Deliverable #3

Technical Report – Performance Evaluation

By Pijus Mackevicius, Alec Scully, Asante Morris

May 9, 2019

**Motivation**

This technical report is the final deliverable for a project aimed to install and configure virtualization containers and automate selected benchmarks inside these containers for performance evaluation using CloudLab infrastructure. The project’s motivation is a study paper published by IBM in 2015 that compares the performances of Native, Docker, and Kernel-based Virtual Machine (KVM) technologies (“An Updated Performance Comparison of Virtual Machines and Linux Containers”). This report serves to assess the project’s progress and final results.

**Introduction**

Virtualization and containerization technologies are significant in executing the goals of cloud computing: scalability, portability, agility and maintenance, among others. Thus, it is important to assess the performance and the cost attached to run the execution of these technologies. Cloud service providers are interested in delivering the fastest option possible with the lowest cost.

In the past, virtual machines lacked in performance when compared to container-based options. The hypervisor layer used in managing virtual machines would create a significant overhead that reduce the operation speeds, especially in networking where maximum output speed compared to a native machine is cut in half. Container-based virtualization technology does not introduce such an overhead and shares the kernel with the operating systems machine without having to create guest operating systems. However, the performance gap of virtual machines and containers can be closed, as virtual machine technology is updated to reduce resource waste and increase efficiency, whereas container-based technology performance is presumed to be capped based on near non-existent overhead. This project aims to explore whether the virtual machines have closed some of the gap between themselves and container-based virtualization, by comparing collected results to the data gathered in the motivation paper from 2015.

**Project Tools**

The team has created a GitHub repository(https://github.com/AS805456/cluster-template) for the project. It is a fork of Dr. Linh Ngo’s cluster-template repository (https://github.com/linhbngo/cluster-template) and the materials are used for CloudLab compatibility and Docker installation. We are also using the code and experiments of the motivation study paper’s team found in their own GitHub repository (https://github.com/thewmf/kvm-docker-comparison) for guidelines for benchmark installation and execution. Our current KVM manual installation code is taken from Cloud Computing course materials. The deployment infrastructure used in this project is CloudLab (www.cloudlab.us) - a cloud service for scientific and academic research. CloudLab allows the option to upload a GitHub repository which it uses to allocate resources and execute operations defined inside.

**Selections**

The team has selected two benchmarks for performance evaluation:

● Linpack benchmark – a software library used to perform numerical linear algebra to measure system’s floating-point computing power by calculating dense systems of linear equations (in Floating Point Operations Per Second, or FLOPS)

● Stream benchmark – a program measuring memory bandwidth by performing operations on vectors. The performance is determined by bandwidth to main memory and cache miss’ costs. The benchmark has four measurable components - copy, scale, add and triad, that perform different operations in the kernel (measured in GB/s).

The team has also selected three virtualization technologies to test these benchmarks in:

● Docker (https://www.docker.com/) – a container virtualization software that runs isolated containers with their own contents that can communicate with each other using dedicated channels. All Docker containers use a single operating system kernel, thus working with less overhead than traditional virtual machine technologies.

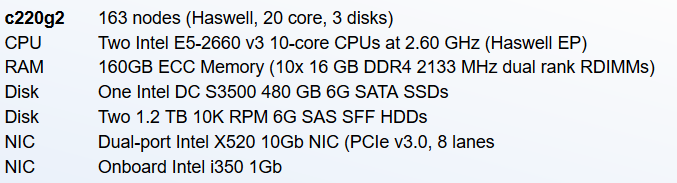
● Kernel Virtual Machine (KVM) – a device node in the Linux kernel that allows the kernel to function as a hypervisor – a component that allows to create, use and manage virtual machines (VMs) and manages execution of guest operating systems by providing an interface/platform for interaction. The operations provided by the KVM device node include creating and allocating memory to a VM, reading and writing virtual CPU registers, handling interrupts inside a virtual CPU, and running a virtual CPU.

● Singularity – A containerization platform that aims to introduce containers to the world of high-performance computing. Singularity utilizes image files that physically contain a container and acts as a representation of the container environment.

To have consistency in our results, it was important to have an equal test environment for all tested virtualization technologies. Since this project heavily involves CloudLab infrastructure provisioning, the team utilized the resources available from the CloudLab services .In our experiments, we used the same hardware infrastructure for all three virtualization platforms - CloudLab Wisconsin cluster node c220g2. This information is specified in the profile.py file found in each of the GitHub branches that CloudLab uses to instantiate deployment:



Here is the list of hardware specifications for the node c220g2, taken from the Hardware section of the CloudLab Manual (http://docs.cloudlab.us/hardware.html):



This setup allows us to execute required tasks for our experiments and have accurate results, with enough computing power to perform all benchmarks.

**Goals**

This project aims to achieve the following goals:

● Automation for Docker, KVM and Singularity installation

● Execution of Linpack and Stream benchmarks in Docker, KVM and Singularity

● Data collection for Linpack and Stream benchmarks in Docker, KVM, and Singularity

● Comparison of benchmark findings between Docker, KVM and Singularity

● Relating the collected data to the findings of the motivation paper study

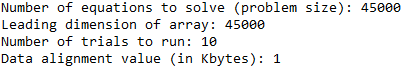
● Instructions for replication of our experiments with fully functioning profiles

**Project Progress**

The team began by creating the GitHub repository denoted in the Project Tools section. All materials, scripts and builds currently used in the project can be found inside this repository. This GitHub repository contains KVM, Docker, and Singularity branches, which were modified to allow compatibility with CloudLab provisioning and deployment technology. Each branch has a profile.py file, which contains how CloudLab should initiate a virtual node deployment. CloudLab can also provide services that are specified by node.AddService() command, which will be used extensively during the automation process. Important to note that CloudLab adds the services after the node is deployed, not before the deployment is finished; thus, there might be a delay when trying to access these services through a shell, as CloudLab might still be executing the addition of services.

At first, our team was set on to use the RandomAccess benchmark, which tests speed of memory operations. However, due to unattainability of RandomAccess benchmark that we hoped we could operate with, we have decided to replace it with the Stream benchmark.

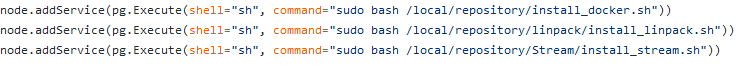
Initially, a different execution of Linpack was used, that operated with smaller array sizes, and did not use the full capability of the systems. Therefore, this execution version was discarded and the team utilized the newest version of Intel Linpack - 10.3.4. An older version of this benchmark was also used in the motivational paper analysis, therefore it was a good choice for comparison. This benchmark uses user input for evaluations; to be consistent with the motivational study paper’s data, and to allow maximum performance evaluation, we have chosen to work with a large input:



Stream benchmark was the most recent addition to our project, replacing RandomAccess due to easier configurability and compatibility. The team used the code of the motivational paper study’s GitHub repository, slightly tweaked to allow compatibility with used infrastructure.

1. **Docker**

Our team has automated Docker installation using a shell script that is executed when CloudLab instantiates the Docker branch. Docker is available to use with minimal delay after node deployment is complete. CloudLab has the option of adding services into provisioned nodes using the *node.addService()* command. This command is used to install Docker, Stream and Linpack benchmarks automatically - scripts *install\_docker.sh, install\_linpack.sh* and *install\_stream.sh* respectively:



Both benchmarks contained Dockerfiles that were used to build the images linpack and stream. Additionally, stream makes use of a *Makefile* that compiles the C files used for execution of the benchmark. Once CloudLab finishes deployment, both benchmarks can be executed using Docker by launching the commands

*sudo docker run --it --rm --privileged linpack*

*sudo docker run --rm --privileged stream*

**B. Singularity**

Singularity’s installation has been automated by launching install\_singularity.sh script after CloudLab deploys the node from the Singularity branch, using the *node.addService()* command. The benchmark installation has not been automated due to the approach the team selected.



The verification of the singularity installation can be done with the following command:

*singularity*

Using *singularity shell contain*, we enter the shell of our container and are able to run the benchmarks from within. The Linpack benchmark compressed file is downloaded, decompressed, and the executable file is ran inside the *contain*:

*wget* *http://registrationcenter-download.intel.com/akdlm/irc\_nas/2169/l\_lpk\_p\_10.3.4.007.tgz*

*tar zxvf l\_lpk\_p\_10.3.4.007.tgz*

*cd linpack\_10.3.4/benchmarks/linpack*

*./xlinpack\_xeon64*

To execute the Stream benchmark, within the *singularity contain*, a *git clone* of the Singularity branch from the project’s GitHub repository is used, and by compiling the code with a *Makefile* inside the Stream folder, Stream can be launched via the *stream.exe* executable:

*cd cluster-template/Stream*

*make*

*cd bin*

*./stream.exe*

**C. KVM**

The team ran into significant problems when trying to install and operate the KVM. Initially, the code that was used from the course materials had outdated references that had to be tweaked, namely the installation image that was used. This change may have had impact on the compatibility of the rest of the parameters that were selected for installation. All tools used were installed as intended. However, when running the command *virt-install*, that installs the virtual machine, it did not recognize the console addition in the parameter *--extra-args* *‘console=ttys0,115200n8 serial’*. Therefore, after VM installation, the team had no means to access the VM through a console, as no text output from the VM was installed.

The automation of KVM proved an easier challenge. We used a generic Kickstarter file that specifies the parameters during installation. Upon specifying this file as one of the parameters in *virt-install*, the installation can execute without any manual input, as any prompt is filled with the information from the file. The Kickstarter file can be found in the KVM branch of the project’s repository, and is used by first cloning the KVM branch and then creating a path to the file:

*sudo git clone -b kvm --single-branch https://github.com/AS805456/cluster-template*

*sudo virt-install ….. \ --initrd-inject cluster-template/Kickstarter.cfg \*

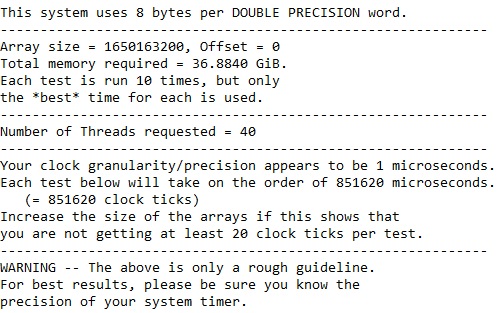
Contrary to the initial belief, specifying the Kickstarter file in the *extra-args* was not the reason for the console unavailability, as even removing any pointers to a Kickstarter proved the same results. Furthermore, the team was unable to establish a connection between the guest OS and host OS using the secure shell protocol (SSH). Connection was refused each time an ssh command was issued that targeted the address of the virtual machine. Upon inspection, we suspect that it is because of the network configuration of the VM that had to be changed from the inside of the VM - but this was impossible due to a non-responsive console. Using *virsh edit*, the team attempted to edit the contents of the VM in hopes of changing the parameters that would allow for an SSH connection, but to no avail. With all known options exhausted, the VM could not be reached, and subsequently, no benchmarks were executed. Therefore, the KVM data for the results section was left blank and will not be used for comparison.

The KVM installation file can be found in the KVM branch of the team’s GitHub repository - *install\_kvm.sh*. If added as a service using *node.addService()*, it takes approximately 20 to 25 minutes for a complete installation, and the progress cannot be seen. Since the state of the automation never reached fruition, the team did not employ *node.addService()*, therefore the installation completion was not tracked by using a suitable notification system indicating that the service addition process was complete. To launch the installation process, upon instantiating the KVM branch, the command

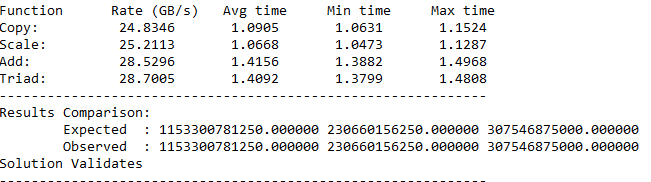
*sudo bash /local/repository/install\_kvm.sh*

is executed in the node’s shell.

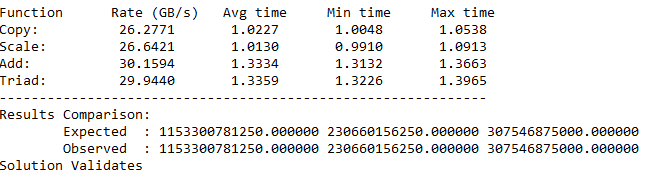
**Results**

The output of the Stream benchmark is split into 4 parts: copy, scale, add, and triad. These operations use different execution and therefore have different speeds. Before each test on each virtualization platform, the following message is sent, containing hardware specifications:

**Docker’s Stream results**:



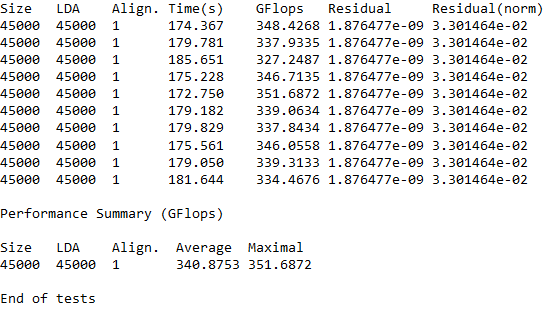
**Singularity’s Stream results**:



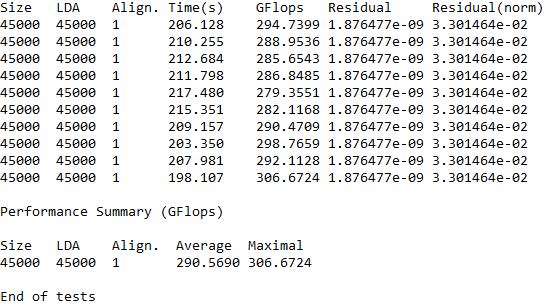
**KVM Stream results**: No Data

The Linpack benchmark input is specified in the **Project Progress** section. This benchmark operates best with large sizes of data, which accurately shows number of operations per second, as working with lower data may introduce significant overhead not related to calculations and will not employ the full capabilities of the machine.

**Docker’s Linpack results**:



**Singularity’s Linpack results:**



**KVM’s Linpack results:** No Data

**Result Discussion**

Docker was about 17% faster than Singularity during the Linpack benchmark. However, Singularity was 5.5% faster than Docker during the executing of Stream - 8% faster Copy, 4% faster Scale, 3% faster Add, and 7% faster Triad. We can make an assumption that Docker is better at utilizing the CPU cycle performance, while Singularity is slightly better at memory bandwidth. Depending on the implementation, the way both containers employ overhead and communication between the container and the kernel, this variation can be explained. Stream benchmark is significantly shorter than Linpack in execution time, and the difference between Docker and Singularity execution is minimal. However, Linpack benchmark clearly showed that Docker is much faster in executing the CPU cycles than Singularity.

When relating these obtained results to those of the motivation paper’s data, few things have to be kept in mind. First, Singularity was not tested, and no comparison can be made in this regard. Second, the motivation paper was published 4 years ago, and the technology regarding the interaction of the layers in container virtualization and virtual machines has most likely improved to be more efficient. Third, the hardware used in this project and in the motivation paper study differs significantly, which affects the comparison by a fair bit. Last but not least, the version of Linpack that this project used is more recent than the one used in the paper. With these points in mind, Docker outperformed its’ paper counterpart, while Singularity tied with it (340 against 290 GFLOPS, respectively). However, Stream results were significantly slower (27.5 average for Singularity and 26.25 for Docker against paper study’s 43.4 for Docker). Therefore, we suspect that our Stream results were optimized differently; nonetheless, they provide a perspective about the performance comparison of Docker and Singularity.

Unfortunately, the most interesting and compelling comparison between virtual machine and container virtualization technologies could not be completed due to lack of data collected for the KVM in this project. Being the main goal of this project and the motivation paper, the possible improvement of the KVM over the years would have been interesting to observe.

**Project Assessment**

Our initial approach for the execution of the project was trying to automate the virtualization technology installation and benchmark execution on the profile. The first step should have been manual installation and testing, gradually moving onto automation - the team lost a lot of time trying to automate the process from the get-go due to unfamiliarity of the deployment and installation infrastructures.

Our failure to implement KVM benchmark testing could be attributed to a lack of devised backup plan. Even though our Kickstarter file worked and automated the installation of the KVM on the node, we were not able to access the node from the inside using the console or the secure shell login. Some changes in our deployment could have solved our problem; possibilities include

* Using a different mounted image
* Specifying SSH parameters during the installation process to allow more comprehensible steps to achieve an SSH connection
* More tests involving manual installation to eliminate parameters that would not affect the troubleshooted problem

Despite our shortcomings, we were able to meet most of the goals that were described in the “Goals” section of this report. We were able to execute the automation of the installation of Docker and Singularity (and KVM, to an extent), run the benchmarks on Docker and Singularity, and collect their data to make comparisons. The profile for CloudLab deployment of the benchmarks and virtualization containers is also available in the form of the team’s GitHub repository. In particular, the team automated the benchmark automation and image building on Docker. Furthermore, the project was able to execute the Stream benchmark, despite it being a recent and sudden change during the project process.

However, the comparison of the performance differences between KVM and containers was the main goal of this project. Failing to achieve this goal significantly reduces the impact and use of this project.