Empirical study of Derivative Clouds

RnD-II Report
by

Shachi Deshpande
(*Roll no.* 140110047)

Supervisor:
Prof. Umesh Bellur

Department of Computer Science & Engineering
Indian Institute of Technology Bombay
Spring Semester, 2017-2018

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I declare that I have properly and accurately acknowledged all sources used in the production of this report. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Shachi Deshpande

Date: 26 April 2018 (*Roll no.* 140110047)

Acknowledgements

I would like to express my sincere gratitude towards my supervisor, Prof. Umesh Bellur and research student, Chandra Prakash for their keen guidance and constant support. Through weekly meetings they helped me in trying out different ideas and setups for the project. Their constant motivation and support has helped me in completing the project enthusiastically. My teammates, Vinay Yogendra and Lohith Ravuru also provided valuable inputs during the weekly discussions on the project.

Contents

Chapter 1

Introduction

1.1 Motivation

Computing resources are virtualized widely nowadays for giving better isolation, security and potentially more efficient use of underlying hardware. It is possible to run multiple virtual machines with different operating systems on a single physical machine, hence reducing the requirement of having a separate physical machine for each instance. This helps in reducing costs for enterprises.

Several technologies are available for achieving this virtualization. In this project, we try to investigate the technology offered by VMware through a suite of products for attaining virtualization. We intend to find out how different set-ups perform after being subjected to combinations of CPU-intensive, memory-intensive and IO-intensive workloads. This will help us understand this framework better, and also understand the scenarios under which a particular virtualization set-up may perform relatively better or worse. We aim to find out configurations of such virtualized set-ups for which certain applications might perform well with regard to certain performance metric, like throughput achieved, or time of completion of certain task. Using these experiments, we try to model the system such that we are able to make certain predictions for its behaviour with different workloads. This will help us in making some estimates about performance of applications running on certain given configuration of resources on this framework. Also, we aim to make certain predictions on resource requirements based on expected performance of an application and its resource utilization profiles.

1.2 Outline

As part of stage two of RnD project, we have investigated IO intensive workloads as continuation of micro-benchmarks (CPU and memory intensive workloads) tested for stage one. We also ran experiments on applications related to Spark, Cloud-computing and Machine Learning. We have run experiments for such workloads on different configurations of virtualization. This report is organized as follows. **Chapter 2** discusses the platform used along with set-ups. **Chapter 3** discusses the experiments over these setups. **Chapter 4** presents some ideas and models of predictions based on experiments discussed in Chapter 3. **Chapter 5** discusses the conclusions and future work

Chapter 2

Setup and Monitoring

In this chapter, we will discuss the general set-up for experiments. For each of the individual experiments, specific details would be added in the corresponding section. Previous work by Sharma et. al[1] mentions some experiments in nesting containers and checking performance of applications on it. We have used the virtualization technology provided by VMware for setting up the experiments. We have used the hypervisor ESXi 6.0 on physical machine. We have used the VMware management and monitoring tools for creation and manipulation of virtual machines on this physical machine

2.1 Hypervisor

A hypervisor is software, firmware or hardware that creates and runs virtual machines.This is installed over a physical machine, over which virtual machines can by created. Hypervisors are categorized into two types. Type-1 hypervisor, also known as bare-metal hypervisor, directly runs on the hardware of the physical machine, without having any other operating system to run over the physical machine. The hypervisor ESXi is a bare-metal hypervisor, or Type-1 hypervisor. Other examples of Type-1 hypervisors are Xen and Microsoft Hyper-V. We will use ESXi hypervisor for our experiments. A Type-2 hypervisor, on the contrary, interacts with an operating system that is already installed on the host machine. Thus, virtual machines with guest OS run as processes on the host machine. Examples of Type-2 hypervisors would be QEMU, VMware workstation, VirtualBox, etc.

2.2 Virtual Machine Management and Monitoring

The creation and manipulation of virtual machines constitutes Virtual Machine Management. Monitoring concerns itself with collecting various statistics from the virtual ma-

chines that are created. VMware provides vSphere vClient [2] for this purpose. VMware provides a vServer which can run on one of the virtual machines. This vServer provides with web-interface called vSphere Web-client[3]. This also gives us the capability to create and manipulate virtual machines along with monitoring capabilities. vClient is a desktop application while web-client can be accessed via any browser. The virtual machines created over host run an operating system which is referred to as guest Operating system. For all the experiments in this report, we have used Photon OS by VMware as guest OS. This OS is described in the succeeding section.

2.3 Photon OS

This is a light-weight open-source operating system optimized to run cloud applications[4]. When Photon OS runs on vSphere virtualization suite provided by VMware, its linux kernel is said to be tuned for better performance. It provides container management and orchestration framworks. It is also claimed to give better security and efficient lifecycle management.

2.4 Containers

Operating system virtualization is where kernel of the OS allows for multiple isolated user-space instances. Containerization is an example of such virtualization. Containers package an application so that it becomes portable across different operating systems. It contains the application along with its run-time environment. Essentially it contains the binary dependencies and libraries too. It runs over an operating system, sharing its kernel. It is different from having a Virtual Machine run on a physical machine, since container does not have its own operating system. This leads to lesser overheads of starting up. In our experiments, we are running such containers over virtual machines, provided by Docker[5]. It is not possible to run containers directly on host machine in our case as the host machine does not have any operating system over it. We have used the Docker framework for creating containers in this project

2.4.1 vSphere Integrated Containers (VIC)

VIC is a container management suite provided by VMware[6]. It is a container runtime which provides for creation and manipulation of containers alongside traditional VMs. We can maintain container registries and also monitor running containers for statistics. We used a different set-up for container monitoring and management. We created

and managed containers directly by logging into the respective VMs. Cadvisor [7] was used for collecting statistics of containers by running the Cadvisor container alongside the other containers to be monitored on a given VM. The data collected through this was redirected to influxDB [8]. Graphana [9] web-interface was used to visualize these statistics collected from containers. This set-up provided us with per second data points, as compared to the the per 20 second data points provided by VIC.

Chapter 3

Experiments

In stage one of RnD project we looked at CPU and memory intensive workloads. In stage two, we also looked at IO intensive workloads, along with few other mixed workloads like CNNs, in-memory analytics from cloudsuite and spark applications as described in following sections. In all the following sections, the experimental data for frameworks other than Photon OS has been taken from the work by my teammates Vinay Yogendra and Lohith Ravuru. Thus the graphs that include comparison of performance across several set-ups other than those of Photon OS is the combined work of us three.

3.1 Micro-benchmarks

3.1.1 IO

Motivation

The questions leading to this experiment are:

1. Which setup is more efficient for IO intensive workloads?
2. As we vary the size of files to be processed, how does the performance of application vary for given configuration of resources at VM and container level?

Benchmark

The benchmark used for this experiment is Filebench [10]. The input to this benchmark is a configuration file which specifies number of threads, size of files to be processed, etc written in Workload Model Language[11].

Table 3.1: Provisioning of CPU's and Memory

Level	CPU	Memory
Host	16	32GB
VM	12	15GB
Container	5	10 GB

Experiment Setup

The resources allotted for this experiment are shown here in **Table 3.1**. The total size of files to be processed is varied as 5GB, 10GB, 15 GB, 20GB, 25GB, 35GB, 40GB. The workload configuration file was written to generate random reads and writes on these sizes of files. The guest OS was Photon OS for this experiment.

Results

The following graph 3.1 of throughput (in terms of operations per second) was obtained for each of the file-sizes mentioned. The experiment did not run to completion for the sizes of 35GB and 40GB.

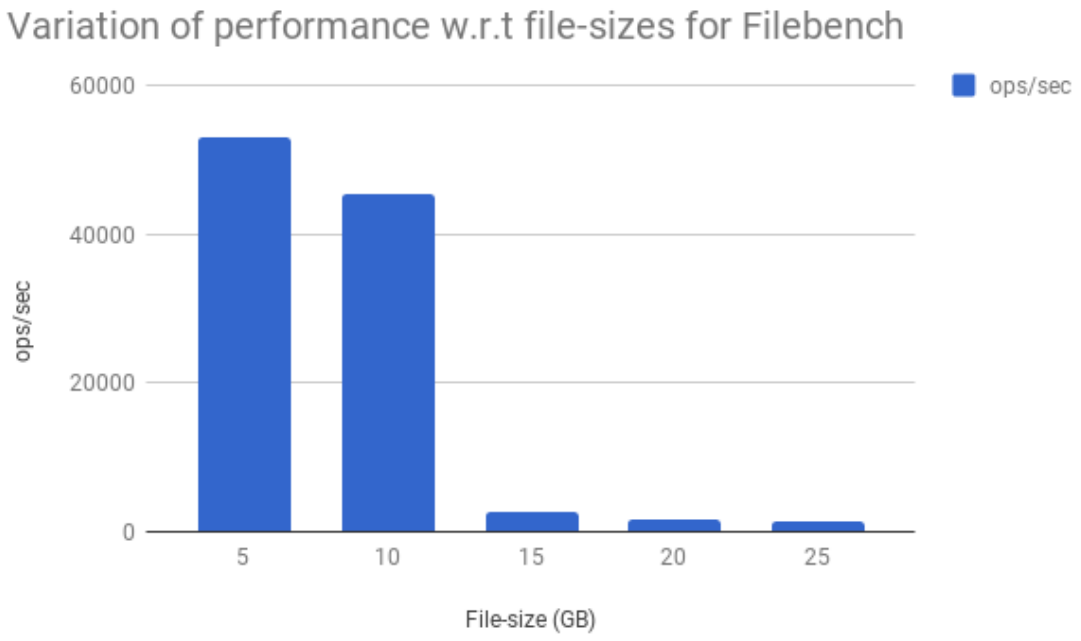

Figure 3.1: Performance of Filebench on Photon OS

We see that the performance falls drastically when the file-size increases beyond 10GB, which also happens to be the memory limit of container. Beyond 15GB, which

is memory limit of the VM, we don't see a lot of difference, though the operations per second fall monotonically as the file size is increased.

3.2 Mixed Applications

3.2.1 CNN

Motivation

After looking at microbenchmarks, now we move to a Machine Learning benchmark which uses Convolutional Neural Network (CNN). We would like to see how this benchmark performance varies with degree of overcommitment on CPUs, and compare it with similar study made on Sysbench benchmark earlier.

The degree of over-commitment (DOC) w.r.t CPU is defined as the ratio of number of CPU's allotted collectively to all containers and the number of CPU's actually available to the VM on which those container are running.

Benchmark

We use a simple CNN over the CIFAR10 dataset [12] of images to classify those images. This work-load is supposed to be primarily CPU-intensive when it trains the images, along with some usage of memory. The training times for the same work-load are recorded as we increase the number of containers or applications running in parallel to vary the degree of overcommitment of the CPU.

Experiment Setup

The resource allocation to the system is shown in the **Table 3.2**. We vary the number of containers running in parallel as 1,3,5,7.

Table 3.2: Provisioning of CPU's and Memory

Host	**VM**	**Container**
16 CPUs	6 CPUs	6 CPUs
32GB RAM	16GB RAM	2GB

Results

I ran this experiment on Photon OS to get the following graph 3.2 showing training times. Naturally as the number of containers running in parallel increases, the completion times

increase, since this corresponds to increasing degree of overcommitment of CPUs. However surprisingly, the completion times of the benchmark as application is seen to be lower than the same benchmark running as container. Similar result was obtained in stage one of the project for microbenchmark of sysbench (CPU-intensive).

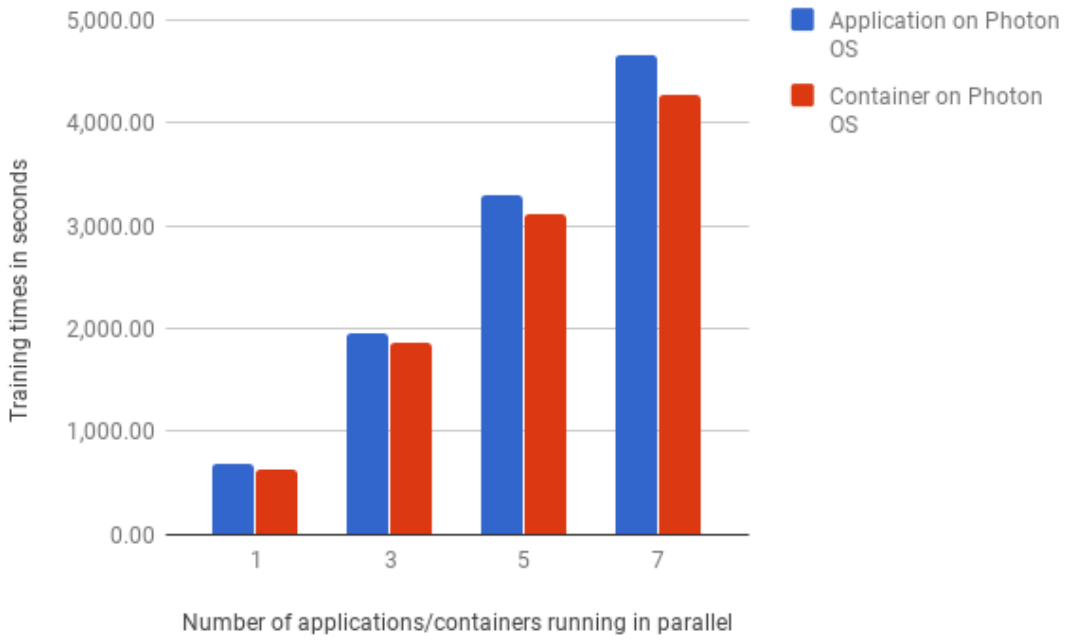

Figure 3.2: Performance of CNN on Photon OS

We also add a graph 3.3 which shows the comparison of performance across all other set-ups used so that the performance of Photon OS can be compared against other set-ups as well.

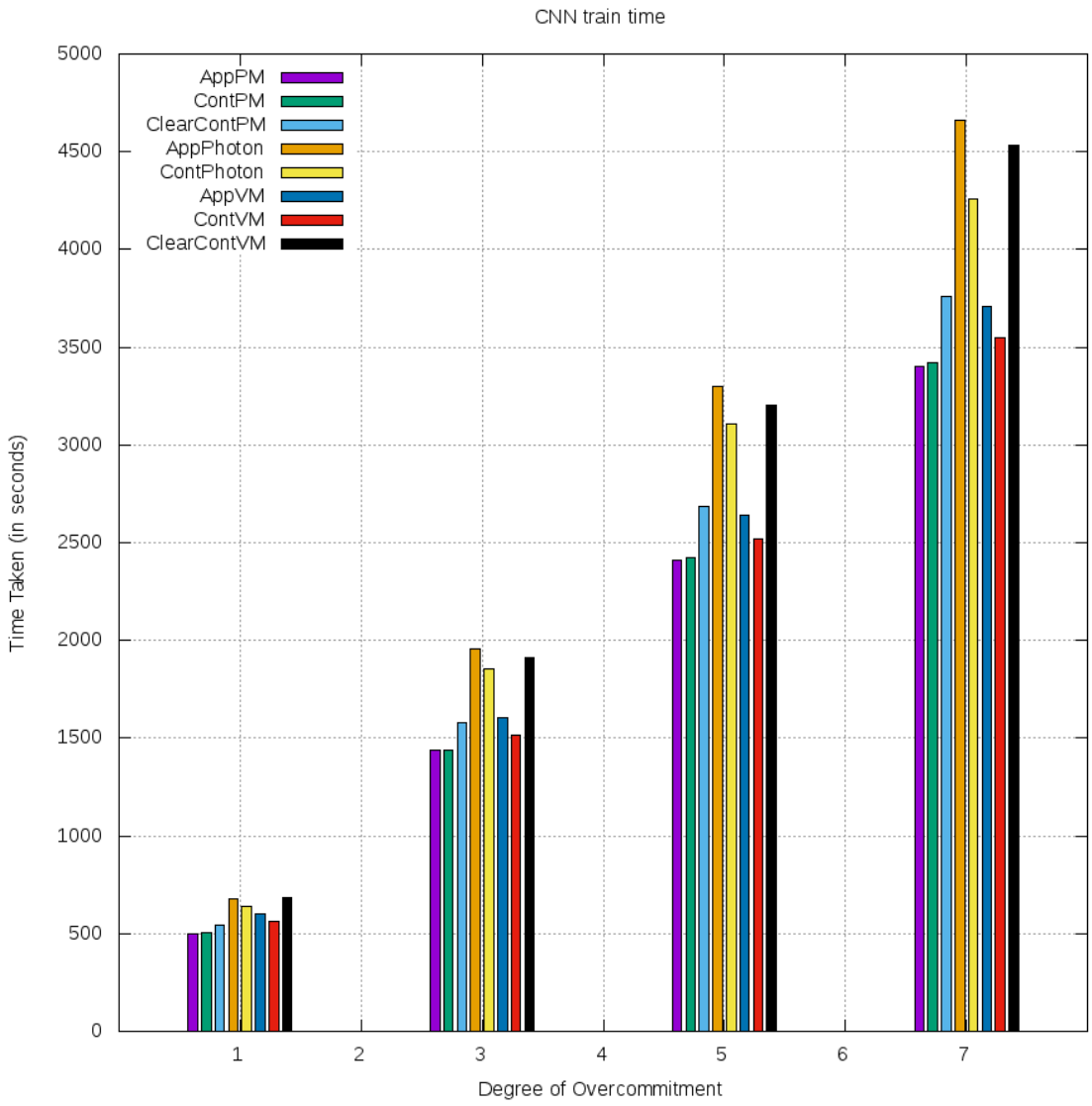

Figure 3.3: Comparison of performance on CNN across all set-ups

ClearCont stands for clear-container technology, App for application and Cont for container

Here, the performance of container on a Linux VM deployed on Linux PM is seen to be the best, and this benchmark running as application on Photon OS seems to perform the worst in comparison to the set-ups shown here.

3.2.2 In-memory cloudsuite

Motivation

This experiment uses an application making computations in-memory. Thus it is supposed to primarily use both CPU and memory.

Benchmark

This benchmark [13] is called in-memory analytics and is part of cloudsuite benchmarks. It uses spark framework and uses filtering algorithms in-memory over a dataset of movie reviews to give movie recommendations in the end. The completion time of this algorithm is the metric to be measured for this experiment.

Experiment Setup

The resource allocation to the system is shown in the table **Table 3.3**. We vary the number of containers running in parallel as 1,3,5,7.

Table 3.3: Provisioning of CPUs and Memory

Host	**VM**	**Container**
16 CPUs	12 CPUs	12 CPUs
32GB RAM	30GB RAM	4GB

Results

I ran this experiment on Photon OS to get the following graph 3.4 showing completion times of the benchmark to finally print the movie recommendations. Naturally as the number of containers running in parallel increases, the completion times increase, since this corresponds to increasing degree of overcommitment of CPUs. The total memory allotted to the VM is larger than the total memory allotted to all containers together in every experiment here. So memory is not overcommitted. Also, the increase in completion times is linear and there is no breaking point within the range of 1 to 7 containers running in parallel.

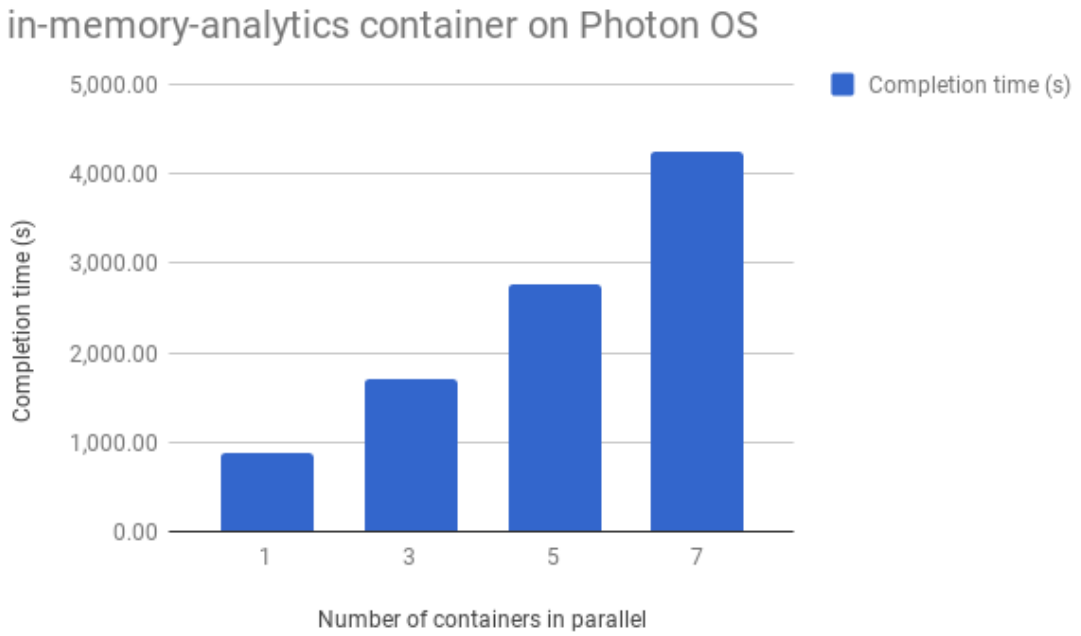

Figure 3.4: Performance of in-memory analytics on Photon OS

We also add a graph 3.5 which shows the comparison of performance across all other set-ups used so that the performance of Photon OS can be compared against other set-ups as well.

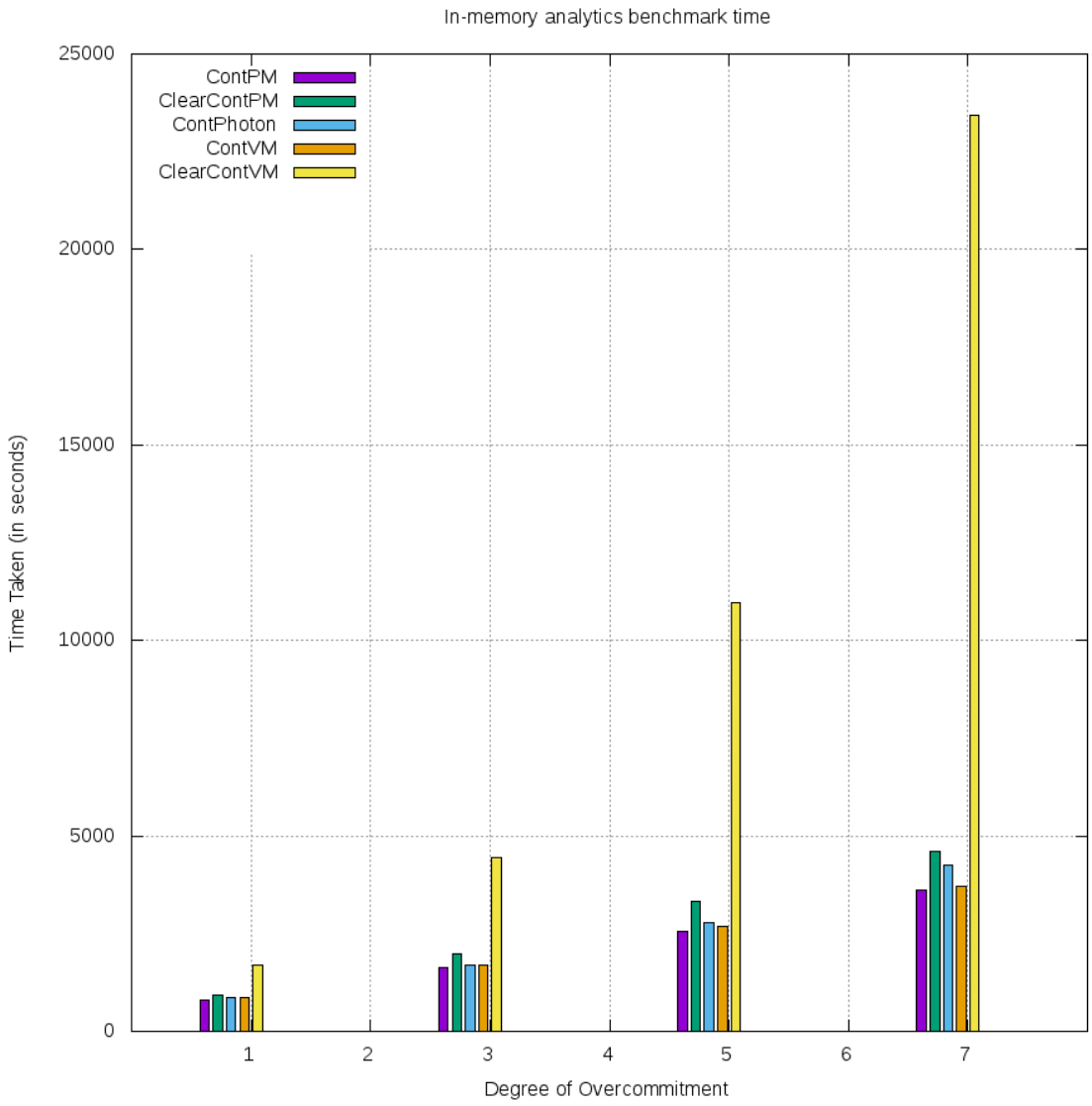

Figure 3.5: Comparison of performance on in-memory analytics across all set-ups

Here, the performance of container on Photon OS as shown under heading 'Cont-Photon' in the graph is better when 1 or 3 containers are running in parallel. However as the overcommitment is further increased, container running on Linux PM as shown under the heading 'ContPM' seems to perform better. Thus, Photon OS could be better if the overcommitment of CPUs is low.

3.2.3 Spark count and sort applications

Motivation

The previous experiment of cloudsuite benchmark uses Spark framework, and in this experiment we use some simple applications using Spark. These applications of counting and sorting words can serve as baseline for more complex experiments using Spark.

Benchmark

These experiments use spark benchmark [14]. We have written two simple python files which count the number of words in a text file and sort the words in a text file respectively, using Spark commands. The spark framework is supposed to divide this task over multiple worker threads and use map-reduce paradigm for the task. Spark makes most of the computations in-memory when compared to the hadoop framework. Thus the IO utilization is expected to be less while primary resources being used would be CPU and memory. We have to provide text files as input to the benchmark.

Experiment Setup

The resource allocation to the system is shown in the **Table 3.4**. We vary the number of containers running in parallel as 1,3,5,7. The same text-file of 2400M size was used as input for all the running containers. After completion of the count or sort task, the completion times can be recorded as metric of performance.

Table 3.4: Provisioning of CPUs and Memory

Host	**VM**	**Container**
16 CPUs	6 CPUs	(no limit on CPUs)
32GB RAM	30GB RAM	4GB

Results

I ran the count benchmark on a container on Photon OS to get the following graph 3.6 showing completion times of the benchmark to finally get the word counts. Also, the graph 3.7 shows completion times for the sorting benchmark running on container on Photon OS. Naturally as the number of containers running in parallel increases, the completion times increase, since this corresponds to increasing degree of overcommitment of CPUs.The total memory allotted to the VM is larger than the total memory allotted to all containers together in every experiment here. (just like the experiment for in-memory-analytics) So memory is not overcommitted.

In the result for Spark count, we have reported the job 0 completion times (as there is a single job for this benchmark). In the result for Spark sort, there were two different jobs (which correspond to the different jobs run by Spark corresponding to some internal commands).

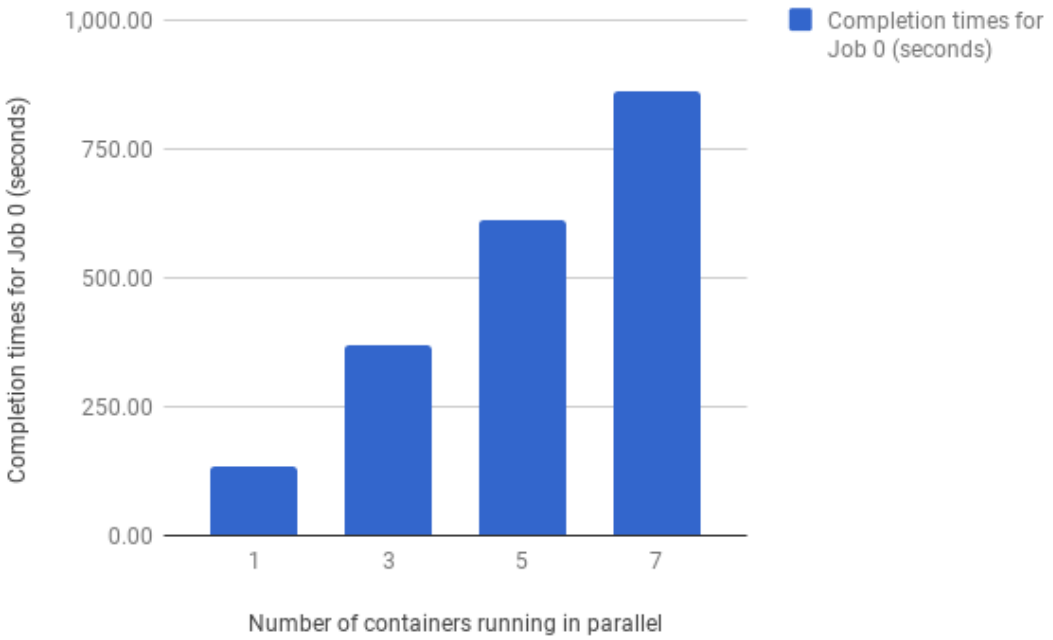

Figure 3.6: Performance of Spark count on container on Photon OS

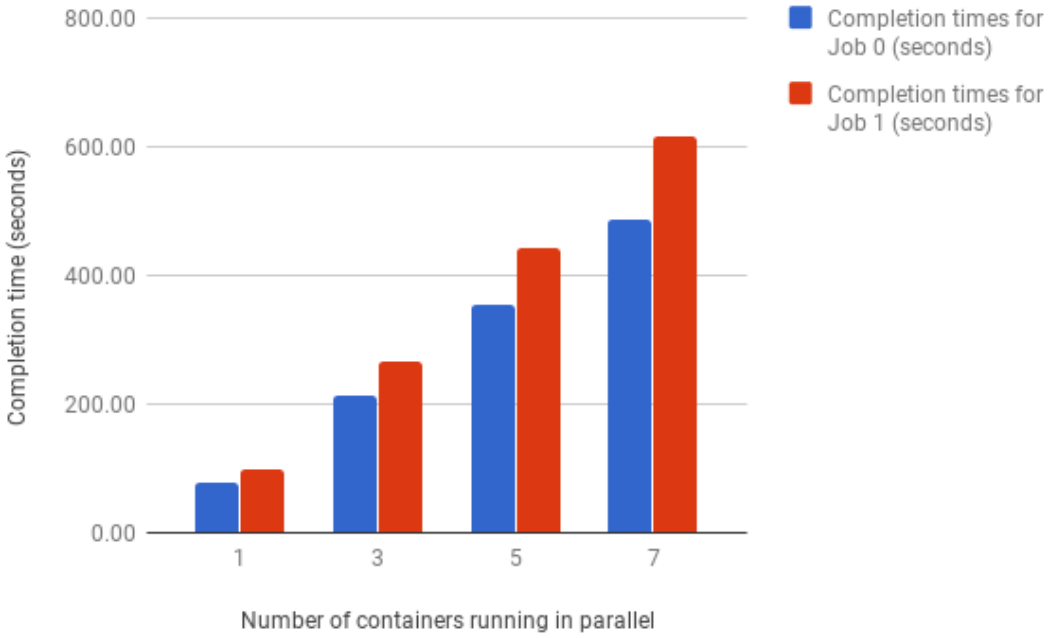

Figure 3.7: Performance of Spark sort on container on Photon OS

We also add a graph 3.8 which shows the comparison of performance for job 0 of Spark sort across all other set-ups used so that the performance of Photon OS can be compared against other set-ups as well. Similarly, the next graph 3.9 compares the performance for spark sort on job 1.

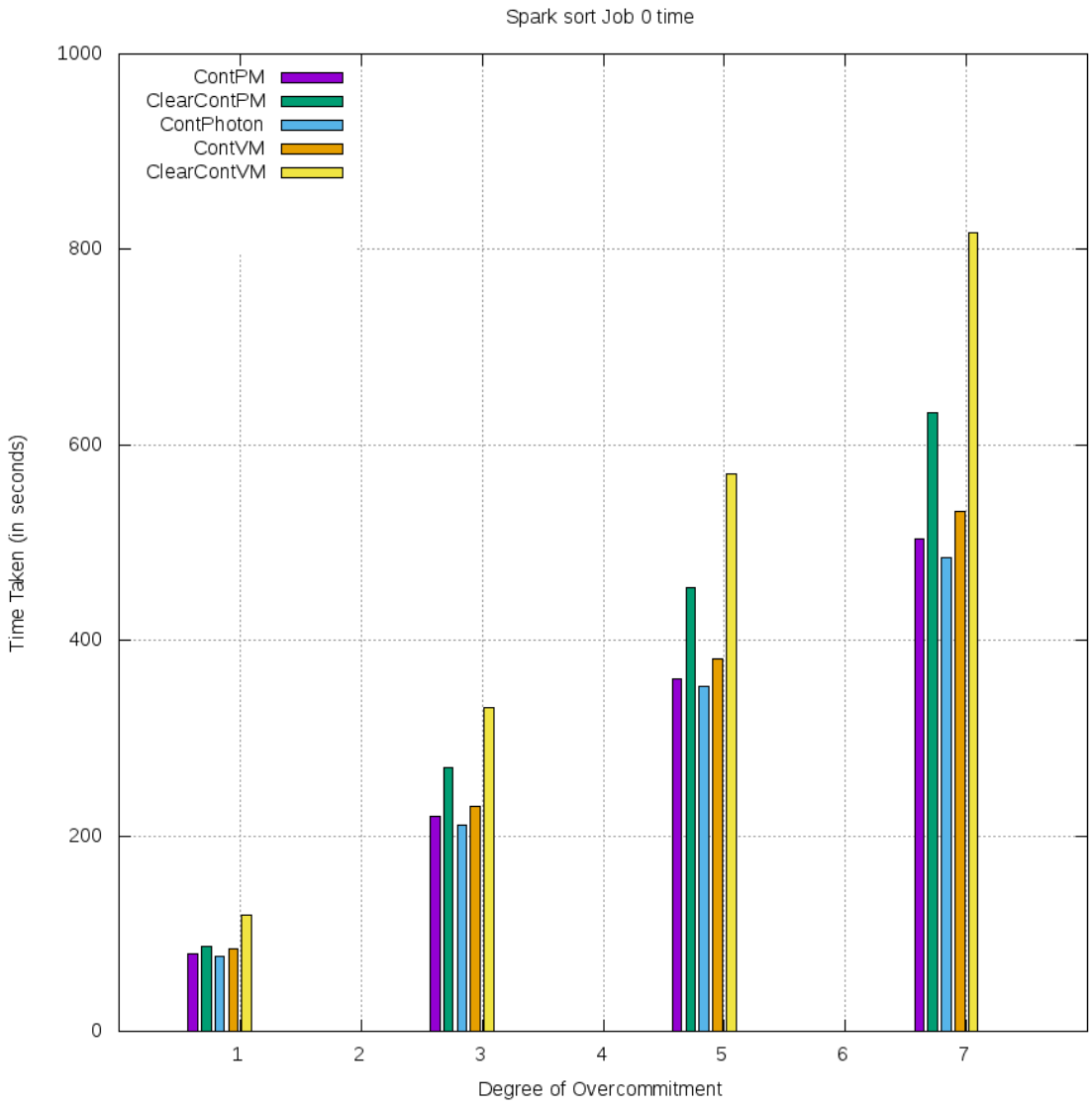

Figure 3.8: Comparison of performance on Spark sort across all set-ups for job 0

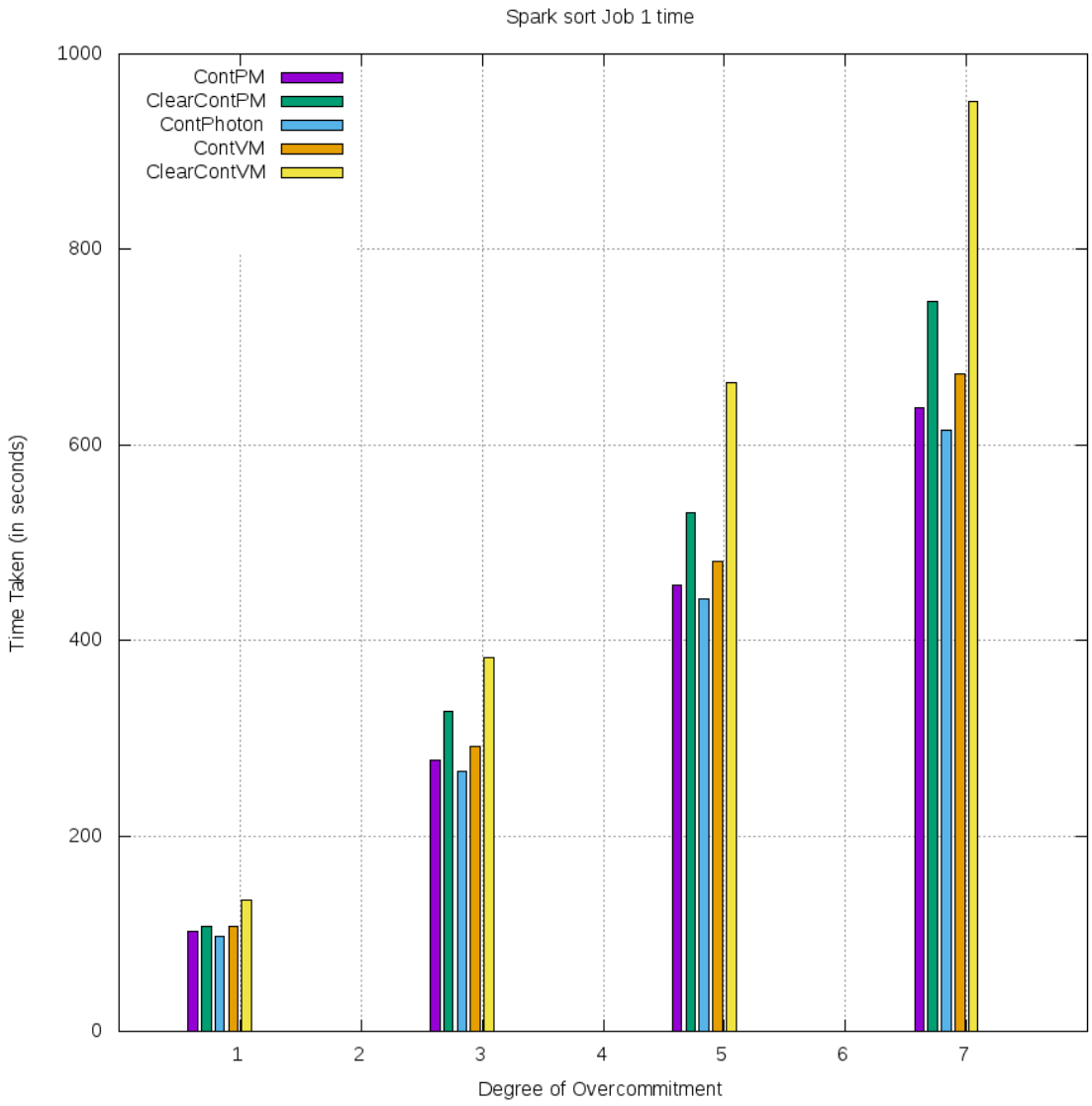

Figure 3.9: Comparison of performance on Spark sort across all set-ups for job 1

Here, the performance of container on Photon OS as shown under heading 'Cont-Photon' in both the graphs is better in terms of completion times, in comparison to the other set-ups shown. Clear containers on VM as indicated by 'ClearContVM' seem to perform worse.

Before closing this part, we also see how the performance of spark count varies with input file-size. The following graph 3.10 shows the container completion times (including total of job 0 times and times for starting/shutting down spark) while we increase the file size of input text-file from 200MB to 2000MB, for the Spark count application. Naturally as file-size increases, the completion times of the container increase. For this experiment, the memory limit on any single container running the spark code was 1GB.

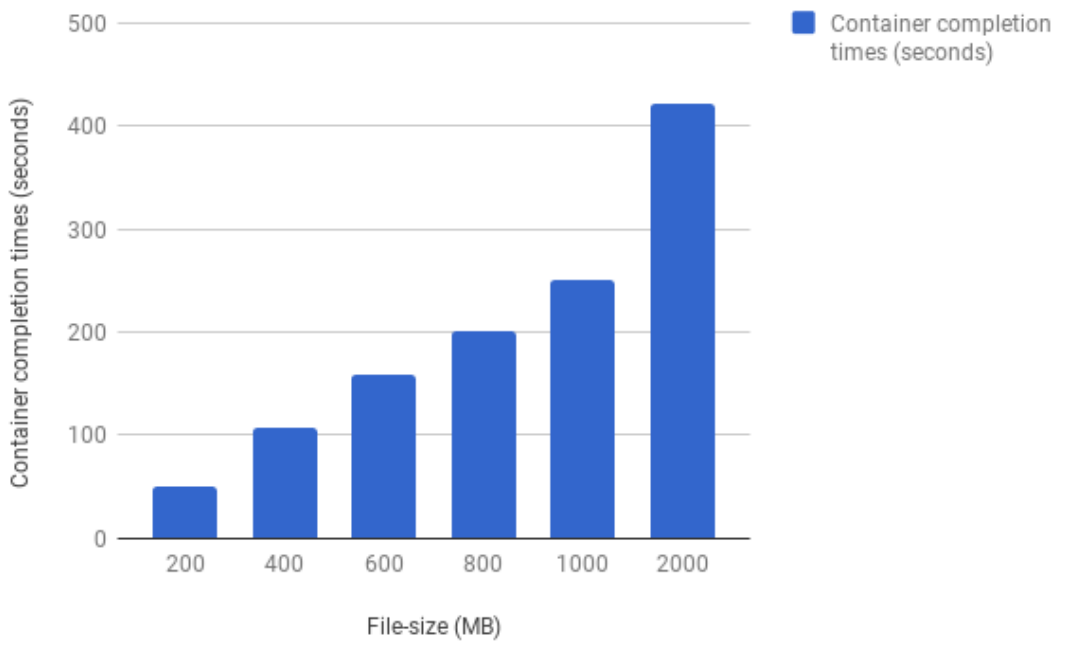

Figure 3.10: Performance of Spark count on container on Photon OS with increasing file-size

Chapter 4

Prediction Models

Using the data from all the previous experiments on microbenchmarks and mixed benchmarks, we hope to model our system in such a way that given some throughput requirements by the user, we should be able to give the resource requirements for the system to produce that throughput. Similarly we can model other things like predicting resource requirements for some other set-up to give same throughput as in current set-up and so on. The work on prediction models is preliminary at this stage. But this can be used to make further extensions to the model.

We begin by considering the experiments done in CNNs and sysbench, where the degree of overcommitment was varied by increasing the number of applications or containers running in parallel. Since the performance metric here was just the completion time, we can take $\frac{1}{completionTime}$ as an estimate of throughput of the application.

4.1 Prediction model for CNN application

For each degree of overcommitment, we can estimate the percentage of total CPUs on VM available to a single container. Thus we can plot the throughput ($\frac{1}{completionTime}$) vs pecentage of CPU utilized. This gives us the some graphs across set-ups. We can use these graphs to make some prediction about the percentage of CPU required for a container on the same VM, if we want to achieve a particular throughput. The following graph 4.1 was obtained by this method. Using the existing data-points, we get approximately linear fit for the curve using linear regression.

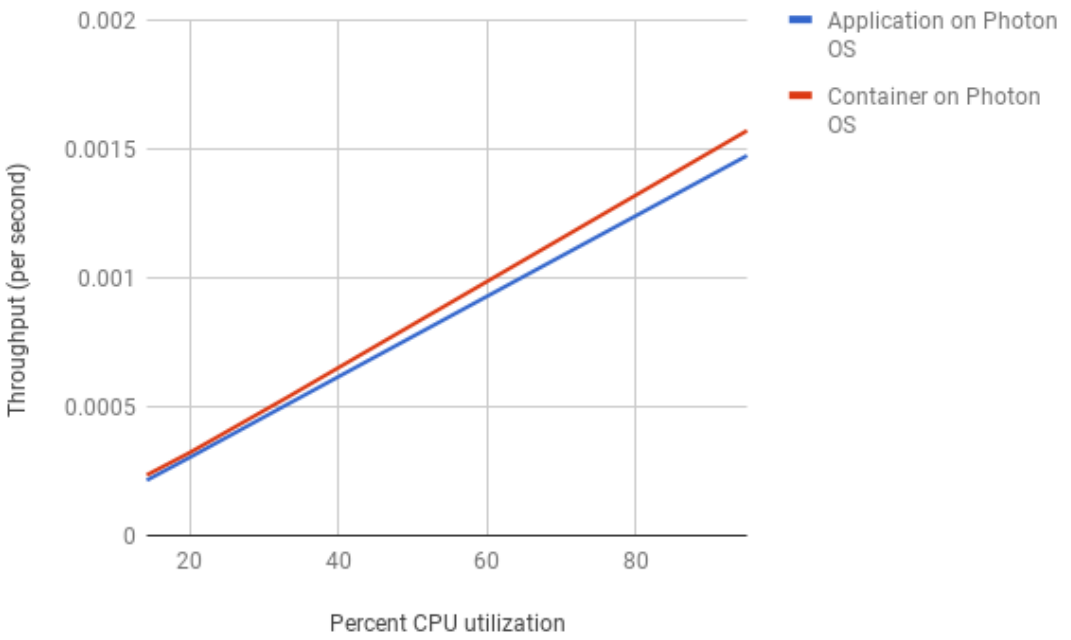

Figure 4.1: Throughput vs CPU utilization graphs for CNN

4.1.1 Intra-Setup

We use the existing data-points at the CPU utilizations of 95%,33.33% and 14.2% to interpolate throughput at CPU utilization of 20%. For my set-up, CNN container on Photon OS is tested for this utilization and the actual value of completion time was noted to be 3,106.67s. The interpolated value was noted to be 3057.44s . Thus, the model indeed seems to be linear for this data, as the error was about -1.5%. Similarly, for the CNN application, the interpolated value was noted to be 3283.34s while the actual value was 3,296.17s (error of -0.39%).

Thus for predictions within the set-up, using linear regression over existing data-points gave a fairly good estimate of completion time given certain resource availability. The CNN workload seems to be CPU-intensive with almost constant use of memory which does not become bottleneck. There is almost no IO utilization as well.

4.1.2 Inter Setup

Now we try to do similar predictions across different frameworks. Here, we plot completion times of CNN as application on Photon OS vs the one for application on Linux PM. So similar to the previous case, we take data points corresponding to different CPU utilizations and find the completion time for Photon OS given the completion times on Linux VM for the same container. The corresponding graph 4.2 shows such a plot.

So for 20% utilization of CPU, we get an interpolated value of 3294.79s for Photon OS after using linear regression on the data points. The actual value of completion time is 3,296.17s. Thus the error is negligible (< 0.1%).

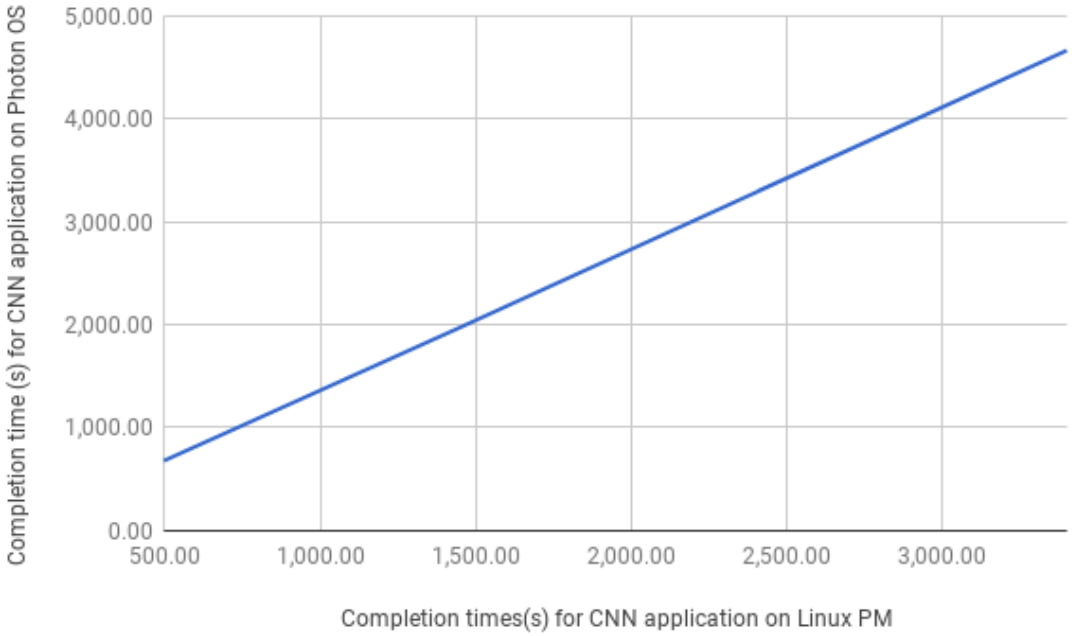

Figure 4.2: Completion times for CNN Application on Photon OS vs Completion times for CNN Application on Linux PM

Similarly, we also do this analysis to predict the completion time for CNN as container on Photon VM using the data for CNN as container on Linux VM. For 20% utilization of CPU, we get an interpolated value of 3034.19s for Photon OS if we use linear regression on the data points once again. The actual value of completion time is 3,106.67s. Thus the error is about -2.3%. The corresponding graph 4.3 is below.

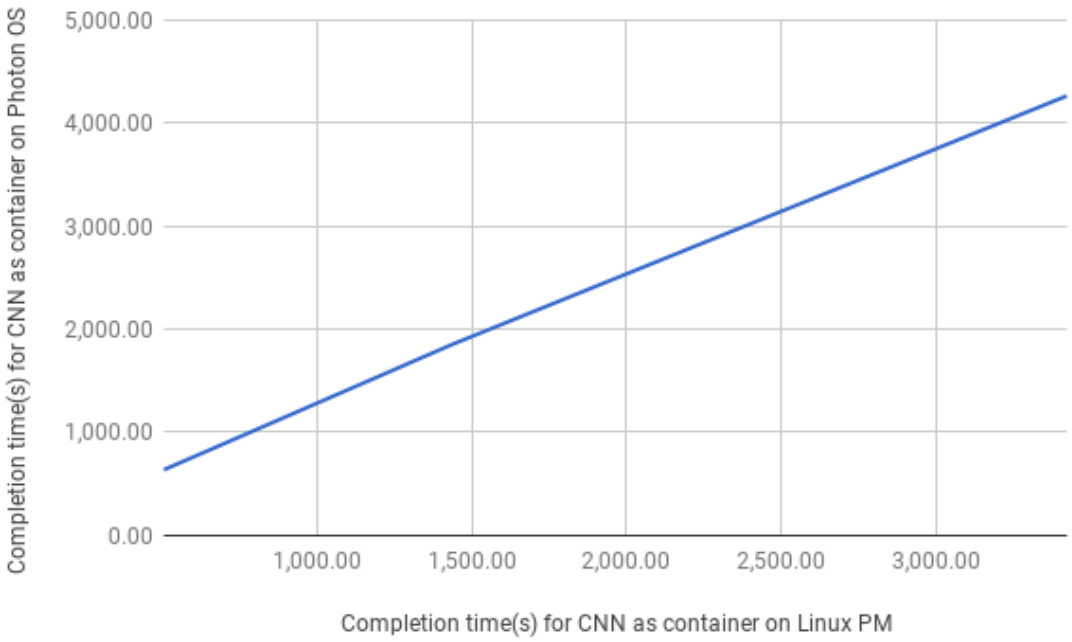

Figure 4.3: Completion times for CNN container on Photon OS vs Completion times for CNN container on Linux PM

Both these observations seem to suggest that similar linear regression model works for making estimates of throughput for Photon OS using data for corresponding set-up on Linux PM.

4.2 Modelling Mixed Application from Micro-Benchmarks

4.2.1 Motivation

If the user gives us resource utilizations and throughputs achieved for any application on his base set-up (which might be similar to having application or container running on a PM), we might want to know the resource utilization overheads for having similar performance on derivative set-ups we have on cloud. We might also want to know how much throughput can be achieved on derivative set-up if we want to have the same resoource utilizations as the user had.

In the previous section we saw that for Photon OS, linear regression fit was able to give these results with reasonable accuracy (errors below 5%). We also try to get some predictions using the micro-benchmark data that we have.

For using micro-benchmark data, we have to classify our mixed benchmarks as using certain combination of resources. For example, spark workloads use CPU and memory, with some IO. So the experiments on microbenchmarks for CPU, memory and IO intensive workloads can be used to make predictions for spark workload, and we will have to model the exact dependence of these predictions on the data obtained for these three benchmarks. We have still not explored this direction fully, but we investigate similar approach for CNN.

4.2.2 CNN and Sysbench

CNN uses CPU and memory, however CPU utilization goes to 100% as the degree of overcommitment of CPU rises. The memory is not a bottleneck. Sysbench, on the other hand, is completely CPU-intensive benchmark.

Using data-points for Sysbench that we already had from stage one of the project report, we try to predict the completion times for CNN on the same set-up on Photon OS. Using the data-points for CPU utilizations of 95%, 33.3% and 14.2%, we predict the completion time of CNN for CPU utilization of 20% using the value already available for the Sysbench application (again using linear regression).

So for CNN as application, we interpolate values from Sysbench as application on the same set-up and we get an estimated value of 3330.02s. The actual value of completion time from experiments is 3,296.17s. Thus, the error is 1%. The corresponding graph 4.4 is shown below.

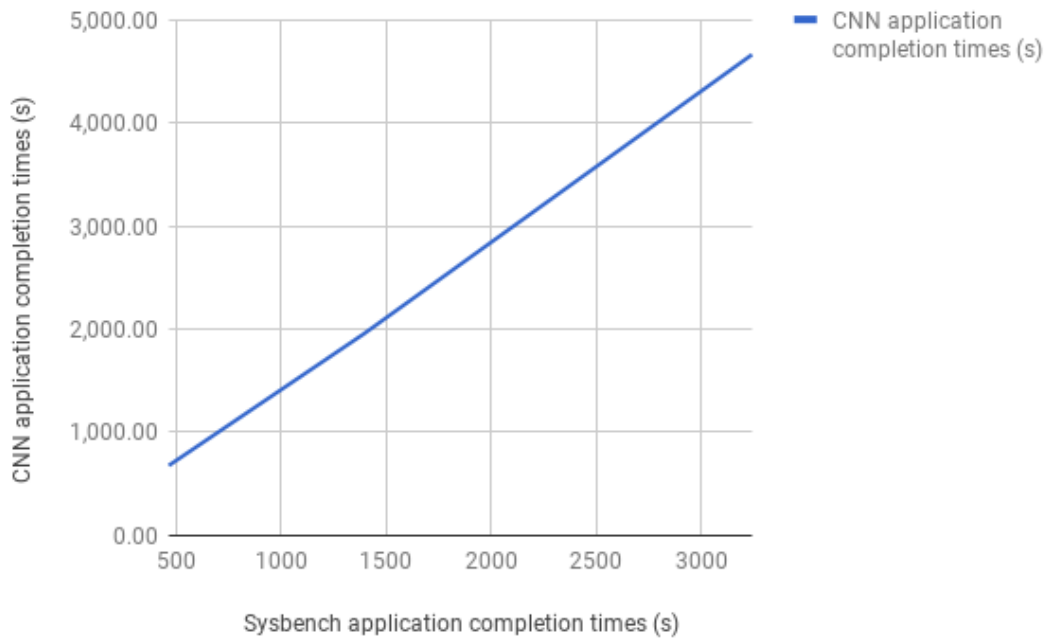

Figure 4.4: Completion times of CNN application on Photon OS vs Sysbench application on Photon OS

Similarly for CNN as container, we interpolate values from Sysbench as container on the same set-up and we get an estimated value of 3061.69s. The actual value of completion time from experiments is 3,106.67s. Thus, the error is -1.5%. The corresponding graph 4.5 is shown below.

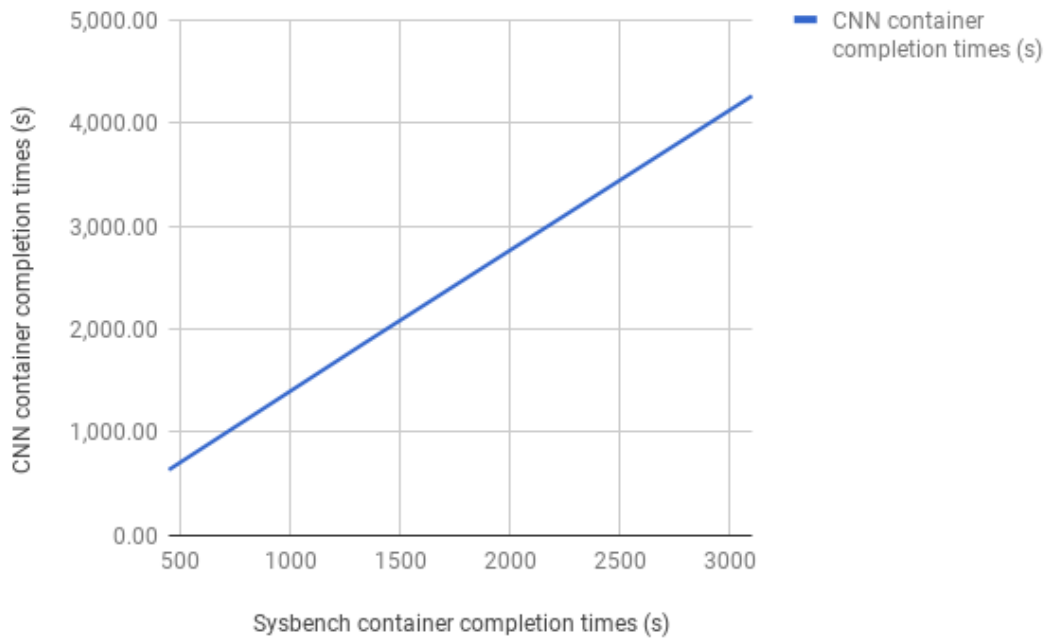

Figure 4.5: Completion times of CNN container on Photon OS vs Sysbench container on Photon OS

Thus, even after using microbenchmark of Sysbench to make predictions about completion times of CNN workloads, we saw that linear regression fit gave reasonably accurate results (error less than 5%).

Chapter 5

Conclusions & Future Work

We conducted experiments on some mixed workloads in this project after completing similar experiments for microbenchmarks. We tried to understand the trends in application performance of the benchmarks themselves and then tried to devise some models to make predictions about application performance and resource requirements. We could make the following conclusions out of this study.

1. Completion times for CNN as application were found lower than those for CNN as container, similar to the result obtained for Sysbench on Photon OS. We need to further investigate the reasons for this.

2. Increasing overcommitment of CPU led to linear increase in completion times of applications for the benchmarks covered in this project

3. Performance of Photon OS (in terms of completion time) was better than other frameworks in case of Spark sort application. For CNN and in-memory-analytics benchmark, the performance of Photon OS framework was worse.

4. Using linear regression (over throughput vs CPU utilization data collected intra-setup on Photon OS), we were able to estimate throughput (in terms of completion time) given some CPU resource utilization for CNN workloads with reasonable accuracy (error below 5%)

5. We used similar regression model over throughput vs CPU utilization data across set-ups using same resource configuration. Thus we could also estimate the throughput (in terms of completion time) for some CPU utilization for CNN workloads using this inter-setup data . We could predict the throughput for CNN on Photon OS using corresponding values on Linux PM, again with error below 5%.

6. We also predicted the throughput(in terms of completion time) for CNN for some CPU utilization using micro-benchmark of Sysbench on same set-up with error below 5% (again using similar model of linear regression over available data-points). Bottleneck resource for CNN had been CPU even though it used some memory and negligible IO. However we were able to make predictions for CNN using data of CPU-intensive micro-benchmark of Sysbench alone. We need to verify if this works for other applications where certain resource is a bottleneck while other resources are also being used without bottlenecking anything.

These observations also suggest that linear regression worked reasonably well for making predictions for CNN workload, using experimental data on micro-benchmarks, or data collected either intra-setup or inter-setup. However we still need to check if this holds true for other kinds of applications. Specifically, we need to perform similar prediction experiments for in-memory analytics application and spark applications discussed in this report. This may give us more insights into the accuracy of the linear regression model that seemed to work well for CNN. We can also explore more applications as part of future work to make similar predictions.

References

[1] Prateek Sharma, Lucas Chaufournier, Prashant J. Shenoy, and Y. C. Tay. Containers and virtual machines at scale: A comparative study. In *Middleware*, 2016.

[2] vClient. http://pubs.vmware.com/vsphere-4-esxi-installable-vcenter/index.jsp?topic=/com.vmware.vsphere.gsinstallable.doc_41/common/install/t_down_client.html.

[3] vSphere Web Client. https://blogs.vmware.com/vsphere/2015/02/vsphere-6-web-client.html.

[4] Project Photon. https://vmware.github.io/photon/.

[5] Docker Containers. https://www.docker.com/what-docker.

[6] vSphere Integrated Containers. https://github.com/vmware/vic.

[7] cAdvisor. https://github.com/google/cadvisor.

[8] InfluxDB container. https://hub.docker.com/_/influxdb.

[9] Graphana Container. https://github.com/grafana/grafana-docker.

[10] Filebench benchmark. https://github.com/filebench/filebench/wiki.

[11] Workload Modelling Language. https://github.com/filebench/filebench/wiki/Workload-model-language.

[12] CIFAR 10 dataset. https://keras.io/datasets/.

[13] In-memory-analytics container. https://hub.docker.com/r/cloudsuite/in-memory-analytics/.

[14] Spark container. https://hub.docker.com/r/gettyimages/spark/.