View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Warwick Research Archives Portal Repository

THE UNIVERSITY OF WARWICK

Original citation:
Shao, Z., Wang, Q., Xuejiao, X., Jin, H. and He, Ligang (2011) Analyzing and improving MPI communication performance in overcommitted virtualized systems. In: 19th Annual Meeting of the IEEE Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'11), Singapore, 25-27 July 2011. Published in: 2011 IEEE 19th International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS) pp. 381-389.

Permanent WRAP url:
http://wrap.warwick.ac.uk/45691

Copyright and reuse:
The Warwick Research Archive Portal (WRAP) makes this work by researchers of the University of Warwick available open access under the following conditions. Copyright © and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable the material made available in WRAP has been checked for eligibility before being made available.

Copies of full items can be used for personal research or study, educational, or not-for profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

Copyright statement:
"© 2011 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting /republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works."

A note on versions:
The version presented here may differ from the published version or, version of record, if you wish to cite this item you are advised to consult the publisher's version. Please see the 'permanent WRAP url' above for details on accessing the published version and note that access may require a subscription.
For more information, please contact the WRAP Team at: publications@warwick.ac.uk

warwickpublicationswrap

highlight your research

http://wrap.warwick.ac.uk

Analyzing and Improving MPI Communication Performance in Overcommitted Virtualized Systems

Zhiyuan Shao, Qiang Wang, Xuejiao Xie and Hai Jin
Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, 430074, China
Email: zyshao@hust.edu.cn

Ligang He
Department of Computer Science
University of Warwick
Coventry, CV4 7AL, United Kingdom
Email: liganghe@dcs.warwick.ac.uk

***Abstract*—Nowadays, it is an important trend in the system domain to use the software-based virtualization technology to build the execution environments (e.g., Clouds) and serve high performance computing (HPC) applications. However, with the extra virtualization layer, the application performance may be negatively affected. Studies revealed that the communication performance of the MPI library, which is widely used by the HPC applications, would suffer a high penalty when a physical host machine becomes overcommitted by virtual processors (VCPU). Unfortunately, the problem has not received enough attention and has not been solved yet in literature. In this paper, we investigate the reasons behind the performance penalty, and propose a solution to improve the communication performance of running MPI applications in the overcommitted virtualized systems. The experimental results show that by our proposal, most HPC applications can gain performance improvement to different extents among the overcommitted systems, depending on their communication patterns and the overcommitting level.**

***Keywords*-virtualization; cloud; MPI; performance;**

I. Introduction

It is an important trend nowadays to use virtualization technologies to build execution environments (e.g., Clouds [1]) and serve scientific applications [2, 3]. Among these virtualization technologies, the Xen hypervisor [4], which allow users to execute hundreds of virtual machines on a single physical machine with low extra overhead, is widely adopted for such purposes, e.g., in Amazon EC2 [5] and GoGrid [6].

With the introduction of this extra virtualization layer, the communication performance of the MPI library, which is widely used by the HPC applications, would suffer a high penalty [7, 8] when the virtualized platform by Xen is overcommitted, i.e., the number of executable VCPUs is more than the number of available PCPUs in the system. This overcomitting situation is very likely to happen in Cloud Computing environments, because Cloud service providers tend to consolidate several virtual machines to limited physical server nodes in order to fully exploit the computing resources. For example, the work in [9] analyzed 17 real-world datacentres and the customer data show that the average VCPUs-to-core ratio is 4:1. The work in [10] investigated the impact of virtualization on Amazon EC2. The work revealed that there were several virtual machines (called instances) running on one physical server, and consequently, one instance only receives a share of the processor (e.g., a small instance typically receives only a 40% to 50% share of the processor). The performance problem caused by overcommitting was also clearly exhibited in [3]. In [3], the NAS EP benchmark [11] was run on Amazon EC2 with the requested number of cores increasing from 4 to 32. For each requested number of cores (VCPUs), the results show that the execution performance greatly fluctuates among the different runs of the same configuration, especially when the benchmark was run by a large number of requested cores.

In this paper, we investigate the MPI communication performance problem in overcommitted virtualized systems, and reveal that the underlying reasons behind the performance hit is the *busy-polling* mechanism employed by the MPI libraries. Further, we propose a solution to improve the MPI communication performance in the overcommitted systems, and implemented the proposed solution into MPICH2-1.2.1, which is one of the most popular MPI libraries. Extensive experiments have been conducted for the implementation. The results show that in the worst-case overcommitted scenarios with two VCPUs, our solution can improve the MPI communication performance by up to about 700 times in the Credit scheduler and about 4 times in the SEDF scheduler. And in the real-world overcommitted virtualized systems, the execution performance of most HPC applications can be improved to different extents, depending on their communication patterns and the overcommitting level.

The work presented in this paper can give the Cloud service providers the insight into the performance problem of running the parallel programs with MPI libraries, and foster further studies on other potential approaches to avoiding such performance loss.

The rest of the paper is organized as follows: Section II gives a brief survey on the related works. Section III conducts the benchmarking experiments to demonstrate the performance penalty occurring in the overcommitting situation, as well as the impacts of such penalty on the performance of benchmark programs. Section IV thoroughly analyzes the performance problem and reveals its underlying causes. Our solution to address the problem is presented in Section V. The proposed solution is then evaluated in Section VI. Section VII concludes the paper.

II. Related Work

The research work related to our study can be categorized into three aspects:

A) Performance studies of HPC workloads on Xen and HPC-in-the-Cloud

Since the Xen virtualization technology was introduced, many performance studies [2, 3] were conducted to investigate the feasibility of using virtualized platforms to run HPC workloads. Among these studies, only a few of them [3, 7] noticed the high performance penalty on MPI communication in overcommitted virtualized systems. However, some of these research studies erroneously attribute the reason of this performance problem to the scheduler of the hypervisor.

In recent years, we have seen an increasing adoption of cloud computing in a wide range of scientific applications, such as high-energy and nuclear physics, bioinformatics, astronomy and climate research [2]. However, in a typical Cloud environment, the users have no control over the underlying physical resources, and overcommitting may occur unwittingly, which results in unexpected performance penalty.

B) Improving MPI communication performance on Xen

The research work in this aspect concentrates on improving the performance of the underlying networks (e.g., InfiniBand, 10Gbps Ethernet and the TCP/IP stack) in virtualized environments. The techniques proposed by these studies include VMM-bypassing [12], using XenLoop [13] alike techniques to replace the loopback network with the shared memory [14], and using a dedicated CPU core to handle all communications [15]. However, none of these studies addressed the overcommitted scenarios.

C) Research on the Xen scheduler

During the short development history of the Xen hypervisor, at least three scheduling algorithms [16] have been introduced, including Borrowed Virtual Time (BVT), Simple Early Deadline First (SEDF) and Credit. As the BVT scheduling algorithm is no longer supported by the latest Xen releases, only the SEDF and the Credit schedulers are investigated in this paper.

Simple Earliest Deadline First (SEDF) is a real-time scheduling algorithm. In the SEDF scheduler, each domain Dom_i specifies its CPU requirements with a tuple (s_i, p_i, x_i), where the slice s_i and the period p_i together represent the CPU share that Dom_i requests: Dom_i will receive at least s_i units of time in each period of length p_i. The boolean flag x_i indicates whether Dom_i is eligible to receive extra CPU time. SEDF distributes this slack time in a fair manner after all runnable domains receive their CPU shares.

The Credit scheduler is currently the default scheduler in Xen. The scheduler allocates the CPU resources to VCPU according to the *weight* of the domain that the VCPU belongs to. It uses *credits* to track VCPU's execution time. Each VCPU has its own credits. If one VCPU has credits greater than 0, it gets UNDER priority. When it is scheduled to run, its credit is deducted by 100 every time it receives a scheduler interrupt that occurs periodically once every 10ms (called a *tick*). If one VCPU's credit is less than 0, its priority is set to OVER. All VCPUs waiting in the run-queue have their credits topped up once every 30ms, according to their weights. The higher weight a domain has, the more credits are topped up for its VCPUs every time. An important feature of the Credit scheduler is that it can automatically load-balance the virtual CPUs across PCPUs on a host with multiple processors. The scheduler on each PCPU can "steal" VCPUs residing in the run-queue of its neighboring PCPUs once there are no VCPUs in its local run-queue. The biggest shortcoming of the Credit scheduler is its poor support for I/O events. The improvement works, such as the introductions of the BOOST priority [17] and BCredit [18], which essentially works by changing the dom0 VCPUs' credits so that dom0 remains in UNDER state and keeps its BOOST priority, have been made to alleviate the problem.

III. Performance Penalty

We use a two-core COTS (Commercial off-the-shelf) PC server as our testbed to investigate the performance problem of MPI communication in overcommitted virtualized systems. It has an Intel Core 2 Duo E6550 processor. The processor has two cores, running at 2.33GHz with 128KBytes L1 cache and shared 4MBytes L2 cache. It is configured with 2GBytes DDR2 memory and 160GB SATA hard disk drive. For virtualization, we use Xen of version 3.4.2. The guest domains (including dom0 and domUs) are installed Redhat Enterprise Linux x86_64 with version 5.1. The Xen-Linux of kernel version 2.6.18.8 is used to boot all guest systems including dom0. Dom0 is the only privileged domain, which contains all drivers of the physical devices, and is configured with 512 MBytes memory. The domUs used in the experiments are Para-virtualized (PV) guests configured with 256 MBytes memory, 4 GBytes virtual hard disk drive, and a virtual network interface card to communicate with the outside world (via the bridge network of dom0). All the experiments in this paper adopt default scheduling parameters for the guest domains: On the Credit scheduler, all domains will have the same *weight* number of

256, the *cap* value 0; On the SEDF scheduler, the scheduling parameter for dom0 is (15ms, 20ms, 1) and that of the domUs is (0ms, 100ms, 1).

As for the MPI library, we chose MPICH2-1.2.1 [19]. We also noticed that other MPI libraries, such as MPICH-1.2.7p1 [20] and OpenMPI-1.4 [21] have similar communication performance problem on overcomitted virtualized systems.

A. Communication Performance

The benchmark we used to collect MPI communication performance data is Beff [22]. We only show the communication performance of the Send/Receive tests in this subsection, since the performance data from other tests (e.g., all-to-all, non-blocking, and etc.) follows the similar pattern.

We boot the testbed with a 2.6.28-rc2 Linux kernel that does not have SMP support to form a uni-processor (UP) environment, and obtain the MPI communication performance (we call it **native_up**) between two benchmark processes of Beff, and use the native_up data as the reference for communication performance.

Since some schedulers of Xen, such as the Credit scheduler, employs the aggressive VCPU migration policy and spreads VCPUs across PCPUs to achieve load-balancing, the VCPUs may switch PCPUs during execution. Therefore, it is difficult to determine whether and in which periods the VCPUs are overcommitted on the fly. In order to facilitate further discussions and effectively evaluate the communication and execution performance of the applications, we manually pinned the VCPUs on one PCPU, which should be the worst-case overcomitted situation. As by using MPI primitives, the processes running in the virtualized system may communicate via two possible channels: the network system of dom0 (i.e., the inter-VM case) and shared memory (i.e., the intra-VM case), we defined the four worst overcommitted cases as follows:

- **inter_vm_credit_pin:** The hypervisor uses the Credit scheduler and is booted with two processing cores. Two domUs are created, each of which has only one VCPU, and all of their VCPUs are pinned to the same physical core. The MPI communication performance between these two domUs is then measured.
- **inter_vm_sedf_pin:** The same as inter_vm_credit_pin, except that the Xen hypervisor uses the SEDF scheduler.
- **intra_vm_credit_pin:** The hypervisor uses the Credit scheduler and is booted with two processing cores. One domU with two VCPUs is created, and both of its VCPUs are pinned to the same physical core. The MPI communication performance inside the domU is measured.
- **intra_vm_sedf_pin:** The same as intra_vm_credit_pin, except that the Xen hypervisor uses the SEDF scheduler.

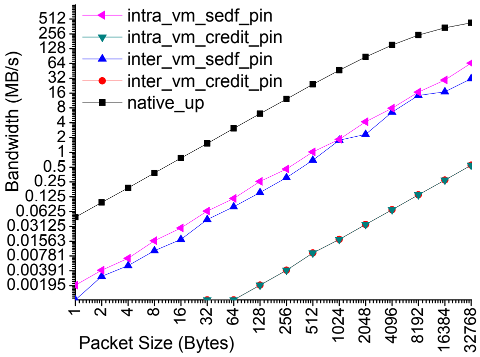


Figure 1: MPI communication performance in the worst-case overcommitted scenarios

The performance data is illustrated in Fig. 1 (The inter_vm_credit_pin curve and intra_vm_credit_pin curve overlap with each other as the data in these two cases are close).

From Fig. 1, it can be observed that compared with the native_up case, the MPI communication performance of the defined overcomitted cases is much worse. In all such four overcommitted cases, the communication performance in the SEDF scheduler is always better than that obtained in the Credit scheduler. This is because the SEDF scheduler defines smaller time slices than the Credit scheduler, and the frequent scheduling reduces the performance loss caused by the busy-polling mechanism employed by the MPI communication library, which will be discussed in detail later.

B. Performance of Selected Benchmark Programs

To further investigate the performance problem in overcommitted systems, we employ three benchmark programs from the NPB suite of version 3.3 [11], i.e., is.A.2 (integer sorting, which relies on all-to-all communication to exchange intermediate results), lu.A.2 (solving five coupled parabolic/elliptic partial differential equations, which needs to communicate to exchange data during computation) and ep.A.2 (random number generator, which is "embarrassingly" parallel computing in that no communication is required for generating the random numbers). The reason we choose these three benchmarks is that they exhibit three typical characteristics of HPC applications: communication intensive with little computation, i.e., IS; CPU intensive with little communication, i.e., EP; and that lies in the middle, i.e., LU. The execution times of these benchmark programs (MPI version) are presented in Table I (each figure in the table as well as in other tables in this paper is the average over four independent runs).

It can be observed from Table I that with the Credit scheduler, compared with the execution time in the native_up case, in the inter_vm_credit_pin case, the execution time of is.A.2 and lu.A.2 is about 30 times and 4 times longer, respectively. And compared with the native_up data also, in the

Table I: The execution times of selected benchmark programs in the worst-case overcommitted scenarios (in Seconds)

	is.A.2	lu.A.2	ep.A.2
native_up	3.39	193.84	31.26
inter_vm_credit_pin	106.2	1039.28	36.51
inter_vm_sedf_pin	4.61	228.65	37.86
intra_vm_credit_pin	45.08	275.56	36.31
intra_vm_sedf_pin	3.97	195.92	37.74

intra_vm_credit_pin case, the execution time of is.A.2 and lu.A.2 is about 12 times and 0.5 times longer, respectively.

The performance penalty is less serious in the SEDF scheduler: Compared with the native_up data, in inter_vm_sedf_pin case, the execution time of is.A.2 only increases by 35%, and that of lu.A.2 increases by about 18%. In intra_vm_sedf_pin case, the execution time of is.A.2 is increased by 17% while that of lu.A.2 is close to the native performance. This is because the SEDF scheduler schedules the VCPUs with smaller time slices and this greatly reduces the chances of wasting the CPU cycles caused by the busy-polling mechanism employed by the MPI library.

It can also be observed that the performance of ep.A.2 seems not to be significantly affected. This is because the processes of ep.A.2 do not need to communicate with each other during the computing phase. However, it can be observed that the execution time of ep.A.2 in the SEDF scheduler is slightly higher than that in the Credit scheduler. This is because with smaller time slices, the VCPUs will be scheduled more frequently in the SEDF scheduler, which results in the higher overhead in the hypervisor.

C. Discussions

We created the worst VCPU overcommitting cases by deliberately pinning the VCPUs to the same physical processor. Although in reality people seldom deliberately confine multiple VCPUs to running on the same PCPU, performance penalty similar to that in these four overcommitted cases may still occur in real overcommitted virtualized systems. There are at least two main reasons:

First, *in the current Xen architecture, the scheduling decisions are made locally: Each PCPU of a multi-processor system maintains its VCPU run-queue and schedules them to run without any coordination with other PCPUs in the system. Therefore, in real overcommitted systems, it is highly likely that two VCPUs that host the processes communicating with each other via MPI primitives are not scheduled to run in parallel. When the VCPU that hosts the message receiving process is running in one PCPU but the VCPU that hosts the message sending process is scheduled out in another PCPU, the busy-polling receiving may still occur (the difference is just that the two VCPUs now reside in the run-queues of different PCPUs).*

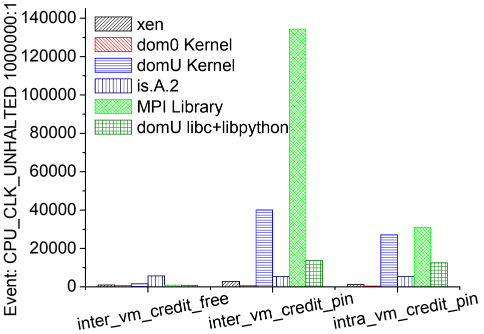


Figure 2: A comparison of XenOprofile data

Second, *in the load-balancing strategy of the Credit scheduler, which is the current default scheduler in Xen, VCPUs are allowed to be migrated among all available PCPUs of the system. This strategy may cause the VCPUs communicating with each other to run on the same PCPU, which forms exactly the same overcommitting scenario as those described in Subsection III-A and III-B. And this situation is more likely to happen in the overcomitted virtualized systems, since there are more runnable VCPUs.*

IV. Causes of the Performance Penalty

We use XenOprofile [23] to collect statistical performance data when running is.A.2 in the configurations of inter_vm_credit_pin and intra_vm_credit_pin. For comparison, we also collect the performance data when running is.A.2 in the configuration of inter_vm_credit_free, in which the VCPUs of two domUs are not pinned to the same PCPU (i.e., to simulate the non-overcommitted case). Fig. 2 illustrates the performance data[1].

From Fig. 2, it can be observed that compared with the non-overcommitted case, the number of CPU cycles spent on domU kernels, MPI library and the user-level libraries of domU increases dramatically in the overcommitted cases, while the application (i.e., is.A.2) itself always consumes almost the same amount of CPU cycles.

From the XenOprofile performance data, we found that in the inter_vm_credit_pin case, the MPI functions in the library (i.e., MPICH2-1.2.1) that consumes most CPU cycles are *MPID_nem_tcp_connpoll*, *MPIDI_CH3I_Progress* as well as *MPID_nem_network_poll*. While in the intra_vm_credit_pin experiments, the function that consumes most CPU cycles is *lmt_shm_recv_progress*. A close look into the source code of the library reveals that these functions belong to the message receiving mechanism of MPI: For block receiving operations (e.g., MPI_Recv and MPI_Wait), the library will continuously poll the socket file descriptor (FD) set (the inter-VM case) or the shared memory (the intra-VM case) until a message is received. During this process, non-blocking operations on the socket FD set or

[1] we only attribute *randlc*, *rank*, *full_verify* and *create_seq* to is.A.2

the shared memory are performed. Therefore the polling mechanism will not bring the VCPU that hosts the message-receiving process into the *block* status, which makes the VCPU keep running until the end of the time-slice.

In order to prevent this busy-waiting message receiving (we call it "*busy-polling*") mechanism from lasting so long as to consume large quantities of processing resources, MPI library uses a threshold based yielding mechanism: When the library polls up to the pre-defined times (e.g., 1000 times for MPICH2-1.2.1), the message receiving process should give up its possession of the processor by issuing a yielding syscall (i.e., *sched_yield* in Linux) in the guest operating system. However, in virtualized environment, for the overcommitted case of inter_vm_credit_pin, there is only one process running inside each of the domUs. The process that polls the socket FD set to receive messages will be scheduled to run again immediately after its yielding syscall, and therefore continue to consume more CPU cycles in vain. In the meanwhile, the VCPU hosting the message sender is not running, but waiting to be scheduled in the run-queue of the same physical processor. This explains the high performance penalty experienced when running is.A.2 in the overcommitted case of inter_vm_credit_pin, since is.A.2 heavily relies on all-to-all communications (i.e., MPI_Alltoall and MPI_Alltoallv) to exchange intermediate results between the processes residing in different domUs, which results in frequent busy-polling. Consequently, the frequent polling and yielding within the guest systems will inevitably cause high CPU consumptions of the guest OS kernel as well as the user-level libraries.

However, the previous analysis cannot explain the performance penalty experienced in the intra_vm_credit_pin case, since two communicating processes of is.A.2 co-exist in the same domU with two VCPUs. Normally, if one process gives up its possession of the processor due to block receiving and polling up to the pre-defined times, another process should be scheduled to run, and this should not have such high performance loss. In order to explain the performance problem in this case, we propose a hypothesis: For the processes communicating with each other using blocking MPI message receiving primitives, each of the processes will be eventually scheduled to execute on each of the VCPUs of the SMP domU. After that, when one of the processes that is blocked in message receiving tries to give up the processor possession by yielding, the scheduler of the guest OS will find that the other process is still running on another processor (i.e., VCPU from the hypervisor's point of view), and therefore the yielding process will be scheduled in the same processor to run immediately again, which continues to consume more CPU cycles in vain until the end of the VCPU time slice. To verify this hypothesis, we employ a simple ping-pong benchmark, which invokes two processes exchanging a message buffer of 1000 bytes by using MPI_Send and MPI_Recv, to run in the domU of the intra_vm_credit_pin case. The message round trip time (RTT) is recorded and illustrated in Fig. 3.

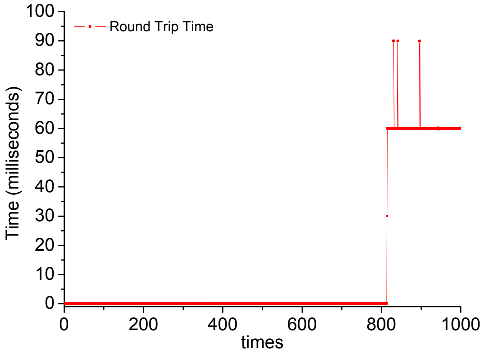


Figure 3: RTT of ping-pong benchmark in intra_vm_credit_pin case

From Fig. 3, it can be observed that for the first 813 times (the 814th time is the breaking point) of communications, the RTT is rather small (below 0.05ms), while after the breaking point, the RTT jumps to 60ms for most of the time and never comes down.

The performance data can be explained as follows. When the benchmark program is invoked, the two communicating processes (denoted as process A and B) are scheduled concurrently on one virtual processor (since it is in the overcommitted scenario and only one VCPU is scheduled to execute at a time). This results in the low RTT before the breaking point, as the yielding operation causes actual process scheduling. However, after the VCPU hosting these two processes uses up its 30ms time slice defined by the Credit scheduler, the VCPU is going to be scheduled out. At this moment, there is a high possibility that one of the processes (assume it is process A) is still running on the VCPU. After that, the hypervisor schedules the other VCPU of the domU to run, which will in turn put the other communicating process (process B) in execution. Nevertheless, the yielding of process B will not cause the actual process scheduling in the guest system again, since process A is considered "running" on the scheduled-out VCPU. This situation will last until the termination of the block receiving operations.

Another noticeable observation that can be observed from Fig. 3 is that after the breaking point, the RTT stabilizes at around 60ms (90ms in some occasions), which is twice of the length of the time slice defined by the Credit scheduler. This further verifies our deduction that the busy-polling mechanism adopted by MPI will result in meaningless CPU consumptions until the VCPU time slice is used up in the intra-VM cases.

From the above discussions, we found that in overcommitted virtualized systems, the MPI performance penalty is caused by the busy-polling mechanism employed by the MPI libraries. However, from another perspective, *the* *performance penalty is also due to the lack of communication between the scheduler of the hypervisor and the scheduler of the guest operating system*. On one hand, since these two schedulers work independently in the current Xen architecture, the scheduler of the guest OS may have the misperception that a processor is still running even after its container (i.e., the VCPU) has actually been scheduled out by the hypervisor. On the other hand, the scheduler of the hyervisor does not really understand whether the workload inside a VCPU is meaningful.

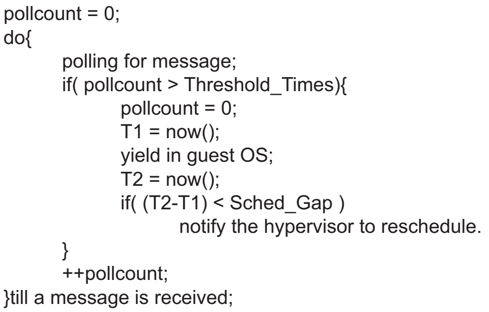
V. Our Solution

From the discussion in Section IV, we can deduce that the key to improve the communication performance in the overcommitted virtualized systems is to avoid wasting processing resources on unnecessary message polling. There are at least three possible ways to achieve this goal: 1) replacing the busy-polling mechanism with block-polling, 2) improving the scheduling algorithm of the hypervisor, and 3) exposing the scheduling information of the guest OS to the hypervisor or vice versa.

For the first possible solution, the non-blocking polling operations on the socket FD set or shared memory can be substituted with the blocking operations. For example, for the inter-VM cases, blocking operations should be performed on the FD set, instead of the original non-blocking ones used in the current MPI library implementation. It seems that this method can avoid wasting CPU resources unnecessarily on message polling. However, this solution will inevitably cause frequent and expensive VCPU context switching, and will therefore damage the performance. Moreover, it is hard, if not impossible, to implement blocking polling operations on the shared memory.

It is also possible to improve the communication performance from the perspective of the hypervisor's scheduling algorithm. However, the smaller time slices also mean more frequent scheduling, which consequently results in heavier scheduling overhead in the hypervisor and negatively affects the performance of applications that do not perform much communications, such as EP from NPB. It is also a promising solution to schedule the VCPUs by groups on multiprocessor hosts (i.e., Gang-scheduling [24]) according to which VCPUs communicate with each other. Although the group scheduling approach has the potential to become a general solution to mitigate the performance penalties in overcommitting VCPU scenarios, adopting such a technique will result in the complexity in the system design, and require addressing many difficult issues, such as how to collect the communication pattern among the VCPUs, how to ensure other non-HPC workloads are not negatively affected, and so forth. We plan to investigate this approach in our future work.

In this paper, we take the third approach to improve MPI communication performance in the overcommitted virtualized systems, i.e., exposing scheduling information of the guest OS to the hypervisor. We employ a straightforward solution: When the process that is blocked at receiving messages is ready to give up its processor possession (because the polling times reach the threshold in the guest OS), it will notify the hypervisor to reschedule if it is appropriate. The pseudo-code of the revised busy-polling mechanism is demonstrated in Fig. 4[2].



```
pollcount = 0;
do{
    polling for message;
    if( pollcount > Threshold_Times){
        pollcount = 0;
        T1 = now();
        yield in guest OS;
        T2 = now();
        if( (T2-T1) < Sched_Gap )
            notify the hypervisor to reschedule.
    }
    ++pollcount;
}till a message is received;
```

Figure 4: The pseudo-code for revised busy-polling

Before notifying the hypervisor to conduct rescheduling operations, a *gap* value is recorded from the time when the process is yielded to the time when the process is scheduled to execute again within the guest OS. The gap value is then compared with a threshold value (i.e., *Sched_Gap*) we defined. Only when the value is smaller than *Sched_Gap*, the process will invoke hypercall in the guest OS to notify the hypervisor for rescheduling. This is because we have to consider the case where multiple processes execute on one processor concurrently. In this case, it is not appropriate to reschedule the VCPU as a whole, since the concurrent scheduling may suggest some communicating tasks are in progress. While in the overcommitted cases, the process blocked in message receiving is likely to be the sole running process on the processor as we have analyzed in the previous section. This means the measured time gap will be rather small (below 2 μs for most of the time), and will trigger the reschedule operation in the hypervisor. We set *Sched_Gap* to be 100 μs empirically after conducting some supporting experiments.

We apply this revision to both network polling and the shared memory polling mechanisms in the MPI library (i.e., MPICH2-1.2.1 examined in this paper), and have submitted the code patch to the maintainers of the MPICH2 project[3].

After a MPI program using our code patch is invoked, during initializing phase (i.e., when MPI_Init is being called), it will first detect if it is running inside a PV guest system created by Xen or not. The detection is performed by trying to open the PRIVATE_CMD interface (i.e., */proc/xen/privcmd*, which is typically provided by the

[2] $Threshold\_Times = 1000$ for MPICH2-1.2.1

[3] The code patch can also be downloaded from http://grid.hust.edu.cn/zyshao/code/xen-mpich2-1.2.1.patch

Linux PV guests). Only when running inside a PV guest system (i.e., PRIVATE_CMD interface is available), will the MPI program adopt the revised busy-polling mechanism as shown in Fig. 4. In this way, the same code can run on both virtualized guests and native environments. The hypercall that notifies the hypervisor of rescheduling is also issued via this PRIVATE_CMD interface.

VI. Performance Evaluation

We conducted extensive experiments to evaluate the performance of our solution. The experiments were carried out not only on the predefined worst-case overcommitted scenarios we used to benchmark and analyze the performance penalty in Section III and IV, but also on a 4-core PC server to simulate overcommitted virtualized systems in real-world.

A. Experiments on the Worst Overcommitting Cases

1) Communication Performance: The performance of MPI communications in the worst-case overcommitted scenarios with our revised MPI library is shown in Fig. 5.

In Fig. 5, the performance data labeled with the "**rev_**" prefix correspond to the data with the revised MPICH2-1.2.1. From Fig. 5, it can be observed that compared with the original data, the communication performance by employing our solution is improved greatly: For the Credit scheduler, the communication performance is improved by up to about 300 times in the inter-VM case, and by up to 700 times in the intra-VM case, while for the SEDF scheduler, the performance is increased by up to about 3 times in the inter-VM case and by up to about 4 times in the intra-VM case.

Moreover, in the intra-VM cases, the communication performance obtained by the revised MPI library is very close to the performance in the native UP environment. This is because in the intra-VM cases, most of the communications are conducted via the shared memory, which is very similar as the inter-process communication in the native case. In the inter-VM cases, the communication data exchanged between domUs needs to be relayed by the loopback network of dom0. This results in worse performance than that in the intra-VM cases although it is still much higher than that with the original MPI library.

2) Performance of Selected Benchmark Programs: Table II presents the execution times of the benchmark programs (i.e., is.A.2, lu.A.2 and ep.A.2) from the NPB suite that is linked with the revised MPICH2-1.2.1.

From Table II, it can be observed that after applying our solution, the execution time of is.A.2 and lu.A.2 is very close to that of running inside the native UP environment in the inter_vm_credit_pin case. Compared with Table I, the performance penalty caused by poor communication is eliminated. There is almost no change in the execution time of ep.A.2, since the benchmark program does not invoke MPI communications during the computing phase, which also shows that our solution does not have negative impact on the application execution with little communications.

Table II: The execution times of selected benchmark programs after improvement in the worst-case overcommitted scenarios (in Seconds)

	is.A.2	lu.A.2	ep.A.2
native_up	3.39	193.84	31.26
rev_inter_vm_credit_pin	3.22	188.08	36.1
rev_inter_vm_sedf_pin	3.84	203.7	37.51
rev_intra_vm_credit_pin	5.12	199.16	36.38
rev_intra_vm_sedf_pin	3.04	192.16	37.54

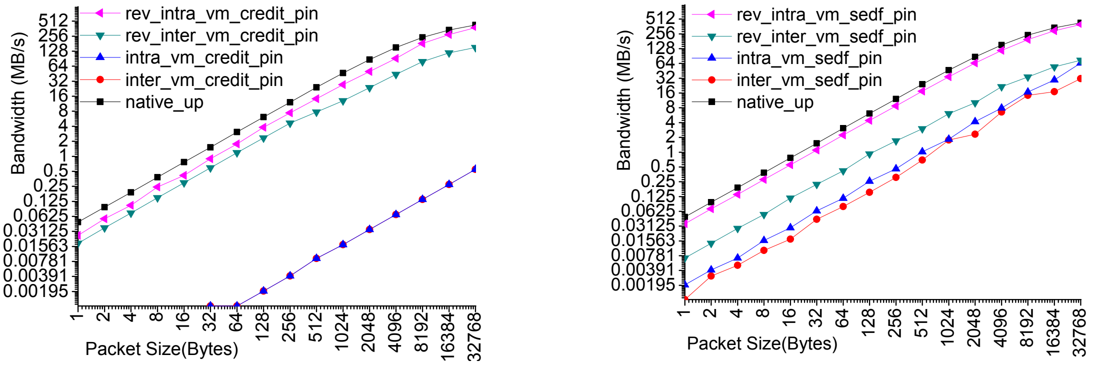
Table III: The execution times of selected benchmarks with different overcommitting levels in the Credit scheduler (in Seconds)

number of domU(s)		1	2	3	4
is.A.4	original	0.85	11.01	16.62	22.48
	revised	0.85	11.13	6.79	7.46
lu.A.4	original	72.763	214.35	332.21	466.04
	revised	72.64	198.04	272.69	353.45
ep.A.4	original	13.33	26.66	43.24	56.19
	revised	13.33	26.49	39.74	52.92

B. Experiments on the Real Overcommitted Systems

In order to exhibit the performance penalty on benchmark programs and verify the effectiveness of our solution in the real use cases, we conduct experiments on a 4-core overcommitted host. The host has two way Intel Xeon 5110 processor (Woodcrest), each of which has two 1.6GHz processing cores with 4MBytes shared L2 cache. It is configured with 2GB DDR2 memory, 160GB SATA hard disk drive, and one NetXtreme II BCM5708 Gigabit Ethernet interface. On this platform, each domU is configured with four VCPUs. The *overcommitting level* of the system is controlled by the number of domUs running on the platform. As the number of the domUs increases (hence, the number of VCPUs), the *overcommitting level* also increases. Unlike the experiments in subsection VI-A, the VCPUs are not deliberately pinned to run on a certain core. In the experiments, identical benchmark programs from NPB (i.e. is.A.4, lu.A.4 and ep.A.4) are invoked simultaneously in these domUs, and their execution times are recorded. Table III and IV demonstrate the average execution time of these benchmark programs in the Credit scheduler and in the SEDF scheduler, respectively.

From Table III, it can be observed that when the overcommitting level is low (in case of the one domU and two domUs), the execution time of the benchmark programs with the revised MPI library is very close to that with the original MPI library. This is because given a small number of candidate VCPUs (two for each PCPU on average), the possibility that the VCPUs with meaningful workload are scheduled to run is still small. Another interesting observation is that when there are two domUs, the performance of is.A.4 is even slightly worse than the original one. This is because is.A.4 has intensive communications, which may cause more frequent VCPU context switching when using the revised



(a) for the Credit scheduler

(b) for the SEDF scheduler

Figure 5: The improved performance of MPI communications in the worst-case overcommitted scenarios

Table IV: The execution times of selected benchmarks with different overcommitting levels in the SEDF scheduler (in Seconds)

number of domU(s)		1	2	3	4
is.A.4	original	0.85	1.64	7.84	10.09
	revised	0.84	1.62	2.49	3.36
lu.A.4	original	72.79	156.10	295.24	367.74
	revised	72.88	153.51	251.33	359.27
ep.A.4	original	13.60	27.25	40.44	54.20
	revised	13.44	26.97	40.26	53.91

MPI library.

However, when the overcommitting level becomes higher (in the cases of three and four domUs), the execution times of the benchmark programs with the revised MPI libraries are significantly shorter than those with the original MPI libraries. This is because with more VCPUs running, the possibility that the VCPUs in the same domU are not scheduled to run in parallel is significantly higher, which generates the similar performance penalty as that experienced in the intra_vm_credit_pin case. Meanwhile, with the increase of candidate VCPUs, the possibilty of scheduling VCPUs with actual workload to run by our scheme is much higher.

From Table IV, it can be observed that in most cases (except for is.A.4), the performance improvement in the SEDF scheduler is smaller than that in the Credit scheduler. This is because the SEDF schedules the VCPUs with smaller time slices than the Credit scheduler, which results in less performance penalty due to the busy-polling mechanism employed by the MPI library. The reason why our solution can still achieve significant improvement for is.A.4 in the SEDF scheduler is because is.A.4 has very high communication-to-computation ratio, which counteracts the gains achieved by the shorter time slices in SEDF.

Another observation from both Table III and Table IV is that for the benchmark programs with original MPI library that need communication during computation (e.g., is.A.4 and lu.A.4), the proportion of the increase in their execution time is much higher than that of the increase in the number of competing domUs. This explicitly exhibits the high performance penalty on MPI communication placed on the applications due to overcommitment. Our solution provided in this paper can alleviate this performance degradation. For example, in the SEDF scheduler, the execution time of is.A.4 increases proportionally with the increment on the number of competing domUs.

VII. Conclusion and Future Works

In this paper, we focus on investigating the performance problem of MPI communication in the overcommitted virtualized systems. By benchmarking the worst-case overcomitted scenarios and thorough analysis, we revealed the reason behind such performance penalty is the "busy-polling" message receiving mechanism employed by the MPI libraries. Such penalty significantly degrades the execution performance of the MPI applications that contain communications and run in the overcomitted virtualized computer systems. We proposed our solution in this paper to mitigate the penalty and implemented our solution into a popular MPI library. We have carried out experiments to verify the effectiveness of the solution in the real-world overcommitted virtualized systems. The results show that compared with the original MPI library, the library that patches our solution can improve the performance of applications to different extents, depending on their communication patterns and the overcommitting level.

We believe that the solution proposed in this paper can be further enhanced. Our future work has been planned to further tune the proposed solution. Also we plan to investigate the feasibility of deploying the gang-scheduling technique into the hypervisor, which we believe might be a better solution to eradicate the performance penalty in the overcommitted virtualized systems.

Acknowledgment

This work is supported by National 973 Basic Research Program of China under grant No.2007CB310900, National Natural Science Foundation of China under grant No.60903022 and the MoE-Intel Information Technology Special Research Foundation under grant No. MOE-INTEL-10-05.

References

[1] NIST definition of cloud computing. [Online]. Available: http://csrc.nist.gov/groups/SNS/cloud-computing/index.html

[2] C. Evangelinos and C. N. Hill, "Cloud Computing for Parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon EC2," in *Proc. of Cloud Computing and Applications Conference*, October 2008.

[3] S. Akioka and Y. Muraoka, "HPC benchmarks on Amazon EC2," in *Proc. of IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, 2010, pp. 1029–1034.

[4] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the Art of Virtualization," in *Proc. of the ACM Symposium on Operating Systems Principles (SOSP'03)*, October 2003, pp. 164–177.

[5] Amazon EC2 Cloud. [Online]. Available: http://aws.amazon.com/ec2

[6] Gogrid Cloud. [Online]. Available: http://www.gogrid.com

[7] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan, "Performance Implications of Virtualizing Multicore Cluster Machines," in *Proc. of the ACM 2nd workshop on System-level virtualization for high performance computing*, 2008, pp. 1–8.

[8] G. Vallee and S. L. Scott, "XenOSCAR for cluster virtualization," in *Proc. of ISPA Workshop on XEN in HPC Cluster and Grid Computing Environments (XHPC'06)*, March 2006, pp. 381–383.

[9] V. Soundararajan and J. M. Anderson, "The Impact of Management Operations on the Virtualized Datacenter," in *Proc. of the 37th annual international symposium on Computer architecture (ISCA'10)*, 2010, pp. 326–337.

[10] G. Wang and T. S. E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *Proc. of The IEEE 29th Conference on Computer Communications (INFOCOM'10)*, 2010, pp. 1–9.

[11] NPB: NAS Parallel Benchmarks. [Online]. Available: http://www.nas.nasa.gov/Resources/Software/npb.html

[12] W. Huang, J. Liu, B. Abali, and D. K. Panda, "A case for high performance computing with virtual machines," in *Proc. of the ACM 20th annual international conference on Supercomputing*, 2006, pp. 125–134.

[13] J. Wang, K. L. Wright, and K. Gopalan, "XenLoop: a transparent high performance inter-vm network loopback," in *Proc. of the 17th international symposium on High performance distributed computing (HPDC'08)*, October 2008, pp. 109–118.

[14] W. Huang, M. J. Koop, Q. Gao, and D. K. Panda, "Virtual Machine Aware Communication Libraries for High Performance Computing," in *Proc. of the 2007 ACM/IEEE conference on Supercomputing (SC'07)*, 2007.

[15] J. Liu and B. Abali, "Virtualization polling engine (VPE): using dedicated CPU cores to accelerate I/O virtualization," in *Proc. of the ACM 23rd international conference on Supercomputing*, 2009, pp. 225–234.

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

[17] D. Ongaro, A. L. Cox, and S. Rixner., "Scheduling I/O in Virtual Machine Monitors," in *Proc. of ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'08)*, 2008, pp. 1–10.

[18] Boost Credit. [Online]. Available: http://www.mailinglistarchive.com/xen-devel@lists.xensource.com/msg58284.html

[19] MPICH2: High Performance and Widely Portable MPI. [Online]. Available: http://www.mcs.anl.gov/research/projects/mpich2/

[20] MPICH: A Portable Implementation of MPI. [Online]. Available: http://www.mcs.anl.gov/research/projects/mpi/mpich1/

[21] Open-MPI: Open Source High Performance Computing. [Online]. Available: http://www.open-mpi.org/

[22] Effective bandwidth (beff) benchmark. [Online]. Available: https://fs.hlrs.de/projects/par/mpi//b_eff/

[23] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel, "Diagnosing Performance Overheads in the Xen Virtual Machine Environment," in *Proc. of the First ACM/Usenix Conference on Virtual Execution Environments (VEE'05)*, June 2005, pp. 13–23.

[24] D. G. Feitelson, "Packing schemes for gang scheduling," in *Proc. of Job Scheduling Strategies for Parallel Processing(OPPS'96 workshop)*, April 1996, pp. 89–110.