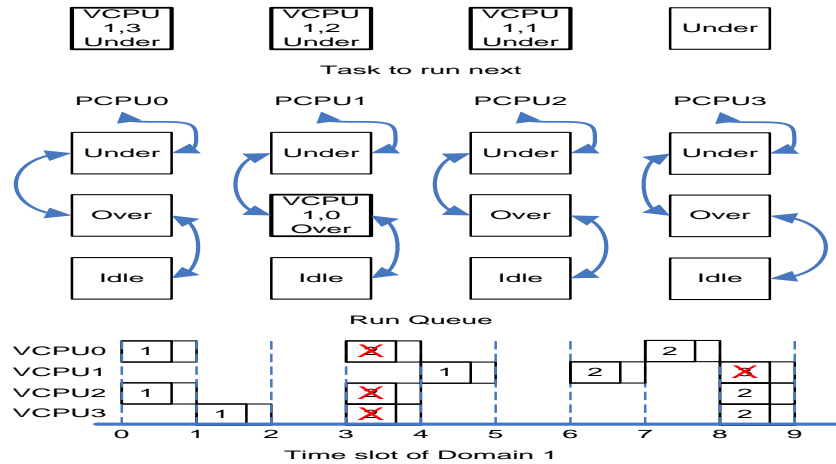


Figure 2. Scenario of the Credit Scheduler Scheduling Sequence in Xen

3. The Design of Priority based CPU Scheduling Algorithm

While an x86 symmetrical multi-processing (SMP) system is booting, it will assign a processor which monitors the operation of the multiprocessor system. This master processor is called a Bootstrap Processor (BSP) while all other processors are called Application Processors (AP). It is the same for a VM since it is a duplication of a real machine. In previous section the performance drop of a concurrent program running credit scheduler was discussed which is due to asynchronous scheduling and the communication or synchronization between threads. BSP is the master processor; hence in a virtualization environment communications and synchronizations are handled by the Virtual Bootstrap Processor (VBSP). So if a VBSP is picked up to run, all non-idling Virtual Application Processors (VAP) in the same domain must be picked up to run at the same time slice for synchronization purpose. Since one VCPU can run only one at a time on a PCPU, all VCPUs must be picked up by different PCPUs and run. So the proposed scheduler must achieve two goals for synchronization. First, pick up all the non-idling VAPs in the need-to-sync domain when their VBSP is picked up. Secondly, assign each VCPUs to run on different PCPU in the same time slice.

A new priority TURBO, which is higher than the BOOST, UNDER, OVER and IDLE is added to the proposed algorithm to avoid the scheduling decisions that was made for synchronization to be modified by awaking VCPUs or the load balancing algorithm. This allows need-to-sync VCPUs to preempt and being picked up to run at the next time slice without impacting overall system fairness. A global variable turbo_domain is defined to keep track of the latest turbo domain setting that will be used in the stage of run queue sorting. The pseudo code of the proposed scheduler is shown in Figure 3.

```
Algorithm csched_schedule (now);  
Input: now (the current time).  
Output: ret (task to run next).  
BEGIN {Check the VCPU that is about to end its time slice}  
    IF current running VCPU is runnable THEN  
        Insert it to local run queue;  
    ELSE  
        Current VCPU is idle or local run queue is empty;  
    END IF  
    {Select next runnable VCPU from local run queue}  
    Get the task from the top of local run queue;  
    IF the VCPU's priority is TURBO, OR no domain is set to be the turbo  
    domain and the VCPU hasn't eaten through its credits  
    THEN  
        Remove the task from local run queue;  
    ELSE {See if there is more urgent task on other PCPU}  
        IF there is more urgent work on other PCPUs THEN  
            Get this more urgent work;  
        ELSE  
            Get the task from the top of local run queue;  
        END IF  
    END IF  
    {Update idlers mask if necessary}  
    IF the VCPU is IDLE THEN  
        IF idlers mask hasn't been updated THEN  
            Update idlers mask;  
        END IF  
    ELSE  
        IF the PCPU was idling THEN  
            Clear it from idlers mask;  
        END IF  
    END IF  
    {SET the time slice}  
    IF it's an IDLE VCPU THEN  
        Set illegal time slice (-1);  
    ELSE  
        Set the time slice to be CSCHED_MSECS_PER_TSLICE;  
    END IF  
    Set task to run next;  
    {Clear turbo domain setting if necessary}  
    IF TURBO DOMAIN is set and task to run next is the VBSP of  
    TURBO DOMAIN  
        THEN CLEAR the TURBO DOMAIN setting;  
    END IF  
    {Set a turbo domain if necessary}  
    Get the next member from local run queue;  
    IF it is not the end of local run queue THEN  
        Get VCPU from the top of local run queue;  
    IF it's a non-idle VBSP and no domain has been set to be the TRUBO  
        THEN Set TRUBO DOMAIN;  
        Inform each PCPU that its run queue needs to be sorted;  
    END IF  
    END IF  
    RETURN task to run next;  
END
```

Figure 3. Algorithm of the Scheduling Sequence

An $O(n)$ optimized sort of the run queue walks through the run queue and boost all non-idle VCPUs in turbo domain to the priority of TURBO, then it moves the TURBOs to the head of the run queue and move up any UNDERS that are preceded by OVERS with the following algorithm (Figure 4).

```
Algorithm csched_runq_sort (cpu);  
Input: cpu (PCPU ID).  
  
BEGIN  
  {Check whether it is necessary to sort the run queue}  
  IF we've served all sorting requests THEN  
    No need to sort and exit;  
  END IF  
  Claim that all sorting requests have been served;  
  Spin lock for protection;  
  {Sort the run queue}  
  WHILE it's not the end of local run queue  
    {Boost all active VCPU in TURBO DOMAIN}  
    IF it's not an idle VCPU, TURBO DOMAIN is set and it's  
    a member of the TURBO DOMAIN THEN  
      Set its priority to TURBO;  
    END IF  
    IF it's a TURBO VCPU THEN  
      IF it's the first VCPU of run queue THEN  
        Set it to be the last-known-under;  
      ELSE  
        {Move TURBOs up if necessary}  
        IF this TURBO is not behind a TURBO THEN  
          Remove it from run queue;  
          Insert it behind last-known-turbo;  
        END IF  
      END IF  
      Set it to be the last-known-turbo;  
    ELSE  
      IF VCPU's priority is UNDER or BOOST THEN  
        IF it's hot behind last-known-under THEN  
          Remove it from run queue;  
          Insert it behind last-known-under;  
        END IF  
        Set it to be the last-known-under;  
      END IF  
    END IF  
  END WHILE  
  Spin unlock;  
END
```

Figure 4. Algorithm of the Run Queue Sort

If this VCPU's priority was boosted it will be reset during the execution to avoid it consuming a non-negligible amount of CPU resources. If the turbo domain is dead and is no longer returning to run queue, the system-wide accounting thread will know it and reset the turbo domain setting.

4. Performance Evaluation

In this section, the necessity of application isolation is verified by comparing the performance between a native machine without virtualization environment installed and the "one VM, one application" model using Xen Credit Scheduler. Then the experiments to

compare the performance of parallel workloads and concurrent workloads using both Credit Scheduler and the Modified Credit Scheduler are performed.

4.1 Experimental Environment

The experimental hardware platform is an IBM Compatible PC, with a 2.66GHz Intel Core i7-920 processor, both Turbo Boost and Hyper-Threading is disabled. The system is installed with 12GB DDR III 1066 MHz RAM and 5x500GB RAID6 7200 RPM SATA II hard disk. The host OS in Dom0 is CentOS 5.4 with kernel 2.6.18 and Xen Hypervisor 3.4.2 as the VMM. The guest OS in DomU is CentOS 5.4 with kernel 2.6.18, the memory allocated for VMs is set to 512MB and the hard disk space is set to 4GB

4.2 Benchmark

The NAS Parallel Benchmarks (NPB) are developed and maintained by the NASA Advanced Supercomputing (NAS) Division, NPB 1.0 was developed in 1991 [8] and released in 1992 with benchmarks such as Embarrassingly Parallel (EP), MultiGrid (MG), Conjugate Gradient (CG), Fast Fourier Transform (FT), Integer Sort (IS), Block Tridiagonal (BT), Scalar Pentadiagonal (SP) and Lower-Upper symmetric Gauss-Seidel (LU). The first five are the parallel kernel benchmarks, and the last three are the simulated application benchmarks. NPB is a set of benchmarks targeting performance evaluation of highly parallel supercomputers for the key characteristics of typical processing in computational aerodynamics; it specifies five problem sizes for each benchmark - smallest class "S", class "W", class "A", class "B" and the largest class "C". The following gives an overview of the two benchmarks which was selected for the experiments.

EP: An "embarrassingly parallel" kernel, numerous separate segments of a single and reproducible sequence can be generated on separate processor cores of a multi-core system. This kernel, in contrast to other parallel kernel benchmarks in the list, requires virtually no inter-core communication.

LU: A regular-sparse, block (5x5) lower and upper triangular system solution. Communication of partition boundary data occurs after completion of computation on all diagonals that contact an adjacent partition. This constitutes a diagonal pipelining method and is called a "wavefront" method [9].

4.3 Experiments and Results

Two benchmarks are selected for the performance evaluation of parallel workload and concurrent workload, EP and LU. All processors work separately with virtually no inter-processor communication in EP which represents the benchmark for evaluating parallel workload, while the concurrent workload is evaluated by LU which is a small message communication sensitive benchmark.

The experiments start with the performance evaluation between the "one VM, one application" model using Credit Scheduler and running multiple benchmarks in a single non-virtualized environment using Linux process scheduler.

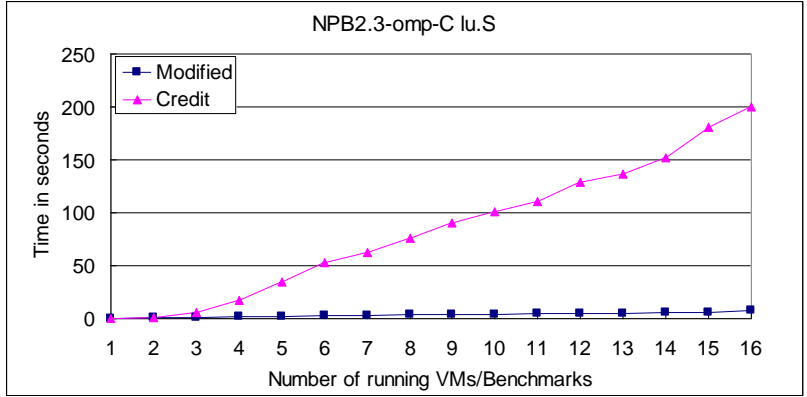


Figure 5. lu.S Benchmark on Xen with Credit Scheduler and the Proposed Scheduler

As shown in Figure 5, the wasted CPU time gains much faster with Credit Scheduler especially when there are more than two active domains since there's a better chance for two domains to get the PCPUs by turns. The average execution time for the proposed scheduler of 15 active domains running LU simultaneously is only 1% higher than average execution time for the Credit Scheduler of 3 active domains running LU simultaneously. It is worthwhile trading 1% more execution time for 5 times more throughput.

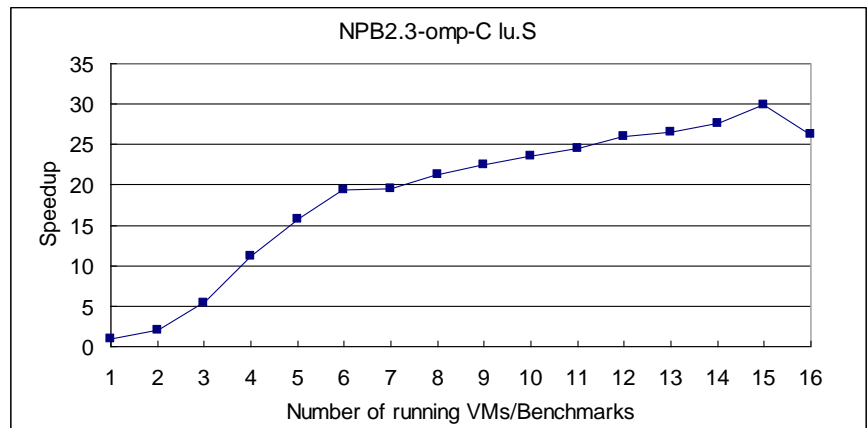


Figure 6. Speed up of Proposed Scheduler with lu.S

As Figure 6 shows, with the scheduler proposed, the performance is greatly enhanced within 6 VMs compared with Credit Scheduler, the default scheduler of Xen. When running 7 to 15 VMs in the experimental environment, the speed up grows slower, approximately 3 times slower than running 1 to 6 VMs. The speedup starts dropping with 16 active domains running one LU each; it shows that the CPU allocation error rate rises too fast while running 16 VMs and the proposed scheduler cannot handle more than 15 active domains.

For a parallel workload, such as EP the proposed scheduler posed a minor performance drop, as shown in Figure 7.

It can be observed in Figure 8 that the performance drop of the proposed scheduler, except only 1 VM's running, is within 3% and can be regarded as the scheduling overhead, mainly on finding more urgent work on other PCPUs. The implement of synchronization in the scheduler is to set a turbo domain and then pick up all runnable VCPUs of turbo domain from all PCPU run queues and run, clearly the defect of the scheduler is the typical range of picking up a job extends from one run queue to all run queues. The speedup is nearly 1.07 with only 1 active domain which takes the advantage of continuous boosting of VCPU's priority. The only active DomU in the system with Computational Intensive Workload (CIW) burns out credit quickly, thus the priority goes OVER very often. OVERs need to find more urgent work on other PCPUs for load balancing and find nothing but idle task. For the proposed scheduler, most of the time the priority of VCPUs are TURBO in the one and only active DomU, it's not necessary to find more urgent works from other PCPUs; hence the scheduling overhead is decreased and shows better performance with only one active Domain running.

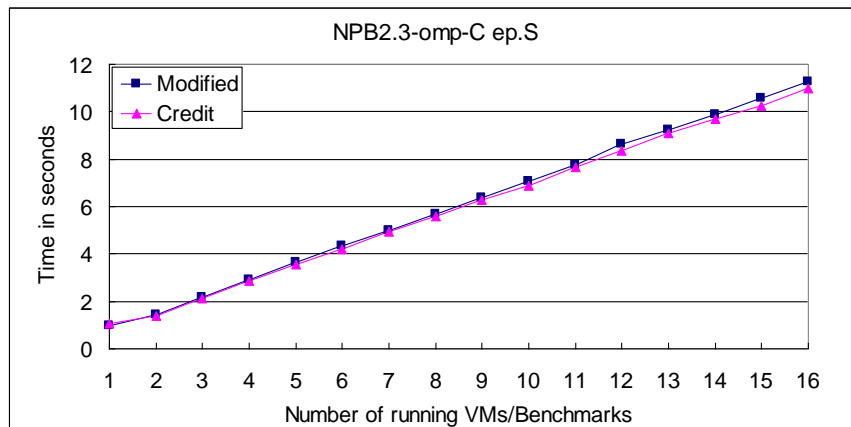


Figure 7. ep.S Benchmark on Xen with Credit Scheduler and the Proposed Scheduler

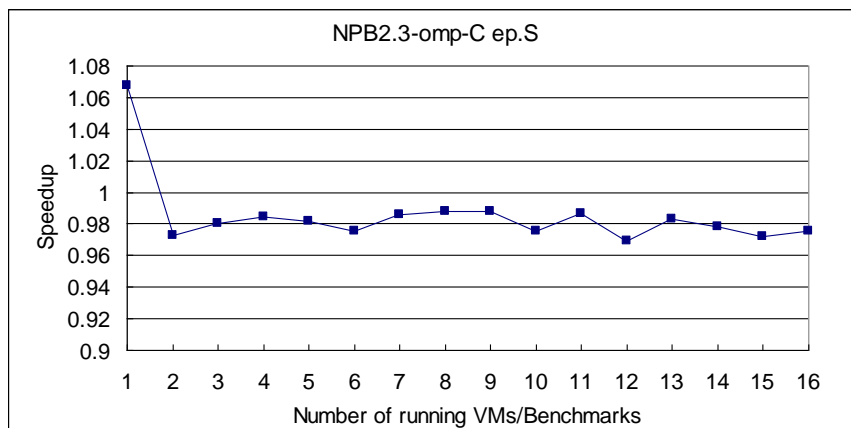


Figure 8. Speed up of Proposed Scheduler with ep.S

The proposed scheduler gains 3% more scheduling overhead compared with Credit Scheduler. For concurrent workloads the performance drop over asynchronous scheduling pose a great threat then the extra scheduling overhead. But for parallel workloads it gains nothing but the execution time. The experiment result suggests that different scheduling strategies should be used for different workloads in order to maximize the throughput.

5. Conclusion

In this paper, a solution is proposed for the synchronization problem of a concurrent program. By modifying the Xen Credit Scheduler, a new priority TURBO is added to the proposed scheduler to avoid the scheduling decisions that was made for synchronization to be modified by awaking VCPUs or the load balancing algorithm. This allows need-to-sync VCPUs to preempt and being picked up to run at the next time slice without impacting overall system fairness. If a VBSP is going to be picked up to run in the scheduling decision, all non-idling VAP in the same domain must be picked up to run at the same time slice for synchronization purpose; hence if there's currently no turbo domain existed the domain of this VBSP will become the turbo domain. All VCPUs' priority will be set to TURBO in turbo domain and resort the run queue. This allows need-to-sync VCPUs to preempt and being picked up to run at the next time slice without impacting overall system fairness; hence the threads in the concurrent program can be synchronized, and the waste of CPU time can be reduced while keeping the design of the scheduler to be light weight, fair and efficient.

The CPU scheduler proposed in this paper does not fit all circumstances, it works fine and greatly enhances the performance in concurrent workload by degreasing CPU allocation errors; but it incurs minor performance drop in parallel workload due to the extra overhead of finding the most urgent work from other PCPUs.

CPU scheduling strategies are crucial for application performance, so different scheduling strategies should be used for different types of application in order to reach the maximum throughput. But even the characteristic of application workload might change for a single application. If an application has both concurrent workloads, parallel workload and sequential workload in different phase of execution, using the scheduler for particular workload may not be the best choice. Different scheduling strategies should be used according to different application workload; hence a self adapting scheduling strategy management is needed. Using Dom0 to analyze current application workload and change the scheduling strategy of each DomU accordingly might be one of the directions for future works.

References

- [1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, "Xen and the art of virtualization", 19th ACM symposium on Operating systems principles, Bolton Landing, NY, USA, (2003), pp. 164-177.
- [2] K. J. Duda and D. R. Cheriton, "Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler", ACM SIGOPS Operating Systems Review, vol. 33, no. 5, (1999), pp. 261-276.
- [3] S. Diestelhorst, "Scheduling Operating-Systems", Großer Beleg, Technische Universität Dresden, Dresden, Germany, (2007).
- [4] E. Ackaouy, "The Xen Credit CPU Scheduler, XenSource: Open Source Enterprise Virtualization", http://www.xen.org/files/summit_3/sched.pdf, (2006) September 18.
- [5] L. Cherkasova, D. Gupta and A. Vahdat, "Comparison of the three CPU schedulers in Xen", ACM SIGMETRICS Performance Evaluation Review, vol. 35, no. 2, New York, NY, USA, (2007), pp. 42-51.
- [6] C. Xu, Y. B. Bai and C. Luo, "Performance Evaluation of Parallel Programming in Virtual Machine

- Environment”, Proceedings of 6th IFIP International Conference on Network and Parallel Computing, Gold Coast, QLD, (2009) October 19-21, pp. 140-147.
- [7] X. H. Xu, P. P. Shan, J. Wan and Y. C. Jiang, “Performance Evaluation of the CPU Scheduler in XEN”, Proceedings of International Symposium on Information Science and Engineering, Shanghai, (2008) December 20-22, pp. 68-72.
- [8] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnam and S. K. Weeratunga, “The NAS Parallel Benchmarks”, International Journal of Supercomputer Applications, vol. 5, no. 3, (1991), pp. 63-73.
- [9] E. Barszcz, R. Fatoohi, V. Venkatakrishnan and S. Weeratunga, “Solution of Regular, Sparse Triangular Linear Systems on Vector and Distributed-Memory Multiprocessors”, Technical Report NAS RNR-93-007, NASA Ames Research Center, Moffett Field, CA, 94035-1000, (1993).
- [10] G. J. Popek and R. P. Goldberg, “Formal requirements for virtualizable third generation architectures”, Communications of the ACM, vol. 17, no. 7, (1974) July, pp. 412-421.
- [11] H. Jin, M. Frumkin and J. Yan, “The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance”, NAS Technical Report NAS-99-011, NASA Ames Research Center, Moffett Field, CA, (1999) October.
- [12] J. H. Che, Q. M. He, Q. H. Gao and D. W. Huang, “Performance Measuring and Comparing of Virtual Machine Monitors”, IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, Shanghai, (2008), pp. 381-386.

Authors



Chia-Ying Tseng received a bachelor degree in industrial education from Taiwan Normal University in 1979 and a master degree in computer engineering from the Electrical and Computer Engineering Department of Arizona State University, USA, in 1985. In 1990, he was selected by Ministry of Education to enter the PhD program in the Computer Science and Engineering Department of Arizona State University for one year research. Currently, he is an Assistant Professor in the Computer Science and Engineering Department of Tatung University, Taiwan. His research interests include Embedded System Design, Embedded Real Time Operating System, Multiprocessor System-on-Chip (MPSoC), Cloud Computing and Virtualization.



Kang-Yuan Liu received the B.S. degree in Chemical Engineering from National Taiwan University of Science and Technology in 2002, and his M.S. degree in Computer Science and Engineering from Tatung University in 2004. Currently, he is the Ph.D. candidate in Tatung University. His research interests include video compression, real-time operating system scheduling, and embedded system programming.

