Exploration of applications running on a ported JVM on top of Tessellation

Benjamin Le Jefferson Lai Wenxuan Cai John Kubiatowicz*

{benjaminhoanle, jefflai2, wenxuancai}@berkeley.edu, kubitron@cs.berkeley.edu
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720-1776

ABSTRACT

Java applications run within Java Virtual Machines (JVM). As an application runs, the JVM performs many parallel background tasks for general housekeeping reasons including garbage collection and code profiling for adaptive optimization. While this design works well to provide isolation when there is a single or small number of Java applications running on a single machine, in practice it is common to find a large number of Java applications running concurrently on a single machine. For example, a machine could be running multiple instances of HDFS, Hadoop, and Spark, simultaneously, with each instance having an associated JVM. As all of these JVMs must ultimately be multiplexed onto a single set of hardware, interference among the large set of parallel tasks arise. There is little published literature documenting the causes of this interference or how to deal with it. In this paper, we determine the specific sources of these interferences and show how running these applications on top of a Tessellation-integrated Xen hypervisor addresses these issues and reduces interference. We evaluate the performance of these âĂIJTessellatedâĂİ machines in comparison with machines running bare Linux when running a large number of JVMs.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management – Scheduling; D.4.8 [**Operating Systems**]: Performance – Measurements, Monitors

General Terms

Multicore, parallel, quality of service, resource containers

*The Parallel Computing Laboratory, UC Berkeley, CA, USA

Keywords

Adaptive resource management, performance isolation, quality of service

1. INTRODUCTION

The Java Virtual Machine (JVM) abstraction provides a consistent, contained execution environment for Java applications and is a big part of why Java has enjoyed so much popularity and success in recent years.

By handling concerns about the lower level kernel and operating system, the JVM allows developers to write an application once and benefit from high portability to a number of different environments and architectures.

There are number of different JVM specifications and implementations maintained both privately and publicly. While we refer to <PAPERs> for a deeper discussion of these variations, we summarize the general responsibilities most of them have in common.

<VERY BRIEF SUMMARY JVM responsibilities (JIT compiling and GC)> <BACKGROUND TASKS THAT CAN INTERFERE WITH ONE ANOTHER>

In this paper, we focus on the HotSpot JVM <REFERENCE> and investigate the effect of ARCC Tesselation on this interference. We show that <summarize findings>

The rest of this paper is organized as follows.

As applications run, the JVM performs many parallel background tasks for general housekeeping reasons including garbage collection and code profiling for adaptive optimization. While this design works well to provide isolation when there is a single or small number of Java applications running on a single machine, in practice it is common to find a large number of Java applications running concurrently on a single machine. For example, a machine could be running multiple instances of Hadoop File System (HDFS), Hadoop, and Spark, simultaneously, with each instance having an associated JVM. As all of these JVMs must ultimately be multiplexed onto a single set of hardware, interference between the large set of parallel tasks arises.

There is little published literature documenting the causes of this interference or how to deal with it. In this paper, we determine the specific sources of this interference and show how running these applications on top of a Tessellation-integrated Xen hypervisor addresses these issues and reduces interference. We evaluate the performance of these âĂIJTessellatedâĂİ machines in comparison with machines running bare Linux when running a large number of JVMs. We conclude with insights gained from our experiments and areas

Table 1: Standard Deviations of Dacapo Benchmarks

Benchmark	Standard Deviation	Variance
avrora	1000	100

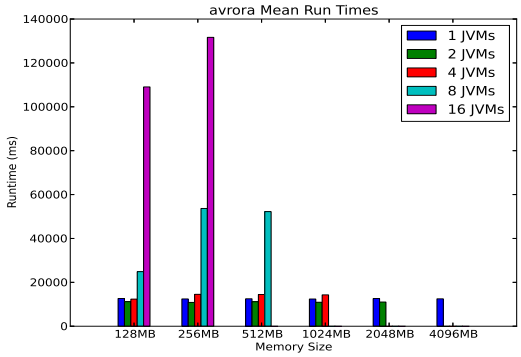

Figure 1: A sample graphic (.eps format) that has been resized with the `epsfig` command.

for future work.

The remainder of this paper is organized as follows. Section 2 describes the systems we are working with, including Xen, Tesselation, and OSV. Section 3 describes our experimental results. We discuss directions for future work in Section 4 and conclude in Section 5.

Baseline OSV, OSV with Gang scheduling.

Make sure to talk about how Tesselation fits in with Xen for Kubi.

1-32 JVMs, Max Heap Sizes: 1-256

2. BACKGROUND

2.1 HotSpot

<SUMMARY OF HOTSPOT (specifics of how HotSpot handles JIT compilation and GC)>

2.2 ARCC

2.3 Tesselation

3. DACAPO

3.1 Experimental Setup

3.2 Results

4. YCSB

4.1 Experimental Setup

4.2 Results

5. CONCLUSION

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

6. ACKNOWLEDGMENTS

Martin Nathan Eric

7. REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

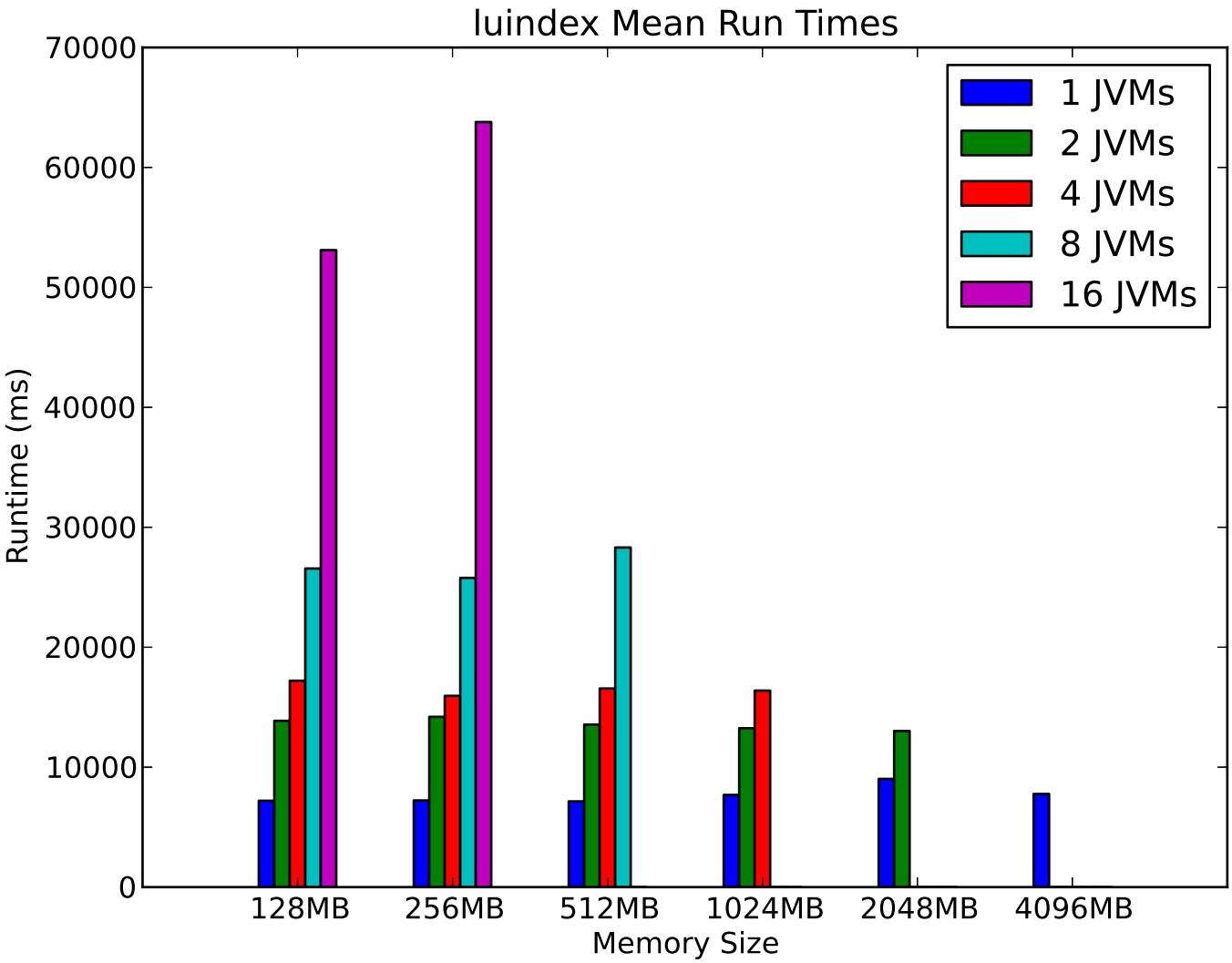

Figure 2: A sample black and white graphic (.eps format) that needs to span two columns of text.