An Updated Performance Comparison of Virtual Machines and Linux Containers

Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio
IBM Research, Austin, TX
{wmf, apferrei, rajamony, rubioj}@us.ibm.com

***Abstract*—Cloud computing makes extensive use of virtual machines because they permit workloads to be isolated from one another and for the resource usage to be somewhat controlled. In this paper, we explore the performance of traditional virtual machine (VM) deployments, and contrast them with the use of Linux containers. We use KVM as a representative hypervisor and Docker as a container manager. Our results show that containers result in equal or better performance than VMs in almost all cases. Both VMs and containers require tuning to support I/O-intensive applications. We also discuss the implications of our performance results for future cloud architectures.**

I. Introduction

This paper looks at two different ways of achieving resource control today, viz., containers and virtual machines and compares the performance of a set of workloads in both environments to that of natively executing the workload on hardware. In addition to a set of benchmarks that stress different aspects such as compute, memory bandwidth, memory latency, network bandwidth, and I/O bandwidth, we also explore the performance of two real applications, viz., Redis and MySQL on the different environments.

Our goal is to isolate and understand the overhead introduced by virtual machines (specifically KVM) and containers (specifically Docker) relative to non-virtualized Linux. We expect other hypervisors such as Xen, VMware ESX, and Microsoft Hyper-V to provide similar performance to KVM given that they use the same hardware acceleration features. Likewise, other container tools should have equal performance to Docker when they use the same mechanisms. The fact that Linux can host both VMs and containers creates the opportunity for an apples-to-apples comparison between the two technologies with fewer confounding variables than many previous comparisons.

We make the following contributions:

- We provide an up-to-date comparison of native, container, and virtual machine environments using recent hardware and software across a cross-section of interesting benchmarks and workloads that are relevant to the cloud.
- We identify the primary performance impact of current virtualization options for server workloads.
- We elaborate on a number of non-obvious practical issues that affect virtualization performance.
- We show that containers are viable even at the scale of an entire server with minimal performance impact.

II. Evaluation

All of our tests were performed on an IBM System x3650 M4 server with two 2.4-3.0 GHz Intel Sandy Bridge-EP Xeon E5-2665 processors for a total of 16 cores (plus HyperThreading) and 256 GB of RAM. The two processors/sockets are connected by QPI links making this a non-uniform memory access (NUMA) system. This is a mainstream server configuration that is very similar to those used by popular cloud providers. We used Ubuntu 13.10 (Saucy) 64-bit with Linux kernel 3.11.0, Docker 1.0, QEMU 1.5.0, and libvirt 1.1.1. For consistency, all Docker containers used an Ubuntu 13.10 base image and all VMs used the Ubuntu 13.10 cloud image.

Power management was disabled for the tests by using the performance cpufreq governor. Docker containers were not restricted by cgroups so they could consume the full resources of the system under test. Likewise, VMs were configured with 32 vCPUs and adequate RAM to hold the benchmark's working set. We use microbenchmarks to individually measure CPU, memory, network, and storage overhead. We also measure two real server applications: Redis and MySQL. In this abstract we present only the MySQL results in detail. Consult our tech report for the full results.

A. MySQL

We chose to evaluate MySQL because it is a popular relational database, it is widely used in the cloud, and it stresses memory, IPC, network and filesystem subsystems. We ran the SysBench oltp benchmark against a single instance of MySQL 5.5.37. Five different configurations were measured: MySQL running normally on Linux (native), MySQL under Docker using host networking and a volume (Docker net=host volume), using a volume but normal Docker networking (Docker NAT volume), storing the database within the container filesystem (Docker NAT AUFS) and MySQL running under KVM. Although MySQL accesses the fileystem, our configuration has enough memory that it performed almost no actual disk I/O, making different KVM storage options moot.

Figure 1 shows transaction throughput as a function of the number of users simulated by SysBench. The right Y axis shows the loss in throughput when compared to native. The general shape of this curve is what we would expect: throughput increases with load until the machine is saturated, then levels off with a little loss due to contention when overloaded. Docker has similar performance to native, with the difference asymptotically approaching 2% at higher concurrency. KVM has much higher overhead, higher than 40% in all measured cases. AUFS introduces significant overhead which

978-1-4799-1957-4/15/$31.00 ©2015 IEEE

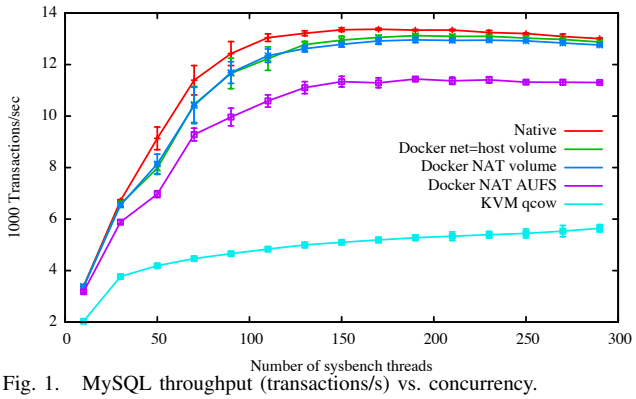

Fig. 1. MySQL throughput (transactions/s) vs. concurrency.

is not surprising since I/O is going through several layers, as seen by comparing the *Docker NAT volume* and *Docker NAT AUFS* results. The AUFS overhead demonstrates the difference between virtualizing above the filesystem, as Docker does, and virtualizing below the filesystem in the block layer, as KVM does. We tested different KVM storage protocols and found that they make no difference in performance for an in-cache workload like this test. NAT also introduces a little overhead but this workload is not network-intensive.

B. Discussion

We see several general trends in our results. As we expect given their implmentations, containers and VMs impose almost no overhead on CPU and memory usage; they only impact I/O and OS interaction. This overhead comes in the form of extra cycles for each I/O operation, so small I/Os suffer much more than large ones. This overhead increases I/O latency and reduces the CPU cycles available for useful work, limiting throughput. Unfortunately, real applications often cannot batch work into large I/Os.

Docker adds several features such as layered images and NAT that make it easier to use than LXC-style raw containers, but these features come at a performance cost. Thus Docker using default settings may be no faster than KVM. Applications that are filesystem or disk intensive should bypass AUFS by using volumes. NAT overhead can be easily eliminated by using –net=host, but this gives up the benefits of network namespaces.

While KVM can provide very good performance, its configurability is a weakness. Good CPU performance requires careful configuration of large pages, CPU model, vCPU pinning, and cache topology; these features are poorly documented and required trial and error to configure. Even on the latest version of Ubuntu we were unable to get vhost-scsi to work, so there is still room for improvement in virtual I/O. KVM's complexity represents a barrier to entry for any aspiring cloud operator, whether public or private.

III. Conclusions and Future Work

Both VMs and containers are mature technologies that have benefited from a decade of incremental hardware and software optimizations. When both are tuned for performance, Docker equals or exceeds KVM performance in every case we tested. Our results show that both KVM and Docker introduce negligible overhead for CPU and memory performance (except in extreme cases). For I/O-intensive workloads, both forms of virtualization should be used carefully.

We find that KVM performance has improved considerably since its creation. Workloads that used to be considered very challenging, like line-rate 10 Gbps networking, are now possible using only a single core using 2013-era hardware and software. Even using the fastest available forms of paravirtualization, KVM still adds some overhead to every I/O operation; this overhead ranges from significant when performing small I/Os to negligible when it is amortized over large I/Os. Thus, KVM is less suitable for workloads that are latency-sensitive or have high I/O rates. These overheads significantly impact the server applications we tested.

Although containers themselves have almost no overhead, Docker is not without performance gotchas. Docker volumes have noticeably better performance than files stored in AUFS. Docker's NAT also introduces overhead for workloads with high packet rates. These features represent a tradeoff between ease of management and performance and should be considered on a case-by-case basis.

In some sense the comparison can only get worse for containers because they started with near-zero overhead and VMs have gotten faster over time. If containers are to be widely adopted they must provide advantages other than steady-state performance. We believe the combination of convenience, faster deployment, elasticity, and performance is likely to become compelling in the near future.

Our results can give some guidance about how cloud infrastructure should be built. Conventional wisdom (to the extent such a thing exists in the young cloud ecosystem) says that IaaS is implemented using VMs and PaaS is implemented using containers. We see no technical reason why this must be the case, especially in cases where container-based IaaS can offer better performance or easier deployment. Containers can also eliminate the distinction between IaaS and "bare metal" non-virtualized servers since they offer the control and isolation of VMs with the performance of bare metal. Rather than maintaining different images for virtualized and non-virtualized servers, the same Docker image could be efficiently deployed on anything from a fraction of a core to an entire machine.

We also question the practice of deploying containers inside VMs, since this imposes the performance overheads of VMs while giving no benefit compared to deploying containers directly on non-virtualized Linux. If one must use a VM, running it inside a container can create an extra layer of security since an attacker who can exploit QEMU would still be inside the container.

Source Code

The scripts to run the experiments from this paper are available at https://github.com/thewmf/kvm-docker-comparison.

Acknowledgements

This work was supported in part by the Office of Science, United States Department of Energy under award number DE-SC0007103.