Accuracy and Responsiveness of CPU Sharing Using Xen's Cap Values

David Schanzenbach Henri Casanova

Information and Computer Sciences Dept.
University of Hawai'i at Mānoa, Honolulu, U.S.A.

Abstract

The accuracy and responsiveness of the Xen CPU Scheduler is evaluated using the "cap value" mechanism provided by Xen. The goal of the evaluation is to determine whether state-of-the-art virtualization technology, and in particular Xen, enables CPU sharing that is sufficiently accurate and responsive for the purpose of enabling "flexible resource allocations" in virtualized cluster environments.

1 Introduction

Over the last decade, commodity clusters have emerged as a corner stone technology for high-performance computing [1], data center operation [2], and network service hosting [3]. With the increase of computing power, comes a larger price tag in the form of power, cooling, space and administration cost. These all add greatly to the cost of maintaining a commodity cluster. With these high costs, *cluster utilization* is critical. Only high utilization can justify the expenditures associated with a large cluster [4, 5].

To increase utilization clusters are shared among users and application. Unfortunately, current cluster sharing strategies have limitations, some of which hinder utilization. Cluster sharing today is almost always done using *rigid* resource allocations (e.g., 10 cluster nodes, 15 virtual host slices). This forces applications to specify resource needs ahead of time. Furthermore, applications can be delayed until enough vacant resources are available, while currently running applications often do not make full use of the resources allocated to them. The consequences are lowered resource utilization and lowered application throughput. Furthermore, the use of rigid resource allocations makes it difficult to determine "good" allocations, i.e., in a view to optimizing some overall metric that quantifies performance and/or fairness as perceived by cluster users.

We claim that the use of virtualization technology can lead to much improved cluster utilization by allowing for "flexible resource allocations" (i.e., allocations that can be fractional and dynamically changed). A prerequisite for enabling flexible resource allocation is that CPU resources be shared in a way that is both accurate and responsive. In this paper, we focus on the Xen [6] virtualization technology and we attempt to quantify the *accuracy* and the *responsiveness* of the default CPU scheduler in Xen. We conduct series of experiments to evaluate these two metrics, using both single-VM instance and multi-VM instance scenarios, and using solely the "cap value" mechanism provided by Xen.

2 Background

2.1 Virtualization with Xen

Our goal is to evaluate qualitatively and quantitatively the CPU-sharing capabilities afforded by state-of-the-art virtualization technology. In this paper we use Xen [6], a VM monitor, or *hypervisor*. A hypervisor is "a virtualization platform that allows multiple operating systems to run on a host computer at the same time." There are two categories of hypervisors: (i) ones that run inside the operating system; and (ii) ones that

run between the hardware and the operating system. Xen falls in the second category, while other systems (e.g., Virtual PC [7], some versions of VMWare [8]) falls in the first category. As a result, Xen requires either a modified operating system, called "paravirtualization", or hardware support for virtualization, called "hardware virtualization" [9].

Xen provides several CPU schedulers via which different strategies for sharing compute resources among VM instance can be enacted. In this work we use the default scheduler, i.e., the Xen credit scheduler. Each VM instance is assigned a *cap* and a *weight*. The cap is a percentage that defines the maximum fraction of the CPU that can be used by the VM instance. When multiple VM instances contend for the CPU, then they are allocated CPU shares proportionally to their weights. In this paper we evaluate how accurate and reactive cap specifications are. We do not consider weights due to the following rationale. The broader objective of this work is to determine whether state-of-art virtualization technology, such as Xen, can be used to enforce precise and quickly adaptable resource allocations in a virtualized cluster environment. In this setting, a resource allocator would have full control over CPU shares allocated to VM instances and thus would only need to set their cap values. Contention among VM instances is therefore addressed by ensuring that the sum of the caps of all these instances is always lower than or equal to 100.

3 Accuracy of CPU Sharing with Xen

In this section, we evaluate the accuracy of Xen for sharing the CPU among VM instances based on cap values. We first study the case when a single VM instance is running, and then the case when multiple VM instances are running.

In all our experiments we use Xen 3.1 on a dual-proc 64-bit machine. All VM instances use identical images of a 64-bit version of Fedora 8. Each VM instance is allocated 700 MB of RAM, and all instances run on the same physical CPU. We restrict our study to the case of a single physical CPU. However, our experiments are controlled by an additional VM instance that runs on a separate physical processor to avoid interference with the VM instances that are part of the experiment.

The overall goal of our experiments is to measure the discrepancy between the cap specifications of and the effective CPU shares obtained by VM instances running under the Xen hypervisor. For now, we only consider CPU-bound VM instances. Each VM instance continuously executes a 100×100 double precision matrix multiplication using the LAPACK DGEMM routine [10]. We determine the effective CPU share obtained by a VM instance based on the rate at which it computes these matrix multiplications.

3.1 Single VM Instance Experiments

For these experiments we run a single VM instance for one hour using 11 cap values (1, 10, 20, . . ., 100), with 10 repeats for each cap values. Results are shown in Table 1, which for each cap shows the average compute rate (in MFlop/s) and its coefficient of variation (CV) in percentage. Moreover, a linear regression of the average compute rate versus the cap yields an R-squared value of 0.99969, showing that the compute rate increases almost perfectly linearly with the cap. The CV values are all low, below 3%, across the experiments.

Cap	1	10	20	30	40	50	60	70	80	90	100
MFlops/s	7.41	77.17	153.41	229.90	299.40	391.25	472.74	550.56	631.20	710.34	790.34
CV (%)	1.33	0.95	1.49	0.69	2.80	0.42	0.08	0.81	0.07	0.07	0.03

Table 1: Average compute rate and CV for a single VM instance for different cap values over 10 trials.

A linear increase of the compute rate with the cap value is necessary for Xen's CPU sharing to be valid. However, we still need to determine whether the actual compute rates achieved are themselves reasonable. Consequently, we run our matrix multiplication program outside of Xen to assess the "raw" performance of the program, which achieves a compute rate of 792.52 MFlop/sec (averaged over 10 trials). We first observed that this compute rate is 0.27% higher than the rate obtained within Xen for a VM instance with

a cap set at 100%. However, we can easily compute the difference between the compute rate achieve by a VM instance when assigned a cap of x%, and between x% of this raw compute rate. It turns out that the R-squared quantifying the difference between the observed average compute rates for different cap values and the compute rates computed based on the cap values and on the raw compute rate is 0.99931. Finally, the CV over the 10 runs outside of Xen is 0.08%, which is comparable to CV values observed within Xen.

We conclude that Xen leads to negligible overhead, and more importantly that cap values can be used to allocate a precise fraction of the CPU to a VM instance.

3.2 Multiple VM Instances Experiments

For these experiments we ran multiple VM instances that ran for one hour using predefined as well as randomly assigned cap values. The experiment conducted for two VM instances used predetermined cap values of 1, 25, 50, 75 and 99. We also conducted experiments with 4, 6, 8 and 10 VM instances using randomly selected cap values. In each instance, the cap values used were picked such that their sum would be 100%. For all but the experiments with two VM instances, 10 different sets of cap values were used. Each individual experiment was repeated five times and results were averaged.

In order to quantify the error, for each VM instance we computed the absolute percentage difference between the cap value divided by 100 and the MFlops/s rate observed divided by the Mflops/s rate obtained with a cap value equal to 100%. A value of zero indicates that the cap value throttles the compute rate accurately. The highest absolute difference that was observed over all our experiments was 5.99%. The average absolute difference was 0.71%. Like for the experiments in the previous section, we found that the CV of a VM's observed compute rates between repeats of the same experiments was within the same 3% range.

As in the previous section, we conclude that Xen leads to negligible overhead and that cap values can be used to allocate a precise CPU fractions.

4 Responsiveness of CPU Sharing with Xen

In this section, we evaluate how responsive Xen is to changes in the cap values for VM instances. More precisely, how long does it take for a VM's effective CPU allocation to change after a change in its cap value?

For these experiments, we use the same system that was used previously. We also reuse the same matrix multiplication code with a slight modification. In order to test how responsive Xen is to changes in cap value, the matrix multiplication was modified to not keep track of a MFlops/s rate, but rather to do a fixed amount of work (i.e., 100 matrix multiplications) repeatedly. We term this fixed amount of work a "probe". After the completion of each probe, the VM instance sends a ping to a daemon running on a dedicated VM instance that keeps track of the time it took to complete the probe. In order to make sure the VM instance running the daemon did not interfere with the experiments, it was running on a different physical processor. This experiment was conducted for one hour, with the cap value of the VM instance changing every 10 minutes. In the end we obtained a time-stamped series of times needed to perform a probe, which we can put in perspective with the times at which the cap values were changed.

For the single VM instance experiment, the cap value of the VM instance was changed between the values of 20% and 80%. For the multiple VM instances experiment, five VM instances were used. The cap values of the VM instances switched between the values of 20% and 5%, 20% and 5%, 20% and 10%, 20% and 20%, and 20% and 60%. The conclusion of these experiments is as follows. Let t be the time at which the cap value is changed. Let p_{t-1} be the last probe time reported before time t, p_t the first probe time reported after time t, and p_{t+1} the second probe time reported after time t. In all cases we found that p_{t+1} is in line with the new cap value. p_t is typically in between p_{t-1} and p_{t+1} because the cap value was changed while the probe was executing. Therefore, it takes at most the time to execute one probe before the new CPU share imposed by the new cap value becomes effective. Probe times depend on the cap values, but in many of our experiments these times were on the order of one second (they were larger when cap values were particularly low).

More fine-grained experiments could be conducted to determine the responsiveness of Xen to cap value changes more precisely. However, for our broader purpose, the responsiveness we observe is sufficient to justify the use of Xen, or technologies like it, to manage a virtual cluster that enables dynamic and flexible resource allocation.

5 Conclusion

In this paper we have evaluated the accuracy and responsiveness of the Xen CPU scheduler when using solely cap values. Our experiments spanned both single- and multi-VM instance scenarios, on a single physical processor. We found that the Xen scheduler is likely sufficiently accurate and responsive in a view to enabling flexible resource allocations in virtual clusters.

References

[1] Top500 supercomputer sites. http://www.top500.org, 2008.

[2] John Markoff and Saul Hansell. Hiding in Plain Sight, Google Seeks More Power. http://www.nytimes.com/2006/06/14/technology/14search.html, 2006.

[3] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2.

[4] Jonathan G. Koomey. Estimating Total Power Consumption by Servers in the U.S. and the World. http://enterprise.amd.com/Downloads/svrpwrusecompletefinal.pdf, February 2007.

[5] U.S. Environmental Protection Agency. Report to Congress on Server and Data Center Energy Efficiency. http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congres August 2007.

[6] 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. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 164–177, 2003.

[7] Microsoft Virtual PC. http://www.microsoft.com/windows/products/winfamily/virtualpc/default.mspx.

[8] VMware. http://www.vmware.com/.

[9] Intel Virtualization Technology (Intel VT). http://www.intel.com/technology/virtualization/index.htm.

[10] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide Third Edition*. Society for Industrial and Applied Mathematics, 1999.