Improving hardware resource allocation for networked applications using unikernels

Jiangqiong Liu
liu3328@purdue.edu
Purdue University
United States

Joshua Reprogle
jreprogl@purdue.edu
Purdue University
United States

Vedaant Rajoo
vrajoo@purdue.edu
Purdue University
United States

Sowmya Jayaram Iyer
jayarami@purdue.edu
Purdue University
United States

Keerthana Ashokkumar
kashokku@purdue.edu
Purdue University
United States

Anmol Sahoo
Purdue University
United States
sahoo9@purdue.edu

ABSTRACT

In this project, we intend to use unikernels to improve the performance of networked applications. Since networked applications contend heavily for scarce CPU resources such as caches, memory bandwidth and compute units, precise hardware resource allocation is difficult to perform for each application. Thus, we propose to use unikernels which only bring the minimal amount of code required to run the application and hope to demonstrate that this would improve resource utilization in the processor. We intend to use the Unikraft unikernel framework and deploy applications on the Firecracker VMM. We will implement our resource allocation policy in Firecracker and use Unikraft to develop some representative workloads across machine learning, distributed systems and control plane applications and then present the effectiveness of our results.

ACM Reference Format:
Jiangqiong Liu, Joshua Reprogle, Vedaant Rajoo, Sowmya Jayaram Iyer, Keerthana Ashokkumar, and Anmol Sahoo. 2018. Improving hardware resource allocation for networked applications using unikernels. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

[1].

2 BACKGROUND AND MOTIVATION

3 PROPOSED SOLUTION

To mitigate the problems that we discussed in Section 2, we develop on two key insights - (1) intra-application cache partitioning and (2) using unikernels to remove OS contention. We discuss these two points in further detail -

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Intra-application cache partitioning. Networked applications contend for cache space within different parts of the application. This is because

Using unikernels to remove OS contention. An Operating System (OS) is responsible for providing an abstraction of the CPU for applications. It maintains multiple in-memory structures for providing various services such as scheduling, virtual memory, device IO and others. While this is necessary for enabling multi-user interactive systems (such as terminals), running specialized applications such as networking services do not require these features. Further, applications running on hypervisors face a duplication in abstraction layers, because the same abstractions are provided at both OS and hypervisor layer (though this can be offset with hardware acceleration such as KVM and SRIOV). But, the most important problem with an OS in the context of intra-application cache partitioning is that it renders any analysis of the application useless. When an application switches into a cache partition, it maybe preempted by the OS at any point of time. Now, the OS needs to switch its cache partition but it may still interfere with the memory buffers of the application (through cache flushes). Thus, even an effective intra-application partitioning scheme would not be useful in the presence of an OS.

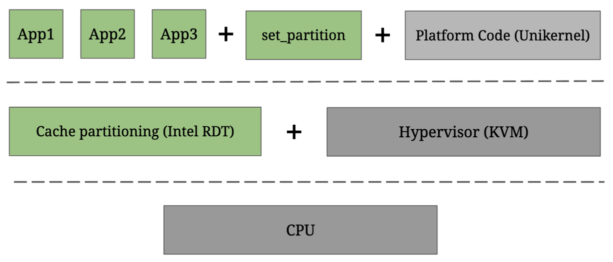

Figure 1: Solution architecture

4 EVALUATION AND RESULTS

The evaluation of our implementation requires careful analysis because there are two competing effects at play. One one hand, partitioning cache space for the networking parts of the code may reduce the cache space available for the remaining parts of the application. Thus, even though the performance of the networking parts improves the overall application performance may suffer. Moreover, since our dynamic cache partitioning scheme is implemented using hypercalls in an hypervisor, even this may lead to an increase in runtime. Given this, it would not be possible to determine the impact of our work by simply measuring wall-clock time. Thus we use hardware performance counter events to benchmark different aspects of an application which we believe capture the performance of our implementation. We detail the different hardware events we capture -

PCIE Writes / Reads from LLC. This is the primary event that identifies if our solution is working. This event provides information about the number of PCIE accesses that were serviced from the LLC. This is important because if we are servicing more requests from the LLC, this implies that more network packets are staying resident in the cache and directly being processed from there, instead of being retrieved from main memory.

LLC Misses. This is the second event that can be used to capture the

Finally, we also measure application level benchmarks such as bandwidth and latency of requests served, since finally any improvement in the software should result in an observed improvement in the client side performance. We present our results and discuss our observations below.

5 CONCLUSION AND FUTURE WORK

ACKNOWLEDGMENTS

To Prof. Muhammad Shahbaz for conducting such a great course and Sharuna Anandraj, Darwin Kamanuru, K M A Solaiman and Kai Ling for helping out throughout the semester!

REFERENCES

[1] Alireza Farshin, Amir Roozbeh, Gerald Q. Maguire, and Dejan Kostić. 2020. Reexamining Direct Cache Access to Optimize I/O Intensive Applications for Multi-Hundred-Gigabit Networks. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'20)*. USENIX Association, USA, Article 46, 17 pages.

A CONTRIBUTIONS

In this section, we provide the details of the contributions of each author.

A.1 Jiangqiong Liu

A.2 Joshua Reprogle

A.3 Vedaant Rajoo

A.4 Sowmya Jayaram Iyer

A.5 Keerthana Ashokkumar

A.6 Anmol Sahoo