Software Performance Engineering in the Cloud

Charles E. Leiserson William Hasenplaugh Tim Kaler Bradley C. Kuszmaul
Maryam Dehnavi Ekanathan Palamadai Natarajan Tao B. Schardl

Supertech Research Group
MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA 02139

Abstract

Cloud computing motivates scientists and others with compute-intensive workloads to demand efficient software. Efficient program code consumes less energy and stretches cloud-computing budgets farther. The cloud should be a good environment for the performance-engineering of compute-intensive workloads, but that potential is as yet unrealized, because cloud servers restrict the ability of performance engineers to observe and control of important hardware and system features.

This paper advocates that clouds support accurate performance measurements for applications. Specifically, the cloud must allow performance engineers to observe and control key system-performance features which otherwise adversely impact the variance of measured performance, often with systematic bias. An added benefit is that control of these key features will make the cloud well suited to performance-engineering techniques such as autotuning, because the vast resources of the cloud will allow many performance tests to be carried out in parallel.

1. Introduction

Cloud computing [1, 3] — the massive agglomeration of multicore servers on which computing time can be rented— has revolutionized the hosting of Internet content. A cloud service can amortize system maintenance, freeing the content provider from ongoing administrative costs. Resources can be dynamically reallocated based on demand, allowing the content provider to "buy computation by the yard".

An emerging use of clouds is for compute-intensive workloads, which are typical in computational science, real-world modeling, machine learning, in-memory databases, image and video processing, and a host of other applications across a variety of domains. Although high-performance computing (HPC) often commands attention with its massive 100,000-processor supercomputers performing important calculations, the sweet spot for computing remains the shared-memory multicore computer, the workhorse for cloud computing. Many compute-intensive applications can run at close to peak performance on a multicore due to its high communication bandwidth among processing cores, whereas obtaining more than a few percent of peak performance on a supercomputer is often difficult or impossible due to communication-bandwidth limitations. Since multicore computers are far more cost effective for scientific applications that fit within the multicore's memory, and clouds provide this bargain commodity in vast quantity, it makes sense to optimize around this sweet spot.

For instance, a scientist studying a real-world phenomenon, such as a molecular simulation, may develop a computer simulation on a multicore. In order to understand the effects of different parameters, such as physical constants or initial conditions, the scientist may wish to run the simulation many times, and now, clouds provide an attractive cost-effective alternative to an owned machine. By deploying the simulations on hundreds of cloud servers simultaneously, the scientist can quickly explore the influences of different parameter settings. The cloud is responsive. When the scientist needs to compute, the vast cloud resources produce answers quickly. When the scientist needs to think, sleep, eat spaghetti, etc., however, the cost for computing is zero. In contrast, the traditional alternative of buying a smaller dedicated computing system suffers under the scientist's bursty workload: the system is either underprovisioned — the scientist must wait a long time for all the computations to finish — or overprovisioned — the resource sits idle while the scientist tends to other things. Cloud computing allows the scientist to enjoy low response time by running many simulations simultaneously, yet pay nothing for the resource when there's no computing to do.

2. Software performance engineering

We contend that the economics of cloud computing — the pay-as-you-go model — will increase the importance of ***software performance engineering***, the theory and practice of making program code more efficient, sometimes called ***tuning***. Tuning an application on an owned multicore may yield better performance on a single execution, but whether it runs fast and sits idle or runs slow, the cost for computing remains the same, because the scientist has already invested in the capital cost of the machine. In contrast, since the cost of cloud servers depends on actual use, a scientist is constantly motivated to be more cost aware, especially for large deployments. Since fast code consumes fewer computing resources, efficiency pays off directly in dollar savings. Before deploying a large-scale simulation on 100 or 1000 cloud servers, for example, a scientist with a limited budget is well motivated to ensure that the application is efficient. Fast code is cheaper.

How do you make code run fast? Basically, performance-engineering methodology is just a simple iterative loop:

1. Measure the performance of Program A.
2. Make a change to Program A to produce a Program A'.
3. Measure the performance of Program A'.
4. If A' beats A, set $A = A'$.
5. If A is still not fast enough, go to Step 2.

Of course, many important steps have been left out of this tuning methodology, such as aggregating performance results across a test suite, testing the correctness of the modified Program A', etc., but our focus is on the performance-engineering aspects of the process.

3. Obtaining good measurements

Although this performance-engineering methodology looks simple, implementing it in practice is hard, even on dedicated machines, let alone on a shared cloud resource. The reason is that it is hard to measure performance, and as Lord Kelvin purportedly said, "If you cannot measure it, you cannot improve it." To obtain reliable measurements, sources of variability and bias must be controlled. The machine must be quiet with unnecessary daemons shut down. The impact of interrupts must be minimized, for example, by handling interrupts on a different processing core from the cores running the application under study. Threads must be placed on cores carefully to minimize variability due to hardware features, such as NUMA and hyperthreading. Features such as Turbo Boost and DVFS, which change the processor clock frequency depending on compute load and temperature, must be controlled. The programmer must be careful in the way the application is compiled and linked.

On a dedicated owned machine, these problems can be overcome, but it takes an experienced performance engineer to do so. On the surface, the cloud would seem to offer a nice opportunity in this space. Cloud providers could offer ***measurement machine instances*** in which exogenous variability due to features such as Turbo Boost and DVFS is controlled. Open-source ***measurement snapshots*** could be developed that control endogenous variability by shutting down all unnecessary daemons, executing applications on particular cores by default, etc.

Solving these issues is not hard technically. Our studies of virtual-machine environments on dedicated machines show that reliable performance numbers can be obtained with care to control exogenous and endogenous sources of variability. On the cloud, many sources of exogenous variability can be eliminated by simply using whole-machine instances. But controlling exogenous sources of variability that require access to the machine's BIOS or hypervisor cannot be done without the cloud provider's cooperation. *Cloud providers should offer measurement machine instances that make it easy to measure the performance of compute-intensive scientific applications.*

4. Autotuning

A side benefit of enabling tuning on the cloud is that the cloud now becomes a more reliable medium in which to do ***autotuning*** [2, 6] — optimizing the performance of a program by searching a space of program parameters for the optimal settings of those parameters. Our experiments with dedicated machines indicate that the quality of autotuning (i.e., the performance of the autotuned application) can be significantly enhanced with attention to accurate measurements. It should be no wonder that sloppy measurements produce suboptimal outcomes.

5. Simulation

A possible alternative to taking exact measurements is to simulate the system. Unfortunately, modern multicores are sufficiently complicated that accurate simulation can incur orders-of-magnitude slowdown in the execution of the application. Machine simulations can be useful tools for understanding performance bottlenecks, however. For example, cache simulators can show when memory bottlenecks might cause a slowdown in application performance, but they are generally useless for comparing the runtimes of two programs, because memory bandwidth is not the only contributor to application performance.

6. Conclusion

Performance engineering will become even more important after Moore's Law [5] ends. Moore's Law is the technological trend of shrinking semiconductor-device sizes which, for half a century, has approximately doubled the number of transistors on semiconductor chips every two years. Due to fundamental physics and the economics of silicon fabrication, semiconductor devices cannot continue to be shrunk forever. Respected technologists [4] predict significant attenuation of the trend by the end of the decade. After the demise of Moore's Law, one of the few ways to enhance the cost-effectiveness of computing will be to make legacy code more efficient and new code efficient to begin with. As the demand for more efficient code rises, cloud providers will have a competitive advantage if they offer systems in which careful measurements.

7. References

[1] Amazon Web Services. Announcing Amazon Elastic Compute Cloud (Amazon EC2) — beta [online]. 2006. Available from: `https://aws.amazon.com/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2---beta/`.

[2] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. OpenTuner: An extensible framework for program autotuning. pages 303–316. ACM, 2014.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, April 2010.

[4] Robert Colwell. The chip design game at the end of Moore's Law [online]. August 2013. Available from: `http://youtu.be/JpgV6rCn5-g`.

[5] Gordon E. Moore. Moore's Law at 40. In David C. Brock and Gordon E. Moore, editors, *Understanding Moore's Law: Four Decades of Innovation*, chapter 7, pages 67–84. Chemical Heritage Foundation, 2006.

[6] Samuel Webb Williams. *Auto-tuning Performance on Multicore Computers*. PhD thesis, University of California at Berkeley, 2008.