Mitigating nesting-agnostic hypervisor policies in derivative clouds

Chandra Prakash, Prashanth, Purushottam Kulkarni, Umesh Bellur
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai, India
{*chandrap, prashanth, puru, umesh*}*@cse.iitb.ac.in*

***Abstract*—The fixed granularity of virtual machines offered by IaaS providers has prompted the evolution of derivative clouds where resources are repackaged into smaller containers and leased out typically in PaaS mode. In such a setup, containers are provisioned within virtual machines. Such a nested setup results in two control centers for the resources used by those containers—the guest OS and the Hypervisor. The latter's control actions are agnostic of the application executing within a VM. This lack of visibility may result in hypervisor control that has a non-uniform effect on the VM's nested containers which is undesirable. In this work, we propose policy based control of the effect of the hypervisor's control actions amongst the containers nested in the affected VM.**

1. Introduction

A Derivative cloud is an intermediate layer between a cloud provider (such as Amazon EC2 or an Enterprise cloud) and cloud service consumer. Derivative clouds deliver services (infrastructure, platform or software) hosted on virtual machines purchased from primary IaaS providers. This model is motivated by the mismatch between the granularity of VMs leased by the IaaS providers and that of the services consumed off of that infrastructure [1]. For example, a PaaS provider can offer fine grained PaaS instances that are co-hosted on larger VMs leased from infrastructure cloud providers. Further, motivation stems from the pricing schemes and lack of standardization of IaaS providers that encourage the purchase of larger VMs that then drive the derivative cloud providers to share this infrastructure for multiple IaaS/PaaS/SaaS instances [2].

PaaS providers resolve this granularity mismatch by using nested virtualization. However, owing to the drawbacks of nested system virtualization (performance overheads) [2], they employ light weight containers corresponding to PaaS instances inside the leased VM. In addition to lower overheads, containers provide other benefits such as faster provisioning, faster spin-up and cost effectiveness [3].

Infrastructure providers frequently over-commit resources, primarily along the CPU and Memory axes [4]. Today's hypervisors provide adequate controls to manage this over-commitment using techniques such as memory ballooning and dynamic vCPU (de)allocation [5]. However, hypervisor actions are limited to managing physical resources whereas the virtual resources allocated to a VM are managed by the guest OS. This separation of responsibilities results in two control centers over essentially the same set of resources. In such a setup, hypervisor actions are agnostic of the way the VM is managing its resources internally. The fact that virtual resources are further divided amongst a set of nested containers is of no consequence to the hypervisor. It is therefore entirely possible that a hypervisor action adversely affect the containers nested inside the VM.

Early indicators of our empirical studies point out that the impact of hypervisor actions related to managing over-commitment are non-uniformly distributed over the multiple containers that are nested inside the VM. In this effort, we propose user specified policy driven control of the hypervisor's action on nested containers.

2. Nesting-agnostic resource management

We consider the effect of hypervisor actions to deal with over-commitment along two resource axes—Memory and CPU. We present the two separately, in terms of illustrating the effects of default behavior and suggest how this can be controlled.

2.1. Memory management issues

As part of this work, we focus on the interplay of hypervisor based memory ballooning actions and its impact on nested containers. The hypervisor employs ballooning to support dynamic memory provisioning to virtual machines. The action of shrinking a balloon inside a VM causes memory pages to be reclaimed by the hypervisor for use elsewhere. We now present the key results of an empirical study to show the effects of this on individual containers nested in the VM.

Two Linux+KVM virtual machines having 16GB memory along with 6 vCPUs and 6GB memory along with 1 vCPU were used. One VM was used in a nested setup (16GB), which hosted containers and the other VM (6GB) generated workloads for the container-hosted applications. First VM nests four Docker containers executing Redis and MongoDB workloads. Default configurations of the applications executed within the containers is as shown in Table 1. The YCSB workbench was used to generate workload datasets.

Container	hard-limit (GB)	soft-limit (GB)	Key Size (# records)	Usage (GB)
Redis-Low	2	0.5	500K	1.3
Redis-High	4	1	1000K	2.6
Mongo-Low	2	0.5	500K	1.3
Mongo-High	4	1	1000K	2.6

TABLE 1: Configuration of the four application containers.

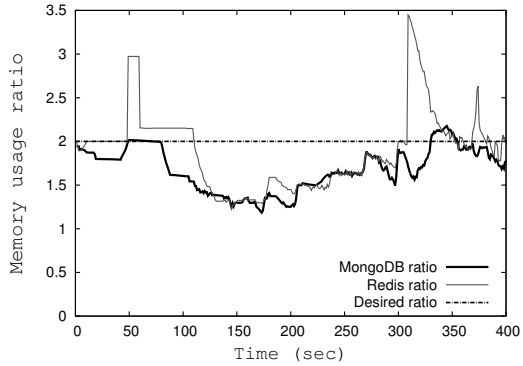

Figure 1: Memory usage ratios.

For the first 100 seconds, the applications were executed without memory pressure to consume as much memory as required. Beyond 100 seconds, memory pressure was generated from the host by reclaiming VM's allocated memory at rate of 2 GB every 30 seconds.

The two containers were configured to consume memory in the ratio of 1:2 (refer to Table 1), via hard-limit and soft-limit *memory cgroup* specifications. The results show no adherence to the ratios as shown in Figure 1. This is caused by the balloon driver doing its work unaware of the ratio and the actual usage of the memory by different containers. A nesting aware solution would have ensured reclamation in the presence of the ratios at all times.

2.2. Implications of CPU over-commitment

CPU over-commitment is a common technique for improving CPU efficiency in virtualized environments. vCPUs allocated to a VM can be distributed among multiple containers by using two Linux cgroups specification parameters: *cpuset.cpus* and *cpu.share*. Containers can be pinned to a particular set of vCPUs by using *cpuset.cpus*.

Pinning containers to vCPUs/CPUs has the potential to provide several benefits—resource isolation, improved throughput and improved power utilization [3]. Further, with over-commitment, dynamic scaling of vCPUs is employed to reduce scheduling overheads and address synchronization related preemptions issues [5]. We envision CPU pinning to be a desired feature and an available specification as part of the application provisioning service along with maintaining their respective CPU shares. We now present the key results of an empirical study to show the effects of vCPU scaling on individual containers nested in the VM.

One Linux + KVM virtual machine having 7 vCPUs and 8 GB memory is used to nest three Docker containers. Docker root node was pinned to vCPU1 to vCPU6. First container was pinned to vCPU1, the second pinned to vCPU2 and the third pinned to vCPUs 3, 4, 5 and 6. Sysbench benchmark is

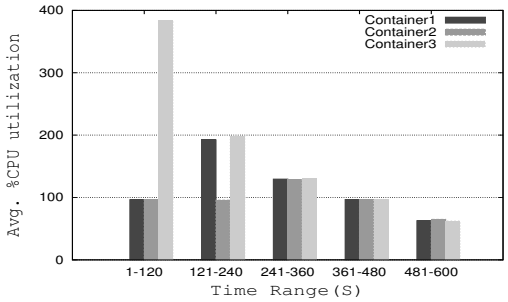

Figure 2: CPU utilization with CPU pinning.

used inside containers to generate CPU intensive workload (generation of prime numbers using four threads for 600 seconds). Number of vCPUs in the virtual machine is reduced by one vCPU every 120 seconds and scaling is stopped when the VM had three vCPUs remaining. The order of reclamation was vCPU1, vCPU2, vCPU3 and vCPU4. With CPU pinning to vCPUs the allocation ratios (1:1:4) do not hold as shown in Figure 2. This is caused by the scaling down of vCPUs in a numerically contiguous way. A nesting aware solution would have ensured CPU allocation ratio even in case of vCPU scaling.

3. Proposed solution approach

To control the effects of hypervisor actions on memory, we are proposing few policies such as proportionate memory allocation among containers or categorize the application containers into various categories such as gold, silver and bronze and perform memory allocation accordingly during memory pressure. In case of CPU, the proposed policies are—either maximize the vCPU pinning by pinning all containers or provide pinned vCPUs to few containers along with maintaining their respective CPU shares even in case of vCPUs scaling.

4. Conclusion

We demonstrated the adverse impact of hypervisor-based management actions in derivative cloud setups for the memory and CPUs. To overcome these nesting-agnostic actions, we have proposed various policies for containers running inside virtual machine.

References

[1] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy, "Spotcheck: Designing a derivative iaas cloud on the spot market," in *Proceedings of the 10th European Conference on Computer Systems*. ACM, 2015.

[2] D. Williams, H. Jamjoom, and H. Weatherspoon, "The xen-blanket: virtualize once, run everywhere," in *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 2012.

[3] A. Podzimek, L. Bulej, L. Y. Chen, W. Binder, and P. Tuma, "Analyzing the impact of cpu pinning and partial cpu loads on performance and energy efficiency," in *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2015.

[4] I. Banerjee, F. Guo, K. Tati, and R. Venkatasubramanian, "Memory overcommitment in the esx server," *VMware technical journal (VMTJ)*, vol. 2, pp. 2–12, 2013.

[5] L. Cheng, J. Rao, and F. Lau, "vscale: automatic and efficient processor scaling for smp virtual machines," in *Proceedings of the 11th European Conference on Computer Systems*. ACM, 2016.