The Linux Load Balance: Wasted vCPUs in Clouds

Matthew Elbing
Hofstra University
melbing1@pride.hofstra.edu

Jianchen Shan
Hofstra University
jianchen.shan@hofstra.edu

I. Introduction

With the rise of multicore machines, the Linux scheduler has introduced a sophisticated load balancing mechanism to spread the tasks over the cores. Study [1] has shown that the load balancer implemented in the default Linux scheduler, Complete Fair Scheduler (CFS), works well to evenly distribute the workload and succeeds in maintaining high CPU utilization.

However, our experiment shows that the Linux load balancer performs poorly in virtualized environments. Specifically, the CPU time available to a virtual machine (VM) cannot be fully utilized. The Linux load balancer assumes that the virtual CPUs (vCPUs) of a VM are symmetric, since the VM will equally assign CPU time to each vCPU with the intention of emulating the underlying multicore machine. But in a multi-tenant cloud where time sharing is employed for high resource utilization and low cost, the vCPU capacity is also determined by the VMs of other users on the server. Thus, this assumption no longer holds true and may mislead the load balancer into making incorrect scheduling decisions.

Our observation indicates that the vCPUs are dynamically asymmetric when the physical machine is time-shared by multiple VMs. A vCPU can reach its maximal capacity when all other vCPUs co-located on the same core are idle. In this case, all the CPU time of the core can be utilized by the vCPU due to the work-conserving principle, which may even allow a vCPU to consume more CPU time than it was assigned. On the other hand, a vCPU would have lower capacity on a core that is highly contended by multiple co-running vCPUs. A vCPU's capacity varies when the contention on the core changes. This makes the vCPU capacity dynamic.

Our major contribution is to experimentally reveal that Linux load balancer is unaware of the dynamic asymmetry of vCPUs, which results in vCPU time wasted and low resource utilization within a cloud VM. We propose dynamically collecting and exposing the vCPU capacity to the Linux scheduler. This can assist the load balancing algorithm in making more informed scheduling decisions when responding to a change in vCPU capacity. For example, heavy or critical jobs would be migrated to powerful vCPUs, and light or trivial tasks would be scheduled to weak vCPUs.

The multi-tenant cloud has been known to suffer from poor and unpredictable performance. To guarantee the Quality of Service (QoS), major cloud vendors, like Amazon AWS, are forced to deploy the most of their VMs on dedicated machines and sacrifice the benefits of time sharing. To overcome this challenge, major issues such as virtualization overhead [2], vCPU discontinuity [3], and poor performance isolation [4], have been heavily studied. However, little attention has been paid to increasing the resource utilization within the VM. Our efforts towards this end complement, and are orthoganal to, previous work. Furthermore, many algorithms such as Bias Scheduling [5], have been successfully designed to schedule tasks on asymmetric multicore machines. Although not directly applicable, the basic ideas could be applied to the Linux load balancer in VMs.

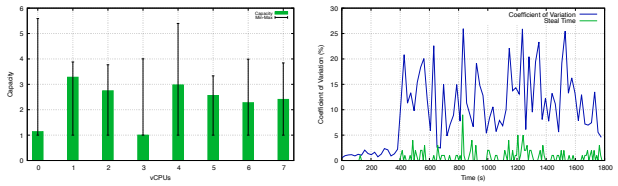
II. Dynamic and Asymmetric vCPUs in the Cloud

To demonstrate the Dynamic and Asymmetric vCPUs (DA-vCPUs) in the cloud, we first define vCPU capacity and explain how it is measured. Since vCPUs are time-sharing the cores, the vCPU capacity can be defined as the percentage of the CPU time a vCPU can consume during a certain period. Increased vCPU capacity comes with an increase in consumable CPU time.

The vCPU capacity is mainly determined by three factors. First, the CPU time allocated to a vCPU is translated to the ability of the vCPU to compete for CPU time. Second, the vCPU capacity depends on the co-located vCPUs of other VMs. If all co-located vCPUs are actively competing for CPU time and the core cannot satisfy the total CPU time demands, a vCPU may consume less CPU time than it was allocated. In addition, if all co-located vCPUs are idle, a vCPU may consume all the CPU time of a core. Third, the vCPU capacity is affected by other vCPUs in the same VM. The vCPUs in a VM participate in time sharing as a group to receive the CPU time allocated to the VM. For instance, if only one vCPU is active and all other sibling vCPUs are idle, the active vCPU would be prioritized and receive a larger share of the CPU time in order to increase the VM's resource utilization.

To measure the vCPU capacity, we developed a multi-threaded tool [7] which launches one thread on each vCPU running a CPU-bound task (i.e., incrementing a counter). Each thread will periodically collect the *steal time*, which is only available within VM to indicate the percentage of the time that the vCPU has to wait while other co-located vCPUs are running on the same core. For example, if the tool reports that the steal time of a vCPU over a certain period is 30%, then the vCPU capacity is 70% of the total CPU time in this period. The tool is implemented inside the VM to allow measurement of relative vCPU capacity of a VM in the cloud where the user has no access outside the VM.

978-1-7281-8266-7/20/$31.00 ©2020 IEEE
DOI 10.1109/IEEECloudSummit48914.2020.00035



(a) Normalized vCPU capacity (b) vCPU capacity distribution

Fig. 1: vCPU capacity is dynamic and asymmetric in the cloud

To gain insight into of the characteristics of vCPUs in a real-world cloud, we deployed our tool in a *t2.2xlarge* instance, which has 8 vCPUs time sharing a physical machine with other VMs on Amazon Web Services (AWS). We ran the tool for 30 minutes. Figure 1a shows the vCPU capacity for each vCPU at a specific point in time and their min-max values during the test. Figure 1b shows how the Coefficient of Variation (CV) of the vCPU capacity distribution and the total steal time of the VM change over the course of the experiment. The results clearly demonstrate that the vCPU capacity can be significantly asymmetric. Moreover, such asymmetry changes notably over time. AWS t2 instances normally serve burstable workloads, which dynamically change the loads on cores which in turn influences the vCPU capacity. This can be shown via the impact of changing steal time on the CV.

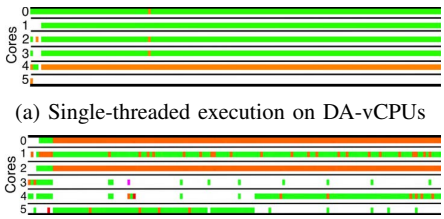
III. Unawareness of DA-vCPUs

In this section, we will show that the Linux load balancer is unaware of the vCPU asymmetry through experiments in which two 6-vCPU VMs (i.e. the main VM and the co-runner VM) sharing a 6-core machine are built to mimic a multi-tenant cloud. Both VMs are created by KVM running Linux kernel v5.4.47. The vCPUs of each VM are pinned one-on-one to cores. To create the vCPU asymmetry in the main VM, we let the co-runner VM keep some vCPUs busy running CPU-bound tasks and some vCPUs idle. Therefore, within the main VM, the vCPUs co-located with idle co-runners would have a higher capacity than the those co-located with busy co-runners.

Linux load balancer mainly functions by thread placement and thread re-balancing. The former attempts to place a new/wakened thread on the core with the least load. The latter detects imbalance (e.g. a core becomes idle) and conducts thread migrations if a re-balance is possible. To see if they work well on the DA-vCPUs, we wrote a synthetic program to generate single-threaded and multi-threaded workloads on the main VM. We observed that both mechanisms failed, leading to wasted vCPU time due to the unawareness.

A. Single-threaded Execution on DA-vCPUs

This experiment is conducted to show that the Linux load balancer cannot place a thread based on the vCPU capacity. In the co-runner VM, only vCPU5 is set to idle on core5, and other vCPUs are set to busy. Thus, in the main VM, vCPU5 has sole usage of a core and can therefore reach its maximum capacity. Then, we let the synthetic program launch one thread running CPU-bound tasks in the main VM for 30 seconds, and tracked its execution by profiling the loads on each of the cores



(a) Single-threaded execution on DA-vCPUs

(b) Multi-threaded execution on DA-vCPUs

Fig. 2: Linux load balancing is unaware of vCPU asymmetry

using the scheduler visualization tool [6]. Figure 2a depicts the results. The red color indicates the load of two busy vCPUs and the green color represents the load of one busy vCPU. The thread is expected to be scheduled to vCPU5 if the Linux load balancer is aware that it has the highest capacity. Instead, it is scheduled to vCPU4 and is thus competing for the core with the co-runner's vCPU.

B. Multi-threaded Execution on DA-vCPUs

This experiment is designed to demonstrate that the Linux load balancer fails to do re-balancing by matching the thread load with the vCPU capacity. In the co-runner VM, vCPU0-2 are set to busy and vCPU3-5 are set to idle. Thus, in the main VM, vCPU3-5 should have higher capacities. In the main VM, the synthetic program launches 3 heavy threads running CPU-bound tasks and 3 light threads alternating between running CPU-bound tasks and sleeping; these threads are run for 30 seconds. Ideally, if the Linux load balancer was aware of the vCPU capacity distribution, the 3 heavy threads should be scheduled to vCPU3-5 and the 3 light threads ought to be scheduled to vCPU0-2 to achieve the "balance" in which the load/capacity ratio on each vCPU is similar. However, as shown in figure 2b, 2 heavy threads are scheduled to core0 and core2, and 1 heavy thread is scheduled to core5 and then migrated to core6. The 3 light threads are migrated among vCPU1 and vCPU3-5, which leads to more vCPU time wasted.

IV. Discussion and Future Work

In the future, more investigation could be done to discover other issues (e.g. fairness problem) caused by the unawareness. Evaluations using realistic workloads will be performed to measure related performance degradation. We propose to periodically collect and expose the vCPU capacity in order to assist the Linux load balancer in making more informed decisions and optimize the resource utilization in the cloud.

References

[1] Bouron, Justinien, et al. "The Battle of the Schedulers: FreeBSD ULE vs. Linux CFS." USENIX ATC 2018.
[2] Ding, Xiaoning, et al. "Gleaner: Mitigating the blocked-waiter wakeup problem for virtualized multicore applications." USENIX ATC 2014.
[3] Cheng Luwei, et al. "vScale: automatic and efficient processor scaling for SMP virtual machines." EuroSys 2016.
[4] Zhao Yong, et al. "Characterizing and optimizing the performance of multithreaded programs under interference." PACT 2016.
[5] Koufaty David, et al. "Bias scheduling in heterogeneous multi-core architectures." EuroSys 2010.
[6] Lozi Jean-Pierre, et al. "The Linux scheduler: a decade of wasted cores." EuroSys 2016.
[7] vCPU capacity measurement, https://github.com/melbing1/da-vcpus