vChecker: an application-level demand-based co-scheduler for improving the performance of parallel jobs in Xen

Peng Jiang¹ · Ligang He¹ · Shenyuan Ren¹ · Zhiyan Chen¹ · Rui Mao²

Published online: 15 February 2019
© The Author(s) 2019

Abstract

Big data analysis requires the speedup of parallel computing. However, in the virtualized systems, the power of parallel computing is not fully exploited due to the limit of current VMM schedulers. Xen, one of the most popular virtualization platforms, has been widely used by industry to host parallel job. In practice, the virtualized systems are expected to accommodate both parallel jobs and serial jobs, and resource contention between virtual machines results in severe performance degradation of the parallel jobs. Moreover, the physical resource is vastly wasted during the communication process due to the ineffective scheduling of parallel jobs. Unfortunately, the existing schedulers of Xen are initially targeting at serial jobs, which are not capable of correctly scheduling the parallel jobs. This paper presents vChecker, an application-level co-scheduler which mitigates the performance degradation of the parallel job and optimizes the utilization of the hardware resource. Our co-scheduler takes number of available CPU cores in one hand, and satisfies need of the parallel jobs in other hand, which helps the credit scheduler of Xen to appropriately schedule the parallel job. As our co-scheduler is implemented at application level, no modifications on the hypervisor is required. The experimental result shows that the vChecker optimizes the performance of the parallel job in Xen and enhances the utilization of the system.

Keywords Xen · Parallel · Virtualization · Credit

1 Introduction

Bigdata analytics is facing the super large volume of data, which normally requires the parallel computing to accelerate the analytic process nowadays. As the virtualization technique has been vastly adopted by the parallel computing server providers, more and more data analytic jobs are executed on the virtualized platform. Among various virtualization platforms, Xen, a novel virtualize solution, has become one of the most popular virtualization platforms in this field. Xen provides a hypervisor which directly works on top of hardware layer to separate

hardware management and software management. VMs are running above the hypervisor known as domains. A privileged domain (dom0) is created at boot time and is given access to the hypervisor's controlling interface, which makes dom0 a key role in managing all other domains (called guest domains or domUs), for example, creating VM, deleting VM, and terminating VM. Xen uses a credit scheduler to allocate the computing resource to vCPUs, which achieves outstanding performance overall.

Credit scheduler, which is a fairly proportionally sharing based strategy, is the primary scheduler of Xen. This scheduler provides a global workload balance during the process of resource allocation, which has achieved an outstanding overall performance. Unfortunately, researchers found a severe problem when it hosts parallel jobs. Since parallelization becomes the mainstream in modern program designing, various parallel job frameworks have been developed to enable the implementation of the parallel fashion. A parallel job usually has communications between the subtasks to exchange the intermediate result, which requires related vCPUs to be online at same time.

✉ Ligang He
Ligang.He@warwick.ac.uk
Rui Mao
mao@szu.edu.cn

¹ Department of Computer Science, University of Warwick, Coventry CV4 7ES, UK

² College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China

Otherwise, the execution of the parallel job will be delayed due to failed communications. For example, the performance of MPI job is severely dependent on the efficiency of communication. However, credit scheduler, the default scheduling strategy of Xen, was initially designed for scheduling serial jobs, which is considered to be less capable of appropriately allocating pCPUs to the vCPUs that are hosting a parallel job.

This paper presents vChecker, a co-scheduler which improves the ability of scheduling parallel job of the credit scheduler. Our co-scheduler is implemented as an application-level co-scheduler which requires no modification to the hypervisor of Xen. Comparing our previous work in [4], vChecker considers the time period of scheduling a parallel job, which achieves better fairness in terms of resource allocation. The experimental results show that the performance of the parallel job is significantly improved. Moreover, the result indicates that the utilization of the system is also optimized.

The rest of paper is organized as follows: Sect. 2 reviews the state-of-the-arts in the field and provides some background. Section 3 states the problem and analyses it. The design of co-scheduler is described in Sect. 3.2. Then we will explain the implementation details of the prototype of vChecker in Sect. 4. Section 5 shows the experimental result and evaluation. A conclusion is given in Sect. 6.

2 Background and related work

2.1 Background

The credit scheduler is a proportionally fair sharing scheduling algorithm, is the default scheduler of Xen. Each domain has two values weight and cap, which defines the share of execution time the domain will be given. Weight indicates the relative proportion of CPU time, and cap defines the upper limit of execution time. Each domain is given credits decided by its weight value, and a domain's credits are evenly assigned to its vCPUs. Credits will be consumed as long as the vCPU gets executed on a pCPU and subtracted according to the execution time of the vCPU. Each scheduled vCPU has a *timeslice* of maximum 30ms to run, and a global value called *ratelimit* indicates the minimum time to occupy the pCPU. A vCPU is set to *UNDER* if there are remaining credits, and *OVER* priority is given to vCPU with negative credits. The credit scheduler re-calculate and replenish credits every 30 ms.

2.2 Related works

Xen has several schedulers which can be chosen by users according their practical needs. Simple Early Deadline

(SEDF) [8] is another scheduler option besides credit scheduler, which is less popular in practice due to lack of global balancing ability. Real Time Deferrable Server (RTDS) [16] is a newly added scheduler for scheduling real-time applications.

Xen has been intensively studied in recent years, many efforts have been made to optimize the performance of Xen. Wu and et al. propose a scheduler to improve the utilization of pCPU of Xen host [18]. In their work, the credit scheduler is given knowledge of workload distribution, which enables resource allocation to be adjusted. Xi and et al. proposes RT-Xen to improve the performance of the real-time application in Xen in terms of catching up the deadline [19]. Moreover, Xu and et al. develop vBOOST [14], vSlicer [21] and vTurbo [20] to mitigate the performance degradation of I/O intensive applications in Xen. Cheng and et al. propose vBlancer to optimize the I/O performance of SMP VM [1]. Some optimizations have been made to reduce the I/O responding time on UniProcessor virtual system [2, 5, 7]. Kim and et al. optimizes inter-domain communications in their work [6] while Huang and et al. improve the communications that are processed in the same host [3]. Furthermore, various studies of job scheduling were conducted [9–13, 15, 17, 22].

3 Strategy design

3.1 Problem discussion

The credit scheduler is initially designed for serial jobs, it has no knowledge of parallel jobs, which leads the particularity of the parallel job is ignored. The parallel job usually requires related nodes to be scheduled at the same time to exchange the intermediate result over internal communications. We assume that there are sufficient number of serial jobs co-running with the parallel jobs on the same host. It is highly possible that the execution of a parallel job will be negatively affected by the failed communications.

In traditional co-scheduling algorithm, the intuitive co-scheduler can be achieved by four steps. Firstly, current information of of Xen virtual system can be obtained by the co-scheduler. Secondly, the co-scheduler observes the VMs that host the running parallel job. If one of the parallel nodes is scheduled, the co-scheduler will de-schedule and pause the running VM to assure there are enough pCPU available for other nodes of a parallel job. Thirdly, all VMs that host the subtasks of a parallel job will be executed for a time slice. Finally, the normal serial jobs are resumed to the running state afterwards. However, critical problem may exist in this strategy. The co-scheduler is only an

additional module to assist the credit scheduler in properly scheduling parallel jobs, it is necessary to respect principle of fairly scheduling of credit scheduler. However, this strategy disrupts the fairness of credit scheduler and the parallel job will be allocated too much resources, the performance of other VMs are severely affected, especially when a large parallel job is launch on the system.

Alternatively, we can let the vCPUs of a parallel job wait for enough CPU resources to be scheduled, and the parallel nodes will be scheduled and execute for a period where there are enough pCPUs. As the result, the allocated computational resource of serial jobs will not be affected. This strategy benefits from appropriately scheduling a parallel job while reducing the influence to the performance of sequential jobs. The currently available pCPUs will be monitored, however, the demand of parallel jobs will be ignored. The co-scheduler will check the available CPU, and schedule all parallel nodes once there are enough pCPUs. However, if there is not enough number of pCPUs, the parallel job may have to wait enough pCPUs available.

Both strategies mentioned above have significant drawback which may severe negative impacts on the performance of the virtualized system. Therefore, a compromised strategy is required to consider both the particularity of the parallel job and the fairness of scheduling.

3.2 Design of the co-scheduler

The compromised strategy observes the number of available pCPUs while takes the waiting time of the parallel jobs into account. In this design, we assume the VMs are uniprocessor system, which is normal setup of parallel job in practice. As the parallel job will achieve the best performance when all vCPUs of subtasks of the parallel job are simultaneously scheduled online, we only allow the vCPUs of subtasks of a parallel job to be scheduled together to avoid performance degradation of the parallel job, namely, single subtask will not be scheduled alone. Moreover, We use a threshold of available pCPUs to indicate the time when the serial job should sacrifice its

execution time for the parallel job. That is, if the available resource reaches the pre-defined threshold, the co-scheduler will execute all subtasks of the parallel job by pausing/resuming VMs operations. For example, if a parallel job with 6 subtasks and the threshold is 5, the parallel job will be scheduled if there are 5 available pCPUs.

The co-scheduler considers how long the parallel job has been waiting for enough resource. We set a timer to guarantee the parallel job being scheduled within a certain period. We assess the current workload of the system to direct the maximum waiting time of the parallel job. Specifically, in the virtual system, each vCPU is given a timeslice to run each turn, if a vCPU is scheduled off, then it has to wait other vCPUs of the same queue to complete the execution, which can be quantified by using the average length of the run queue and the time slice setting in Xen.

4 Implementation

The implementation of this co-scheduler includes four modules: (1) the *Initializing module*, (2) the *System information collecting module*, (3) the *Detecting and monitoring module* and (4) the *Operating module*. The workflow is illustrated in Fig. 1. The detail of each module is given in the rest of the section.

4.1 Initializing

This module will prepare the tools by establishing the connection between the vChecker and management tools of Xen. Firstly, the vChecker will be connecting to the Xen hypervisor for using the APIs of virtualized system. The global system information of the Xen host can be retrieved from the APIs. Secondly, the *XI* package, which is a set of Xen management functions, will be prepared. Thirdly, *libvirt* interface connection is established.

The function *init()* actually implements the initializing operations. When the vChecker is called to run, *init()* will be called at once. To establish connection to the virtual

Fig. 1 The workflow of the vChecker

system, we need to call *virtConnectOpenReadOnly()* to prepare the *libvirt*. Moreover, as some information of the virtual machines are privilege to the *Xl* package, we need to call *xc_interface_open()* to create the connection to *Xl*.

4.2 System information collecting

Most of the information in this module is static information. For instance, amount of pCPUs of the physical host, amount of subtasks of the parallel job. This module require two arguments: (1) the threshold of available pCPUs and (2) the list of virtual machines which host the subtasks of the parallel job. Base on the provided information, the size of the parallel job can be calculated. The algorithm is given in Algorithm 1.

Algorithm 1: Information collecting function

Input: None
Output: None

```

1 if (threshold == NONE) || (parallel_doms == NONE) then
2   threshold = INPUT;
3   parallel_doms = INPUT;
4 end
5 for dom NOT IN parallel_doms do
6   Add to serial_doms;
7 end
8 activeDomstotal = virtConnectNumOfDomain();
9 Ptotal = virtNodeGetInfo();
10 queue_length = activeDomstotal / Ptotal;
11 waiting_time = (queue_length - 1) * timeslice;
12 Goto monitoring();
```

In this module, the function *vitNodeGetInfo()* of *Libvirt* is called to retrieve the information of the virtual machines. Moreover, the amount of *active* domains can be obtained by calling *virtConnectNumOfDomian()* provided by *Libvirt*. For the other system information we need in the runtime, we calculate based on these two values, for example, amount of serial jobs.

4.3 Monitoring

The monitoring module observes the execution process of the virtual machines. In this module, the information of the virtual system will be repeatedly evaluated in the runtime, which includes the number of available pCPUs and the waiting time of the parallel job. At the beginning of this module, a timer is set for scheduling the parallel job. As explained above, the number of the active domains can be obtained by calling *virtConnectNumOfDomian()*, and the total number of the pCPUs of the host can be got by calling *vitNodeGetInfo()*, thus, the length of the run queue can be obtained. The waiting time of the parallel job can also be obtained, which will be used to set the timer. After that, a loop monitors the available pCPUs and check if the threshold is reached. The number of available pCPUs

requires the information of the number of running serial jobs, and this module repeatedly checks the running status of the serial jobs. If the threshold is reached, then the operation module will be called to perform a scheduling of the parallel job. Otherwise, the loop will keep checking until the timer calls the operation module. We call *xc_domain_getinfo()* of *Xl* to get the list of virtual machines. *xc_dominfo_t* structure is used to store the domain information. The algorithm is given in Algorithm 2.

Algorithm 2: Monitoring function

Input: None
Output: None

```

1 time = now();
2 set_timer(time, waiting_time);
3 count = 0;
4 while true do
5   for dom IN serial_doms do
6     if dom.running == 1 then
7       count++;
8     end
9   end
10  Pavailable = Ptotal - count;
11  if Pavailable ≥ threshold then
12    break;
13  end
14 end
15 Goto operating();
```

4.4 Operating module

In the operation module, we use *xc_dmain_pause()* and *xc_dmain_resume()* functions provided by *Xl* package to control the progress of the parallel jobs and the serial jobs. When this module is called, it will pause the running domains first, and then resume the parallel nodes to running state. As the all serial jobs are paused, the host is fully occupied by the parallel job, and the credit scheduler will allocated these parallel nodes to different pCPUs according to the principle of workload balancing. The parallel nodes will be permitted to run simultaneously for a time slice by counting the running time. The parallel job will be allowed to executed for a time slice, then the co-scheduler will pause the parallel VMs and resume those serial jobs to execute. After that, the co-scheduler goes back to the system information collecting module.

5 Performance and evaluation

We conduct the experiments on a host which has a quad-core CPU (Intel Core I7-3820 CPU @ 3.60GNz) and 16GB RAM. The Xen 4.2.0 is adopted as the virtualization

Fig. 2 NPB performance of a four nodes parallel job

platform, and the guest operating system is Ubuntu 14.04. We create a total of 32 uniProcessor virtual machines for the experiment, namely, each VM has only one vCPU. These VMs will be randomly allocated to the cores, apart from vCPUs of dom0, each pCPU will be assigned to serve 4 vCPUs.

5.1 Evaluations on performance of the parallel job

In this experiment, we choose NAS Parallel Benchmark Suite to evaluate the performance of vChecker. In this experiment, we randomly choose some VMs as PVMs to execute the parallel workload in a hybrid execution environment. We use bash to generate serial workloads on non-parallel VMs. Additionally, we add periodic communications between parallel node to simulate the servers in real world. We set the problem size to C and run all benchmark programs using 4 nodes for five times. The average results which are normalized to native performance are shown in Fig. 2.

The experimental result shows that vChecker is able to enhance the performance of communication-intensive parallel workloads. As can be seen from the Fig. 2, the vChecker reduces the execution time for IS (15%) LU (10.6%) FT (6.7%), these testing workloads are communication-intensive programs which use massive *AlltoAll* primitive, *AllReduce* and *PointtoPoint* primitives. However, the performance of computation-intensive parallel workload is decreasing, the performance of BT drops 8.7% for example. This is due to the fact that parallel nodes spend extra time in waiting to be scheduled together.

In this experiment, we use the number of communications per unit time (NCUT) to evaluate the resource utilization. We record the number of parallel job related I/O operations processed by *dom0* and combine it with the execution time to calculate NCUT. This metric is more accurate because it reflects the real resource utilization rather than the load of CPU. As can be seen from Fig. 3, the utilizations for IS and LU increase by 38% and 17% respectively, while resource utilizations for other workloads are also slightly improved.

Fig. 3 The number of communications per unit time (NCUT)

6 Conclusion

In this paper, we have presented an application-level co-scheduler designed for improving the performance of the parallel job in Xen. This strategy considers the waiting time of parallel job on one hand, and takes the number of available pCPUs into account on the other hand. Moreover, we have implemented a co-scheduler named vChecker based on the co-scheduling strategy. The experimental results shows that the vChecker dramatically enhances the performance of the parallel job in Xen by reasonably assigning pCPUs to the sub-tasks of the parallel in runtime. The utilization of CPU is also optimized by effective communications between parallel nodes.

Acknowledgements This work is partially supported by the National Key R&D Program of China 2018YFB1003201, Worldwide Byte Security Information Technology Ltd and Guangdong Pre-national Project 2014GKXM054.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

- Cheng, L., & Wang, C. L. (2012). vbalance: Using interrupt load balance to improve i/o performance for smp virtual machines. In *Proceedings of the Third ACM Symposium on Cloud Computing* (p. 2). ACM.
- Cheng, L., & Wang, C. L. (2013). Network performance isolation for latency-sensitive cloud applications. *Future Generation Computer Systems*, 29(4), 1073–1084.
- Huang, W., Koop, M. J., & Panda, D. K. (2008). Efficient one-copy mpi shared memory communication in virtual machines. In *IEEE International Conference on Cluster Computing, 2008* (pp. 107–115). IEEE.
- Jiang, P., & He, L. (2017). Developing co-scheduling mechanisms for virtual machines in clouds. In *2017 14th International Symposium on Pervasive Systems, Algorithms and Networks 2017 11th International Conference on Frontier of Computer Science and Technology 2017 Third International Symposium of Creative Computing (ISPAN-FCST-ISCC)* (pp. 257–264). <https://doi.org/10.1109/ISPAN-FCST-ISCC.2017.90>.
- Kim, H., Lim, H., Jeong, J., Jo, H., & Lee, J. (2009). Task-aware virtual machine scheduling for i/o performance. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (pp. 101–110). ACM.
- Kim, K., Kim, C., Jung, S. I., Shin, H. S., & Kim, J. S. (2008). Inter-domain socket communications supporting high performance and full binary compatibility on xen. In *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (pp. 11–20). ACM.
- Lee, M., Krishnakumar, A. S., Krishnan, P., Singh, N., & Yajnik, S. (2010). Supporting soft real-time tasks in the xen hypervisor. In *ACM Sigplan Notices* (Vol. 45, pp. 97–108). ACM.
- Leslie, I. M., McAuley, D., Black, R., Roscoe, T., Barham, P., Evers, D., et al. (1996). The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7), 1280–1297.
- Li, K., Liu, C., Li, K., & Zomaya, A. Y. (2016). A framework of price bidding configurations for resource usage in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 27(8), 2168–2181. <https://doi.org/10.1109/TPDS.2015.2495120>.
- Li, K., Mei, J., & Li, K. (2018). A fund-constrained investment scheme for profit maximization in cloud computing. *IEEE Transactions on Services Computing*, 1–1. <https://doi.org/10.1109/TSC.2016.2589241>.
- Li, K., Tang, X., & Li, K. (2014). Energy-efficient stochastic task scheduling on heterogeneous computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 25(11), 2867–2876. <https://doi.org/10.1109/TPDS.2013.270>.
- Li, K., Tang, X., Veeravalli, B., & Li, K. (2015). Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. *IEEE Transactions on Computers*, 64(1), 191–204. <https://doi.org/10.1109/TC.2013.205>.
- Liu, C., Li, K., Xu, C., & Li, K. (2016). Strategy configurations of multiple users competition for cloud service reservation. *IEEE Transactions on Parallel and Distributed Systems*, 27(2), 508–520. <https://doi.org/10.1109/TPDS.2015.2398435>.
- Ongaro, D., Cox, A. L., & Rixner, S. (2008). Scheduling i/o in virtual machine monitors. In *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (pp. 1–10). ACM.
- Tang, Z., Ma, W., Li, K., & Li, K. (2018). A data skew oriented reduce placement algorithm based on sampling. *IEEE Transactions on Cloud Computing*, 1–1. <https://doi.org/10.1109/TCC.2016.2607738>.
- Tran, G. P. C., Chen, Y. A., Kang, D. I., Walters, J. P., & Crago, S. P. (2016). Hypervisor performance analysis for real-time workloads. In *High Performance Extreme Computing Conference (HPEC), 2016 IEEE* (pp. 1–7). IEEE.
- Wang, Y., Li, K., Chen, H., He, L., & Li, K. (2014). Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints. *IEEE Transactions on Emerging Topics in Computing*, 2(2), 134–148. <https://doi.org/10.1109/TETC.2014.2300632>.
- Wu, J., Wang, C. Y., & Li, J. F. (2016). La-credit: A load-awareness scheduling algorithm for xen virtualized platforms. In *IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), 2016* (pp. 234–239). IEEE.
- Xi, S., Wilson, J., Lu, C., & Gill, C. (2011). Rt-xen: Towards real-time hypervisor scheduling in xen. In *Proceedings of the Ninth ACM International Conference on Embedded Software* (pp. 39–48). ACM.
- Xu, C., Gamage, S., Lu, H., Kompella, R. R., & Xu, D. (2013). vturbo: Accelerating virtual machine i/o processing using designated turbo-sliced core. In *USENIX Annual Technical Conference* (pp. 243–254).
- Xu, C., Gamage, S., Rao, P. N., Kangarlou, A., Kompella, R. R., & Xu, D. (2012). vslicer: Latency-aware virtual machine scheduling via differentiated-frequency cpu slicing. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*.
- Zhang, L., Li, K., Xu, Y., Mei, J., Zhang, F., & Li, K. (2015). Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster. *Information Sciences*, 319, 113–131. <https://doi.org/10.1016/j.ins.2015.02.023>, <http://www.>

[sciencedirect.com/science/article/pii/S0020025515001231](https://www.sciencedirect.com/science/article/pii/S0020025515001231), energy Efficient Data, Services and Memory Management in Big Data Information Systems.

Peng Jiang is a PhD student in the Department of Computer Science at the University of Warwick. His research area is Cloud computing, and parallel and distributed computing.

Ligang He received the BSc and MSc degree from Huazhong University of Science and Technology, China, and received the PhD degree in Computer Science at the University of Warwick, United Kingdom. After PhD studies, he worked as a post-doctoral researcher at the University of Cambridge, UK, and then joined in the Department of Computer Science at the University of Warwick as Assistant Professor.

He is now the Associate Professor in the department. His research interests focus on parallel and distributed processing and Cloud computing. He has published more than 100 papers in international conferences and journals. He has been a co-chair or a member of the program committee for a number of international conferences, and been the reviewers for many international journals. He is a member of the IEEE.

Shenyuan Ren is a PhD student in the Department of Computer Science at the University of Warwick, UK. Her research area is parallel and distributed computing, and Cloud computing.

Zhiyan Chen is a PhD student in the Department of Computer Science at the University of Warwick, United Kingdom. His research area is mobile cloud computing.

Rui Mao received BS (1997) and MS (2000) in computer science from the University of Science and Technology of China, and MS (2006) in Statistics and PhD (2007) in Computer Science from the University of Texas at Austin. After three years work at the Oracle USA Corporation, he joined Shenzhen University in 2010. He is now Associate Dean of the College of Computer Science and Software Engineering, Shenzhen University, and Executive Director of

the Shenzhen Institute of Computing Sciences.