A Virtualization Platform Designed for Irregular Multi-Process Applications

Guangli Dai, Pavan Kumar Paluri, Albert Mo Kim Cheng, Panruo Wu
{gdai,pvpaluri,amcheng}@uh.edu,pwu6@central.uh.edu
University of Houston
Houston, Texas, USA

ABSTRACT

Many High-Performance Computing applications involve multi-process programming. While multiple processes running in parallel can greatly reduce the completion time of applications, there are many irregular applications where the workloads on different processes may vary significantly and thus are not balanced. An irregular application is unbalanced in the sense that processes with smaller workloads have to be suspended to wait for processes with larger workloads when each process is allocated a sole CPU. Since such unbalanced workloads can lead to a tremendous waste of computational resources, we propose the RRP-Xen virtualization platform for these irregular applications. Specifically, RRP-Xen can cluster multiple processes with small workloads from one or more irregular applications onto the same CPU. Compared to other HPC cloud solutions, which also involve virtualization, RRP-Xen can offer real-time performance guarantees to each process. Consequently, the number of CPUs needed for deploying multiple irregular applications can be reduced while the completion of each irregular application experiences only minor delays.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

ACM Reference Format:
Guangli Dai, Pavan Kumar Paluri, Albert Mo Kim Cheng, Panruo Wu. 2018. A Virtualization Platform Designed for Irregular Multi-Process Applications. In *ICPP 2021.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Among 13 types of HPC applications summarized by [1], most follow some form of parallelism. Thus, the application execution can be greatly accelerated by employing multi-process programming. However, executing applications in parallel involving multiple processes leads to another challenge for irregular applications [5]: the workload on each process can hardly be strictly balanced. This means some processes with small workloads will be idled before they can resume execution, waiting for the results from other processes with larger workloads. Such a challenge is a common bother in the following types of applications: (1) Unstructured Grids, (2) Sparse Linear Algebra, (3) Graph Traversal, (4) Backtrack and Branch + Bound, and (5) Construct Graphical Models. Unbalanced processes in irregular applications result in longer completion time and wastage of computational resources if each process is allocated a dedicated core.

While it is hard to give a general solution to balance the workloads in different processes of irregular partitions, virtualization is a practical technique that can be applied to reduce the computational resource wastage with minimal latency on the completion time of each application. The intuitive idea is to deploy multiple applications on the same set of computational resources and allow the processes with smaller workloads to share the same core so that the core will not be idled. In this way, with the same number of CPUs in the system, more applications submitted can be executed at the same time with minimal latency added to their completion time so that the resource utilization efficiency is greatly increased.

When deployed on a virtualized platform, each process will be bound with a Virtual CPU (vCPU). A scheduler inside the hypervisor will schedule the vCPUs accordingly. Though there has been research on virtualized HPC platforms, especially on resource allocations, most previous studies do not offer real-time performance guarantees to the vCPUs. These guarantees are critical to processes in an irregular application; Without them, a process with a smaller workload may still be running while other processes are already done. This would increase the completion time of the application. Besides, the real-time performance guarantees must still be valid during reconfiguration, i.e., when new applications are added into the system, to increase the usability of the system.

Based on the two requirements, we propose a solution for the deployment of irregular applications deployment based on the popular hypervisor Xen [2].

2 RRP-XEN

RRP-Xen is an implementation of the Regularity-based Resource Partitioning (RRP) model on the popular Xen hypervisor [4]. The RRP model is built based on the cyclic scheduling approach specified by the ARINC 653 standard, which is designed for real-time applications in virtualized environments. The scheduler strictly follows a cyclic schedule generated in the user space to schedule Virtual Machines (VMs). In short, RRP-Xen includes three major modifications compared to the original Xen, as presented in Fig. 1: (1) The RRP-Toolset in the user space is responsible for generating the schedule of vCPUs; (2) The hypercall will transfer the cyclic schedule into the hypervisor kernel for the scheduler to access;

ICPP 2021, Aug-09, 2021, Virtual

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

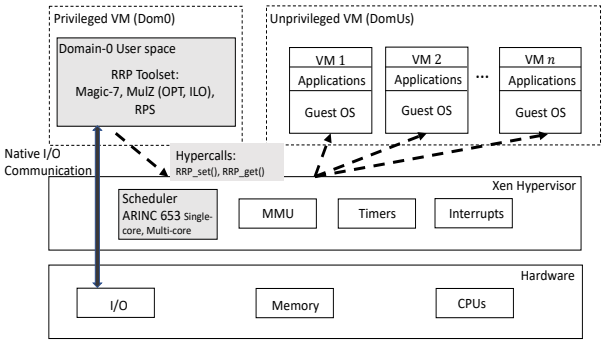

Figure 1: The architecture of RRP-Xen.

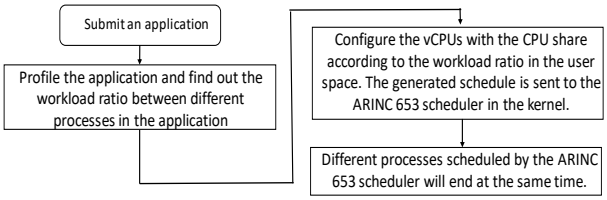

Figure 2: A workflow of the solution based on RRP-Xen

(3) The ARINC 653 scheduler follows the cyclic schedules sent in on the corresponding CPUs to schedule the vCPUs.

Compared to traditional schedulers, RRP-Xen has three major advantages: (1) By following a pre-generated schedule, the scheduling time is very low and thus minimizes the temporal overhead caused by scheduling. (2) With the pre-generated schedule, the resource supply (execution time a vCPU can obtain during a certain period) is highly predictable with real-time performance guarantees. (3) When applications are added into or removed from the system, a new schedule can be generated in the user space and loaded into the hypervisor kernel to reconfigure the scheduler with real-time performance guarantees preserved meanwhile [3].

RRP-Xen can offer real-time performance guarantees to each vCPU even with reconfiguration requests arriving online, exactly fitting the requirements for deploying irregular applications. The current version of RRP-Xen only considers one vCPU in each Virtual Machine (VM). For HPC applications, we must modify the current RRP-Xen to handle multiple vCPUs in the same VM since each application (inside a VM) has multiple processes (vCPUs).

With the implementation of RRP-Xen, the resource supply is going to be stable as long as the system is properly configured. To configure the system, we present a workflow in Fig. 2. Specifically, for a submitted application, the system needs to profile the application using tools like mpiP [6] to determine the workload ratio between different processes and configure the vCPUs accordingly. RRP-Xen will then take over the execution following the configuration. When setting the CPU share each vCPU needs, we can assign a whole CPU, i.e., CPU share 1, to the vCPU with the largest workload. The value of the CPU share for other vCPUs can be calculated using their workload divided by the largest workload.

3 EVALUATIONS

Since the support of multiple vCPUs in each VM and the online reconfiguration on RRP-Xen is still under development, we present the benchmarking results of RRP-Xen to validate its real-time performance with each VM (domain) including only one vCPU. Specifically, the metrics we benchmark include: (1) Scheduling Latency: the time spent in the scheduler during 1 second. This metric indicates the temporal cost of a scheduler. (2) Number of Context Switches: the number of context switches between domains during 1 second. This metric reveals the time wasted in saving and loading contexts. (3) Redis-Cli scheduling delay: the end-to-end delay for the Redis-server deployed on a certain domain.

Table 1: Scheduling Overhead Measurements for a 1 second Tracing period with 2 Idle domains.

	RRP	RTDS	Credit
Scheduling Latency	1.4ms	1.66ms	4.972ms
Number of Context Switches	67	85	71

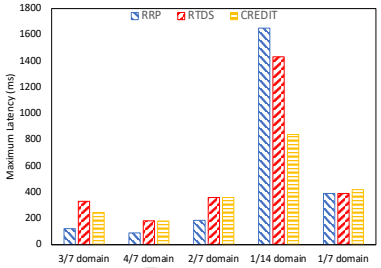

Figure 3: Maximum scheduling delay measured by Redis-Cli of different schedulers.

Table 1 shows the benchmarking results of the first two metrics. When compared with other conventional schedulers on Xen, i.e., RTDS and Credit. Note that RTDS is also a real-time scheduler employing dynamic queues. RRP-Xen spends the least time in making scheduling decisions and in saving and load contexts. This makes the performance of RRP-Xen more predictable with real-time performance guarantees. Fig. 3 shows the performance of RRP-Xen against the other two schedulers in Xen measured by Redis-Cli. The maximum scheduling delay of RRP-Xen is shorter or no longer than others in relatively large-share domains, i.e., 3/7, 4/7, and 1/7.

REFERENCES

[1] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. 2006. The landscape of parallel computing research: A view from berkeley. (2006).

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the art of virtualization. In *ACM SIGOPS operating systems review*, Vol. 37. ACM, 164–177. https://doi.org/10.1145/1165389.945462

[3] Wei-Ju Chen, Peng Wu, Pei-Chi Huang, Aloysius K Mok, and Song Han. 2019. Online Reconfiguration of Regularity-Based Resource Partitions in Cyber-Physical Systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 495–507. https://doi.org/10.1007/s11241-021-09364-5

[4] Pavan Kumar Paluri, Guangli Dai, and Albert M. K. Cheng. 2021. ARINC 653-Inspired Regularity-based Resource Partitioning on Xen. In *The 22nd ACM Conference on Languages, Compilers, and Tools for Embedded Systems*.

[5] Leonardo Solis-Vasquez, Diogo Santos-Martins, Andreas F Tillack, Andreas Koch, Jérôme Eberhardt, and Stefano Forli. 2020. Parallelizing Irregular Computations for Molecular Docking. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*. IEEE, 12–21.

[6] Jeffrey Vetter and Chris Chambreau. 2005. mpip: Lightweight, scalable mpi profiling. (2005).